Approximated Canonical Signed Digit for Error Resilient Intelligent Computation

Cardarilli, Gian Carlo; Di Nunzio, Luca; Fazzolari, Rocco; Nannarelli, Alberto; Re, M.

Published in:
Proceedings of 53rd Asilomar Conference on Signals, Systems, and Computers

Link to article, DOI:
10.1109/IEEECONF44664.2019.9049051

Publication date:
2020

Document Version
Peer reviewed version

Approximated Canonical Signed Digit for Error Resilient Intelligent Computation

G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, A. Nannarelli,[(1)] *and M. Re*

Department of Electronics, University of Rome Tor Vergata, Rome, Italy
[(1)]DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

***Abstract*—Lowering the energy consumption in applications operating on large datasets is one of the main challenges in modern computing. In this context, it is especially important to lower the energy required to transfer data from/to the memory. Usually, this is obtained by applying smart encoding techniques to the data. In this work, we show how to reduce the switching activity in buses and floating-point units by an approximated canonical signed-digit encoder. The precision of the encoding is programmable and can be chosen depending on the application's required accuracy.**

I. Introduction

Lowering the energy consumption in architectures implementing Intelligent Computation algorithms is a primary target for new applications such as, for example, Internet-of-Things (IoT). This goal is achieved by using architectural and microelectronic technology level techniques. At architectural level, the most important actions are aimed at minimizing power dissipation in parts characterized by high capacitive loads due to interconnect (buses), or due to complex architectures, such as hardware multipliers.

Data on power dissipation break-down in multicore chips show that the cores consume about 50% of the total power, and that the several levels of memories account for the remaining 50% [1].

In [2], the authors show that for a 45nm CMOS process the memory access measured in [pJ] is three order of magnitude more energy expensive than the arithmetic implemented in the core. Consequently, energy reduction in memory access is very important for lowering power consumption in applications with large datasets, e.g., artificial neural networks (ANNs).

In [3] and [4], the authors show different techniques for the implementation of low power interconnects by using Canonical Signed Digit Number System and the Binary Coded Canonical Digit Signed Digit. In [5], the authors use Canonical Signed Digit (CSD) coding to reduce the time of execution of the multiplications that are massively used in the ANN inference phase, while in [6], an ANN architecture is implemented on FPGA using CSD-based multipliers obtaining a reduction in hardware resources.

Moreover, a number of engineering fields deal with error resilient applications (image and audio processing) and algorithms (ANN, fuzzy logic, genetic algorithms). Consequently, techniques such as approximate arithmetic and approximate data representation can be fruitfully applied in these areas to achieve energy savings without sacrificing algorithm performances.

As examples, an energy efficient neuromorphic architecture based on approximated multipliers, and a systematic design space exploration approach based on approximated ANNs are presented in [7] and [8].

To address energy efficiency in computation for ANNs, we introduced in [9] a floating-point variable precision format, the Tunable Floating-Point (TFP) format, to handle several precisions and dynamic ranges. Moreover, TFP units for addition and multiplication were introduced in [10] and [11].

In this work, we propose a method to reduce the switching activity in the processing units and in the memory buses by an approximate CSD encoding.

The idea is to suppress sequences of '1's by CSD recoding and approximate the encoded number by omitting the bits of negative weight. Fig. 1 shows an example of the proposed Approximate CSD (ACSD) recoding. Sequences of three or two '1' are not recoded.

The main contribution of this work is the design of the ACSD coder, including a parallel CSD recoding algorithm. The coder precision can be programmed to adjust the accuracy of the approximated recoding for different applications during the algorithm execution. The ACSD coder is implemented in standard cells technology, and we provide data on performance, error rates, and power savings for a few test cases.

Simulations of traffic on the bus for several sample sequences of *binary32* (single-precision) vectors with ACSD encoded significand show a best-case reduction in switching activity of 6%, and a worst-case reduction of 2%.

Similarly, the execution of matrix multiplication for ACSD encoded operands in a *binary32* floating-point unit results in average power savings of about 3%.

II. Parallel Recoding Algorithm

Canonical recoding of binary numbers into the digit set $\{-1, 0, 1\}$ is a sequential recoding that minimizes the number of non-zero bits [12]. For example, A=01 0111 1001 is recoded into F=01 1000 $\overline{1}$001, where $\overline{1} = -1$.

In our algorithm, to parallelize the recoding, the binary number is split into 4-bit digits as illustrated in Fig. 2. To simplify the recoding, we recode only sequences with at least four '1's. Since a sequence of '1's can spread across more digits, we use the transfer bits Q and T to indicate:

input	1 0 1 1	0 0 1 1	1 1 1 1	1 0 1 1	1 1 0 1	1 1 0 1
CSD	1 0 1 1	0 1 0 0	0 0 0 0	0 1 0 0	0 0 0 1	1 1 0 1
recoded	0 0 0 0	0 0 0 0	0 0 0 0	$\overline{1}$ 0 0 0	0 $\overline{1}$ 0 0	0 0 0 0
ACSD	1 0 1 1	0 1 0 0	0 0 0 0	0 1 0 0	0 0 0 1	1 1 0 1

Fig. 1. Example of Approximated Canonical Signed Digit (ACSD) recoding.

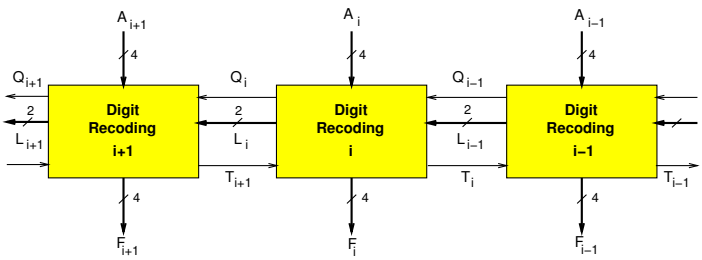

Fig. 2. Architecture of 4-bit digit-based recoder.

- $Q_i = 1$ there is a sequence of four '1's in digit i.
- $T_{i+i} = 1$ there is a sequence of four or more '1's in digit i+1 originated in digit i.

Moreover, if there is a sequence of '1's active in digit i, its length L_i is passed to digit i+1. Based on these definitions, we describe how the recoding is done for the 4-bit digit i (Fig. 3). We do not use indices to refer to local variables in digit i.

The recoding algorithm is divided in the following logical steps.

Step 1. The sequence of '1's can be either toward the right or the left, or toward both ends. Therefore, we list in Table I the sequence lengths L (left) toward digit i+1 and R (right) toward digit i-1.

For example, when A=1101 there is a sequence of two '1's toward the digit at left ($L = 2$) and a sequence of one toward the digit at right ($R = 1$)
The sequence toward the left is passed to digit i+1

$$L_i \leftarrow L = (L1L0)_2 \quad \text{(two bits)}$$

Moreover, the bit Q_i is set to one if A=1111.

Step 2. The results of Table I and the incoming Q_{i-1} and L_{i-1} are used to compute the length of the sequence in digit i, as follows. We compute the length p of the sequence of '1's

$$p = p_{local} + p_R$$

where

- p_{local} is the length in the digit, computed as:

$$\begin{array}{lcll} p_{local} & = & 4 & \text{if } Q_i = 1 \\ p_{local} & = & R & \text{otherwise} \end{array}$$

In the latter case ($Q_i = 0$), there is some '0' in the digit and there is a sequence ≥ 4 only if started in digit i-1.

A	L1	L0	R1	R0	Q_i
0000	0	0	0	0	0
0001	0	0	0	1	0
0010	0	0	0	0	0
0011	0	0	1	0	0
0100	0	0	0	0	0
0101	0	0	0	1	0
0110	0	0	0	0	0
0111	0	0	1	1	0
1000	0	1	0	0	0
1001	0	1	0	1	0
1010	0	1	0	0	0
1011	0	1	1	0	0
1100	1	0	0	0	0
1101	1	0	0	1	0
1110	1	1	0	0	0
1111	0	0	0	0	1

TABLE I
SEQUENCE LENGTHS: $L = (L1L0)_2$ HOLDS THE LENGTH TOWARD THE DIGIT AT LEFT, $R = (R1R0)_2$ TOWARD THE DIGIT AT RIGHT.

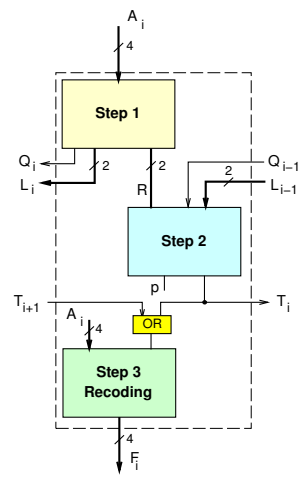

Fig. 3. Detail of the basic block and transfer bits.

- p_R holds the sequence length originating from the digit at right to be added to p_{local}. This sequence is interrupted when there are '0's in the bits of A toward the left:

$$\begin{array}{lcll} p_R & = & 3 & \text{if } L_{i-1} = 3 \text{ AND } R \geq 1 \\ p_R & = & 2 & \text{if } L_{i-1} = 2 \text{ AND } R \geq 2 \\ p_R & = & 1 & \text{if } L_{i-1} = 1 \text{ AND } R = 3 \\ p_R & = & 0 & \text{otherwise} \end{array}$$

For example, A=1001 and $L_{i-1} = 3$, we have $p_{local} = 1$, $p_L = 3$, and $p = 4$. There is a sequence of '1's ending in the digit. In this case, the bits in the digit are recoded as F=1010.

When in digit i there is sequence originating from the right, we have to pass the recoding info to digit i-1. This is done through the bit T (going left-to-right in Fig. 2)

$$T_i = Q_i \text{ OR } (p \geq 4)$$

Step 3. Bit T_i and the incoming (from left) T_{i+1} determine the recoding in the digit. The actual recoding, i.e., transforming the sequence of '1', is done according to L and R values determined in Table I.

Extending the previous example, A=1001 1110, we have: $L_{i-1} = 3$, $p = 4$, $T_i = 1$ resulting in F=1010 00$\overline{1}$0. Clearly, we need a special code to represent $\overline{1}$.

The transfer bits Q_i and T_i, and the L_{i-1} value do not propagate further than one digit, therefore, the recoding is done in parallel. The detail is illustrated in Fig. 3.

The recoder of Fig. 2 requires additional output bits to represent the bits with negative weights $\overline{1}$.

The approximated CSD recoding is implemented by omitting the $\overline{1}$ bits. The approximation introduces an error 2^j where j is the position of $\overline{1}$ in the word (we assume unsigned integers, in this case).

A. Non-Recoding Zone

Since the error can be quite large if the omitted $\overline{1}$ is in the most-significant part of the word, we can adjust the accuracy of the approximated recoding by introducing a *"non-recoding zone (NRZ)"*. Fig. 4 shows an example where the NRZ is extended to the 16 most-significant bits (MSBs). With this protection, $H = 16$, the maximum error is 2^7 (unsigned integers). The actual error in the example of Fig. 4 is 2^6.

The NRZ is specified by a mask M, implemented by a decoder. For example, for a 16-bit integer, if we want to limit the error to 2^8 we have:

$$\begin{array}{lcll} A & = & 0011\ 1110\ 0111\ 1110 & \text{(input)} \\ M & = & 1111\ 1111\ 0000\ 0000 & \text{(mask)} \\ F & = & 0011\ 1110\ 1000\ 00\overline{1}0 & \text{(recoded)} \\ F_A & = & 0011\ 1110\ 1000\ 0000 & \text{(approx. recoded).} \end{array}$$

In each digit of the recoder, we need to introduce an enable signal (EN) to enforce the NRZ. The operations in the modified 4-bit digit encoder, illustrated in Fig. 5, are the following:

- In each digit, the mask is checked against the position of the trailing '1' (to become $\overline{1}$, and, therefore, omitted). For example, if A=1110 we define the position of the trailing '1' as B=0010 and we determine if the position falls in the NRZ for each of the four bits B_j and M_j

  $$K = \text{NOT}(B_3M_3 \text{ OR } B_2M_2 \text{ OR } B_1M_1 \text{ OR } B_0M_0)$$

  Therefore, $K = 0$ indicates that there is potential trailing '1' falling in the NRZ. For example, for B=0010, if M=1111 $\rightarrow K = 0$, but if M=1100 $\rightarrow K = 1$, and the potential trailing '1's can be recoded because the error introduced by omitting $\overline{1}$ falls outside the NRZ.
- If $K = 1$, the partial sequence '1' is recoded only if $T_{i+1} = 1$. In this case the "enable recoding" information is communicated to the adjacent digits if $E_i = T_{i+1}$ AND K.
- To preserve the NRZ across several digits, long sequences of '1', the enable in digit $i+1$ is generated as:

  $$\text{EN}_{i+1} = E_i \text{ AND } E_{i-1} \text{ AND } \ldots \text{ AND } E_0$$

input	1 0 1 1	0 0 1 1	1 1 1 1	1 0 1 1	1 1 0 1	1 1 0 1
CSD	1 0 1 1	0 1 0 0	0 0 0 0	0 1 0 0	0 0 0 1	1 1 0 1
recoded	0 0 0 0	0 0 0 0	0 0 0 0	$\overline{1}$ 0 0 0	0 $\overline{1}$ 0 0	0 0 0 0
	N R Z					
ACSD	1 0 1 1	0 0 1 1	1 1 1 1	1 1 0 0	0 0 0 1	1 1 0 1
				ϵ_{max} ↑		

Fig. 4. Example of ACSD recoding with "non-recoding zone" (NRZ).

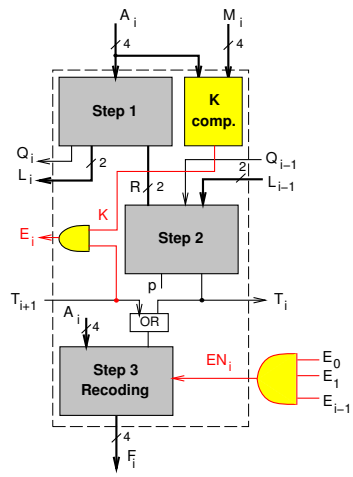

Fig. 5. Modified ACSD recoding block to handle NRZ.

B. Critical Path

Fig. 6 shows the architecture for a (n+1)-bit ACSD recoder with NRZ. The highlighted blocks contributes to the critical path. In two adjacent digits i and $i+1$, there is competition between the sum of delays in **Step 1** and **Step 2** in block $i+1$ (producing T_{i+1}, and the delay of **K comp.** in block i plus the delay of the decoder to generate the mask M depending on the value of the non-recoding zone H, not depicted in the figure. Since the synthesis tends to equalize the slacks in the different paths, we reported both possibilities in Fig. 6.

Once all E_i values are generated in parallel (same delay for all digits), the longest delay in the critical path depends on the fan-in of the AND gate generating the EN_i values. Therefore, EN_n in the most-significant digit is the one on the critical path because the fan-in of the AND gate is n (the largest).

Fig. 6 demonstrates that in the overall recoding process with non-recoding zone there is no carry propagation through the n+1 digits.

III. Experimental Results

We ran some tests to evaluate the errors introduced by the approximated recoder for approximating the 24-bit significand of *binary32* (single-precision) floating-point numbers. This format is suitable for training in deep learning applications.

We ran tests on 1024 24-bit random vectors and extended the *"non-recoding zone"* from $H = 0$ (fully approximated) to $H = 24$ (no recoding).

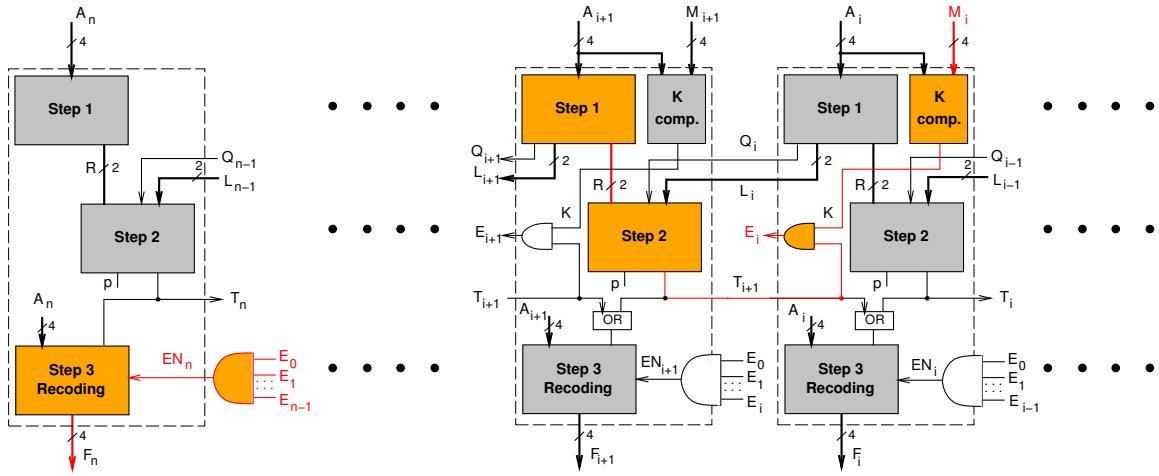

Fig. 6. Critical path for an ACSD (n+1)-digit recoder with NRZ.

numbers (total)	:	1024
n. seq. '1's ≥ 4 (total)	:	697
average seq. length	:	4.82 bits
max. seq. length	:	12 bits

NRZ H	numbers approx.	errors ϵ_{ave}	ϵ_{max}	reduct. 1s count
24	0	0	0	0
20	107	2^{-23}	2^{-18}	6%
16	168	2^{-20}	2^{-14}	9%
12	344	2^{-16}	2^{-11}	14%
8	434	2^{-12}	2^{-6}	18%
4	527	2^{-9}	2^{-3}	21%
0	697	2^{-8}	2^{-2}	23%

TABLE II
APPROXIMATED RECODING: SUMMARY OF RESULTS FOR DIFFERENT NRZ.

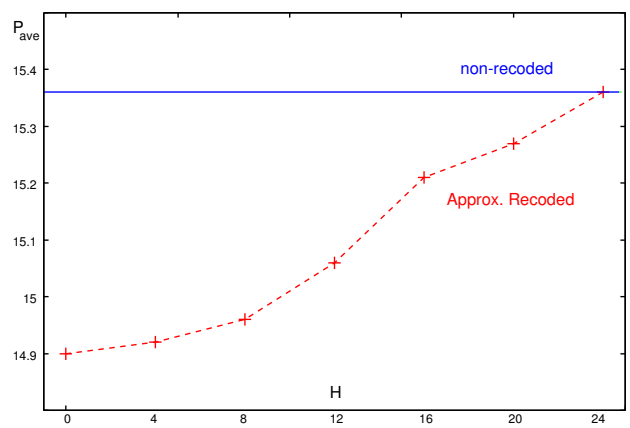

Fig. 7. Average power dissipation in FP-unit as H scales. P_{ave} is in mW at 1GHz clock rate.

The results, reported in Table II, show that the average error on the significand ϵ_{ave} is quite acceptable and the reduction in the number of '1's in the representation is quite significant. Moreover, for each NRZ, the table reports the number of approximated elements (col. 2), and the reductions in the number of '1's for the 1024 elements after recoding (col. 5)

The reduction in the switching activity in a bus depends on how the sequence of data are accessed (transit on bus). We ran simulations for several sample sequences (traffic on the bus) and found a best-case reduction in switching activity of 6%, and a worst-case reduction of 2%. Although the reduction is small, it may lead to sizeable power dissipation savings in the bus if the switched capacitance is large.

A. Hardware Implementation

The 24-bit approximated recoder is implemented in a commercial 45 nm library of standard cell. The unit includes hardware to renormalize the significand in case the recoding causes an overflow.

The latency of the approximated recoding is 520 ps, corresponding to 8 FO4 delay.

As application, we chose matrix multiplication, arguably, the most common kernel in machine learning.

Fig. 7 shows the average power dissipation reduction when the approximated recoding is applied to a FP-unit (FP-multiplier and FP-adder) executing the matrix product. By increasing the approximation (reducing H) we can achieve power savings of about 3%.

IV. CONCLUSIONS

In this work, we designed a parallel approximated CSD recoder to reduce the switching activity in buses and FP-units.

The recoding is done in parallel on 4-bit digits and the overall operation is carry-free. The critical path traverses the blocks belonging to three digits, but contributing to the delay of one digit only, plus a few gates (Fig. 6).

Although the energy savings are not large, the trade-off error/savings makes the ACSD recoder suitable for applications in deep learning and other areas where huge datasets need to be transfered from/to memory.

REFERENCES

[1] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb. 2014, pp. 10–14.

[2] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," in *Advances in Neural Information Processing Systems (NIPS)*, Dec. 2015, pp. 904 – 907.

[3] S. Saini, *Low Power Interconnect Design*. Springer, 2015.

[4] R. Hashemian, "A new method for conversion of a 2s complement to canonic signed digit system and its representation," in *Proc. of 30th Asilomar Conference on Signals, Systems and Computers*, 1996, pp. 904 – 907.

[5] A. Parvin, M. Ahmadi, and R. Muscedere, "Application of neural networks with CSD coefficients for human face recognition," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2013, pp. 1628 – 1631.

[6] S. L. Pinjare and E. H. Kumar, "Implementation of Artificial Neural Network Architecture for Image Compression Using CSD Multiplier," in *Proc. of International Conference on Emerging Research in Computing, Information, Communication and Applications (ERCICA)*, 2013, pp. 581–587.

[7] W. Qian, L. Youjie, S. Botang, D. Siddhartha, and L. Peng, "Energy Efficient Parallel Neuromorphic Architectures with Approximate Arithmetic on FPGA," *Neurocomputing*, vol. 221, 10 2016.

[8] M. Nazemi and M. Pedram, "Deploying customized data representation and approximate computing in machine learning applications," in *Proc. of the International Symposium on Low Power Electronics and Design*, Jul. 2018, pp. 1–6.

[9] A. Nannarelli, "Tunable Floating-Point for Energy Efficient Accelerators," in *25th IEEE Symposium on Computer Arithmetic*, Jun. 2018, pp. 29–36.

[10] ——, "Tunable Floating-Point Adder," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1553–1560, Oct. 2019.

[11] ——, "Fused Multiply-Add for Variable Precision Floating-Point," in *32nd IEEE International System-on-Chip Conference (SOCC)*, Sep. 2019, pp. 342–347.

[12] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.