



HAL
open science

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.science/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 function, leads to factored-form expressions with the least number of boolean operators and the least number of literals (considering also the outputs of previous operators) as operands of those operators. This factored-form representation is interesting because it approximates the complexity of a gate-level implementation [10]. The only constraint we have is the encoding of the ternary values, given by $\mu : \{-1, 0, 1\} \mapsto \{11, 00, 01\}$. This choice is appropriate for use in classical two's complement arithmetic circuits, for instance when these values are directly fed into multipliers or adders [9].

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 $\log_2(3) \approx 1.586$ 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 $\lceil \log_2(3^n) \rceil = \lceil n \log_2(3) \rceil$ bits are necessary to

O. Muller and F. Pétrot are with Univ. Grenoble Alpes, Grenoble-INP (Institute of Engineering Univ. Grenoble Alpes), TIMA, F-38000 Grenoble, France (e-mail: {olivier.muller, frederic.petrot}@univ-grenoble-alpes.fr).

A. Prost-Boucle was with Univ. Grenoble Alpes, TIMA, F-38000 Grenoble, France. He is now with Synopsys, 38330 Montbonnot-Saint-Martin, France (e-mail: adrien.prost-boucle@synopsys.com).

A. Bourge was with Univ. Grenoble Alpes, TIMA, F-38000 Grenoble, France. He is now with Atos, 38130 Échirolles, France (e-mail: alban.bourge@atos.net).

TABLE I: Gain and free values for encoding trits on bits

trits	bits	gain	free values ($2^{bits} - 3^{trits}$)
1	2	0.00%	1
2	4	0.00%	7
3	5	16.67%	5
4	7	12.50%	47
5	8	20.00%	13
6	10	16.67%	295
7	12	14.29%	1909
8	13	18.75%	1631
9	15	16.67%	13085
10	16	20.00%	6487
17	27	20.58%	5077565

encode this sequence in binary. For n trits the gain compared to the non encoded sequence is given by $u_n = \frac{2n - \lceil n \log_2(3) \rceil}{2^n}$. By definition $x \leq \lceil x \rceil < x + 1$, hence

$$\underbrace{\frac{2n - n \log_2(3)}{2^n}}_{v_n} \geq u_n > \underbrace{\frac{2n - (n \log_2(3) + 1)}{2^n}}_{w_n}$$

Yet $\forall n \in \mathbb{N}, v_n = 1 - \frac{\log_2(3)}{2}$ and $\lim_{n \rightarrow \infty} w_n = 1 - \frac{\log_2(3)}{2}$. Since v_n and w_n increase monotonically, this yields by the squeeze theorem $\lim_{n \rightarrow \infty} u_n = 1 - \frac{\log_2(3)}{2} \approx 0.2075$.

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 2^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 $2^{b \cdot 3^t} = \frac{2^{b!}}{(2^b - 3^t)!}$ of them, where n^m represents the falling factorial. We focus on two particular instances of the problem, the mapping of 3 trits on 5 bits, leading to $3 \cdot 2^{27} \approx 2.2 \cdot 10^{33}$ possible mappings, and the mapping of 5 trits on 8 bits, leading to $256^{243} \approx 1.4 \cdot 10^{497}$ 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 unpractical.

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.

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 were 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 (λ). 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 μ 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 ≈ 77.4 with a standard deviation of ≈ 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 ≈ 886.8 with a standard deviation of ≈ 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, a ASIC synthesis on the chosen pre-characterized library of logic cells requires at least a minute.

The solutions presented below were 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:

$t_5 \dots t_0$	$b_4 \dots b_0$	$t_5 \dots t_0$	$b_4 \dots b_0$
000000	-0001	000101	10000
000001	-0010	110101	10011
000011	-0110	000111	10100
000100	00-00	110100	10101
001100	01-00	110111	10111
010101	00011	001101	11000
010100	00101	110000	11001
010111	00111	110001	11010
010000	01001	111101	11011
010001	01010	001111	11100
011101	01011	111100	11101
011100	01101	110011	11110
010011	01110	111111	11111
011111	01111		

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 ($b_4.t_4$, $t_4.b_3.t_2$, $t_2.b_2.t_2$, 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 $(0, 1, b_4)$. Similarly, the third set contains the 6 codes corresponding to the magnitude vector $(b_3, 0, 1)$. The last set contains the last 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 $b_1 b_0$.

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

$$\begin{aligned} x_0 &= b_0 (b_1 + b_2), & x_1 &= b_0 + b_1 \\ t_0 &= b_1 + \overline{b_0} b_4, & t_1 &= t_0 b_2 \\ t_2 &= x_0 + \overline{x_1}, & t_3 &= t_2 b_3 \\ t_4 &= x_0 + b_3 x_1, & t_5 &= t_4 b_4 \end{aligned}$$

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 $6.52 \mu\text{m}^2$, (11 gates instantiated), and a propagation time of 41 ps.

B. 8 bits to 5 trits

Again, denoting $t_{9..0}$ the binary encoded ternary values and $b_{7..0}$ the binary codes, our best assignment solution is:

$t_9 \dots t_0$	$b_7 \dots b_0$	$t_9 \dots t_0$	$b_9 \dots b_0$
0100010101	00000000	0111111111	01111111
0101000001	00000001	1100010101	10000000
0101000000	00000010	1101000001	10000001
0101010100	00000011	1101000000	10000010
0101000101	00000100	1101010100	10000011
0101010001	00000101	1101000101	10000100
0101010000	00000110	1101010001	10000101
0101010101	00000111	1101010000	10000110
0001010101	00001000	1101010101	10000111
0001000000	000-1001	0011010101	10001000
0100010100	00001010	0011000000	100-1001
0001010100	00001011	0000010101	10001010
0100000101	00001100	0001010001	10001011
0001000001	00001101	1100000101	10001100

t_9	...	t_0	b_7	...	b_0	t_9	...	t_0	b_9	...	b_0
0000010100			-00011110			0011000001			10001101		
0101010111			00001111			1101010111			10001111		
0100010111			00010000			1100010111			10010000		
0101000011			00010001			1101000011			10010001		
0100000100			0-010010			1100000100			1-010010		
0101011100			00010011			1101011100			10010011		
0101000111			00010100			1101000111			10010100		
0101010011			00010101			1101010011			10010101		
0101000100			00010110			1101000100			10010110		
0101011101			00010111			1101011101			10010111		
0001010111			00011000			0011010111			10011000		
1100010100			00011010			0000010111			10011010		
0001011100			00011011			0001010011			10011011		
0100000111			00011100			1100000111			10011100		
0001000011			00011101			0011000011			10011101		
0000011100			-00111110			1101011111			10011111		
0101011111			00011111			1100011101			10100000		
0100011101			00100000			1100010001			10100001		
0100010001			00100001			1100010000			10100010		
0100010000			00100010			1101110100			10100011		
0101110100			00100011			1101001101			10100100		
0101001101			00100100			1101110001			10100101		
0101110001			00100101			1101110000			10100110		
0101110000			00100110			1101110101			10100111		
0101110101			00100111			0011011101			10101000		
0001011101			00101000			0000000001			10101001		
0100000000			00101001			0000011101			10101010		
0100011100			00101010			0001110001			10101011		
0001110100			00101011			1100001101			10101100		
0100001101			00101100			1100000001			10101101		
0100000001			00101101			1101110111			10101111		
0000110100			-01011110			1100011111			10110000		
0101110111			00101111			1100010011			10110001		
0100011111			00110000			1100001100			1-110010		
0100010011			00110001			1101111100			10110011		
0100001100			0-110010			1101001111			10110100		
0101111100			00110011			1101110011			10110101		
0101001111			00110100			1101001100			10110110		
0101110011			00110101			1101111101			10110111		
0101001001			00110110			0011011111			10111000		
0101111101			00110111			0000000011			10111001		
0001011111			00111000			0000011111			10111010		
1100000000			00111001			0001110011			10111011		
1100011100			00111010			1100001111			10111100		
0001111100			00111011			1100000011			10111101		
0100001111			00111100			1101111111			10111111		
0100000011			00111101			1100110101			11000000		
0000111100			-01111110			1111000001			11000001		
0101111111			00111111			1111000000			11000010		
0100110101			01000000			1111010100			11000011		
0111000001			01000001			1111000101			11000100		
0111000000			01000010			1111010001			11000101		
0111010100			01000011			1111010000			11000110		
0111000101			01000100			1111010101			11000111		
0111010001			01000101			0011110101			11001000		
0111010000			01000110			0000010000			11001001		
0111010101			01000111			0000110101			11001010		
0001110101			01001000			0011010001			11001011		
0000000100			010-1001			0011000101			11001100		
0100110100			01001010			0000010001			11001101		
0011010100			01001011			0011010000			11001110		
0001000101			01001100			1111010111			11001111		
0000000101			01001101			1100110111			11010000		
0001010000			01001110			1111000011			11010001		
0111010111			01001111			1111011100			11010011		
0100110111			01010000			1111000111			11010100		
0111000011			01010001			1111010011			11010101		
0111011100			01010010			1111000100			11010110		
0111000111			01010100			1111011101			11010111		
0111010011			01010101			0011110111			11011000		
0111000100			01010110			0000000000			11-11001		
0111011101			01010111			0000110111			11011010		
0001110111			01011000			0011010011			11011011		
0110011010			01011010			0011000111			11011100		
0011011100			01011011			0000010011			11011101		
0001000111			01011100			0011000100			11011110		
0000000111			01011101			1111011111			11011111		
0001000100			01011110			1100111101			11100000		
0111011111			01011111			1100110001			11100001		
0100111101			01100000			1100110000			11100010		
0100110001			01100001			1111110100			11100011		
0100110000			01100010			1111001101			11100100		
0111110100			01100011			1111110001			11100101		
0111001101			01100010			1111110000			11100110		
0111001100			01100100			1111110011			11100111		
0111001101			01100101			1111110101			11100110		
0111001100			01100110			1111110111			11100111		
0111001101			01100111			1111111001			11100110		
0111001100			01100110			1111111000			11100111		
0111001101			01100100			1111111001			11100110		
0111001100			01100101			1111111011			11100111		
0111001101			01100100			1111111010			11100110		
0111001100			01100101			1111111001			11100111		
0111001101			01100100			1111111000			11100110		
0111001100			01100101			1111111011			11100111		
0111001101			01100100			1111111010			11100110		
0111001100			01100101			1111111001			11100111		
0111001101			01100100			1111111000			11100110		
0111110001			01100101			1111111011			11100111		

t_9	...	t_0	b_7	...	b_0	t_9	...	t_0	b_9	...	b_0
0111110000			01100110			0011111101			11101000		
0111110101			01100111			0000110000			11101001		
0001111101			01101000			0000111101			11101010		
0000001100			011-1001			0011110001			11101011		
0100111100			01101010			0001100101			11101100		
0011110100			01101011			0000110001			11101101		
0001001101			01101100			0001100101			11101110		
0000001101			01101101			0000001101			1111110111		
0001110000			01101110			0001110000			1111001111		
0111110111			01101111			0011110111			1111000111		
0100111100			01110000			0100111100			1111000011		
0100110011			01110001			0111110011			1111000111		
0100110010			01110010			0111001111			1111001011		
0100110011			01110011			0111110011			1111001101		
0100110010			01110010			0111110010			1111001100		
0100110011			01110011			0111110011			1111001011		
0100110010			01110010			0111110010			1111001100		
0100110011			01110011			0111110011			1111001011		
0100110010			01110010			0111110010			1111001100		
0100110011			01110011			0111110011			1111001011		
0100110010			01110010			0111110010			1111001100		
0100110011			01110011			0111110011			1111001011		
0100110010			01110010			0111110010			1111001100		
0100110011			01110011			0111110011			1111001011		
0100110010			01110010			0111110010			1111001100		
0100110011			01110011			0111110011			1111001011		
0100110010			01110010			0111110010			1111001100		
0100110011			01110011			0111110011			1111001011		
0100110010			01110010			0111110010			1111001100		
0100110011			01110011		</						

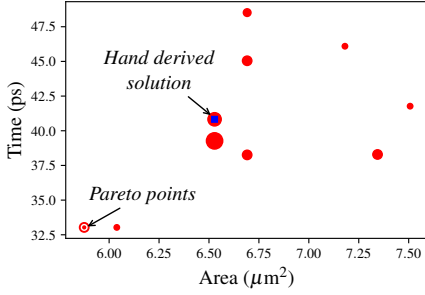


Fig. 1: Design space for the 5 bits to 3 trits decoder

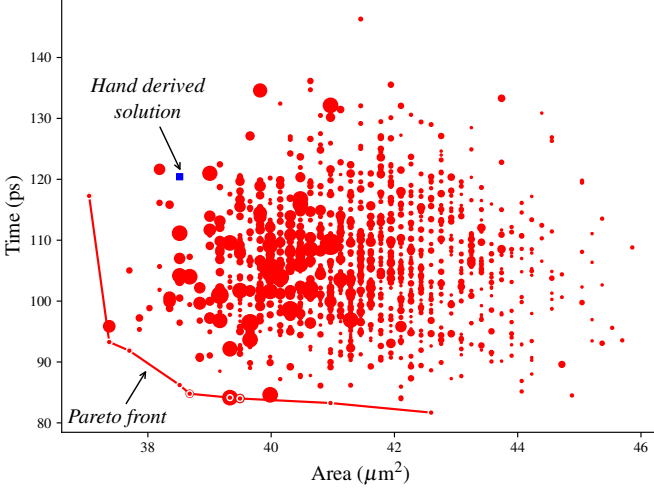


Fig. 2: Design space for the 8 bits to 5 trits decoder

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 $18.44 \mu\text{m}^2$ (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 $102.7 \mu\text{m}^2$ (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.

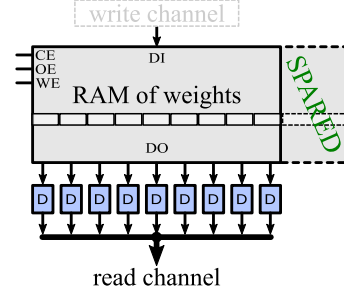


Fig. 3: Impact of decoding (D) on an ternary ANN weight memory. Weights are written once at configuration time, and read simultaneously during inference.

In [20]², the authors report an SRAM bit cell area of $A_B = 0.12 \mu\text{m}^2$ 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:

$$\frac{B}{6} \times W \times A_B - A_D \times \frac{B}{6} > R \times W \times B \times A_B \quad (1)$$

Which simplifies to:

$$W > \frac{A_D}{A_B} \times \frac{1}{1 - R \times 6} \quad (2)$$

A similar argument gives in the 5-trit case:

$$W > \frac{A_D}{A_B} \times \frac{1}{2 - R \times 10} \quad (3)$$

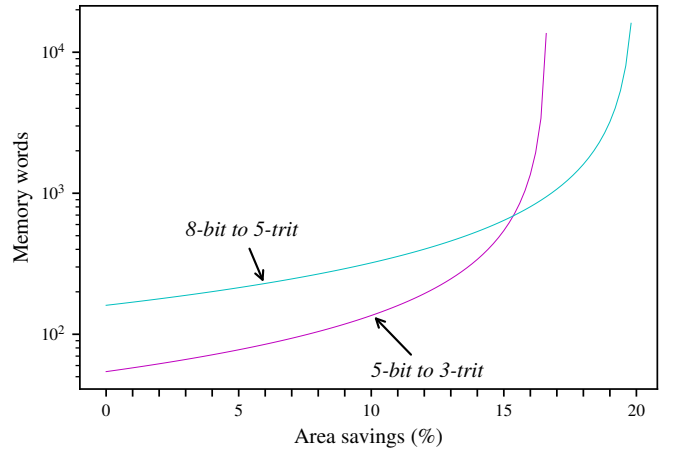


Fig. 4: Minimum memory size for a given area savings

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 cuts depends 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 FDSOI 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 $8\times$ 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

- [1] N. P. Brusentsov and J. R. Alvarez, "Ternary computers: The setun and the setun 70," in *Perspectives on Soviet and Russian Computing*. Springer, 2011, pp. 74–80.
- [2] Z. Sahinoglu and S. Gezici, "Ranging in the IEEE 802.15.4a standard," in *IEEE Annual Wireless and Microwave Technology Conference, 2006*. IEEE, 2006, pp. 1–5.
- [3] M. Abdelaziz and T. A. Gulliver, "Ternary convolutional codes for ternary phase shift keying," *IEEE Communications Letters*, vol. 20, no. 9, pp. 1709–1712, 2016.
- [4] J. Nystad, A. Lassen, A. Pomianowski, S. Ellis, and T. Olson, "Adaptive scalable texture compression," in *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*. Eurographics Association, 2012, pp. 105–114.
- [5] A. V. Burlakov, M. V. Chekhova, O. A. Karabutova, D. N. Klyshko, and S. P. Kulik, "Polarization state of a biphoton: Quantum ternary logic," *Physical Review A*, vol. 60, no. 6, p. R4209, 1999.
- [6] A. K. Ghosh and A. Basuray, "Binary to modified trinary number system conversion and vice-versa for optical super computing," *Natural Computing*, vol. 9, no. 4, pp. 917–934, 2010.
- [7] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.
- [8] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on fpga," in *27th International Conference on Field Programmable Logic and Applications*. IEEE, 2017, pp. 1–7.
- [9] A. Prost-Boucle, A. Bourge, and F. Pétrot, "High-efficiency convolutional ternary neural networks with custom adder trees and weight compression," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 15:1–15:24, Dec. 2018.
- [10] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 1990.
- [11] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Multi-level logic minimization using implicit don't cares," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 6, pp. 723–740, 1988.
- [12] S. Devadas and A. R. Newton, "Exact algorithms for output encoding, state assignment, and four-level boolean minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 1, pp. 13–27, Jan. 1991.
- [13] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4, no. 3, pp. 269–285, 1985.
- [14] G. De Micheli, *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.
- [15] S. Mitra, L. J. Avra, and E. J. McCluskey, "An output encoding problem and a solution technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 761–768, Jun. 1999.
- [16] L. Burgun, N. Dictus, A. Greiner, E. P. Lopes, and C. Sarwary, "Multilevel logic optimization of very high complexity circuits," in *Proceedings of the European Design Automation conference*. IEEE Computer Society Press, 1994, pp. 14–19.
- [17] P. C. McGeer, J. V. Sanghavi, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Espresso-signature: A new exact minimizer for logic functions," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 1, no. 4, pp. 432–440, 1993.
- [18] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.
- [19] E. J. McCluskey Jr, "Minimization of boolean functions," *Bell system technical Journal*, vol. 35, no. 6, pp. 1417–1444, 1956.
- [20] G. Desoli, N. Chawla, T. Boesch, S. P. Singh, E. Guidetti, F. D. Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh, and N. Aggarwal, "A 2.9tops/w deep convolutional neural network soc in fd-soi 28nm for intelligent embedded systems," in *2017 IEEE International Solid-State Circuits Conference*, Feb 2017, pp. 238–239.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [22] A. Pedram, S. Richardson, M. Horowitz, S. Galal, and S. Kvatinsky, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, no. 2, pp. 39–50, 2017.