

Multi-input Logic-in-Memory for Ultra-Low Power non-von Neumann Computing

Tommaso Zanotti ^{1,*}, Paolo Pavan ¹ and Francesco Maria Puglisi ¹

¹ Dipartimento di Ingegneria "Enzo Ferrari", Università di Modena e Reggio Emilia, Via P. Vivarelli 10/1, 6 41125 Modena, Italy; francescomaria.puglisi@unimore.it (F.M.P.); paolo.pavan@unimore.it (P.P.)

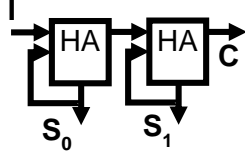
* Correspondence: tommaso.zanotti@unimore.it;

# Step	Operation	Equivalent Operation	# Step	Operation	Equivalent Operation
1-4	FALSE (C, M ₁ , M ₂ , M ₃)	C = M ₁ = M ₂ = M ₃ = 0	1-4	sFALSE (C, M ₁ , M ₂ , M ₃)	C = M ₁ = M ₂ = M ₃ = 0
5	2-SIMPLY (I, M ₁)	M ₁ = \bar{I}	5	2-SIMPLY (I, M ₁)	M ₁ = \bar{I}
6	2-SIMPLY (S ₀ , M ₂)	M ₂ = \bar{S}_0	6	2-SIMPLY (S ₀ , M ₂)	M ₂ = \bar{S}_0
7	2-SIMPLY (S ₀ , M ₁)	M ₁ = $\bar{S}_0 \bar{I}$	7	3-SIMPLY (M ₁ , M ₂ , C)	C = S ₀ I
8	2-SIMPLY (M ₂ , I)	I = $\bar{S}_0 \bar{I}$	8	2-SIMPLY (M ₂ , M ₃)	M ₃ = S ₀
9	2-SIMPLY (M ₁ , M ₃)	M ₃ = S ₀ I	9	FALSE (S ₀)	S ₀ = 0
10	2-SIMPLY (I, M ₃)	M ₃ = S ₀ I + $\bar{S}_0 \bar{I}$	10	2-SIMPLY (M ₂ , I, S ₀)	S ₀ = S ₀ \bar{I}
11	FALSE (S ₀)	S ₀ = 0	11	2-SIMPLY (M ₃ , M ₁ , S ₀)	S ₀ = $\bar{I} \bar{S}_0 + S_0 \bar{I}$
12	2-SIMPLY (M ₃ , S ₀)	S ₀ = S ₀ \bar{I} + $\bar{S}_0 \bar{I}$			
13	2-SIMPLY (M ₁ , C)	C = S ₀ I			

(a)

(b)

Figure S1. 1-bit HA sequence of computing steps using (a) 2-SIMPLY only and, (b) up to 3-SIMPLY. The use of 3-SIMPLY reduces the number of computing steps from 13 to 11.

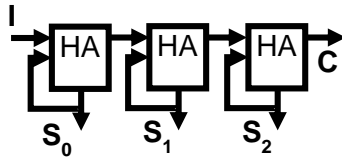


# Step	Operation	Equivalent Operation
1-6	sFALSE (C, M ₁ , M ₂ , M ₃ , M ₄ , M ₅)	$C = M_1 = M_2 = M_3 = M_4 = M_5 = 0$
7	2-SIMPLY (I, M ₁)	$M_1 = \bar{I}$
8	2-SIMPLY (S ₀ , M ₂)	$M_2 = \bar{S}_0$
9	2-SIMPLY (S ₁ , M ₃)	$M_3 = \bar{S}_1$
10	4-SIMPLY (M ₁ , M ₂ , M ₃ , C)	$C = S_0 S_1 I$
11	2-SIMPLY (M ₂ , M ₄)	$M_4 = S_0$
12	2-SIMPLY (M ₃ , M ₅)	$M_5 = S_1$
13-14	sFALSE (S ₀ , S ₁)	$S_0 = S_1 = 0$
15	3-SIMPLY (I, M ₂ , S ₀)	$S_0 = \bar{I} S_0$
16	3-SIMPLY (M ₁ , M ₄ , S ₀)	$S_0 = \bar{I} \bar{S}_0 + S_0 \bar{I}$
17	3-SIMPLY (I, M ₃ , S ₁)	$S_1 = S_1 \bar{I}$
18	3-SIMPLY (M ₄ , M ₃ , S ₁)	$S_1 = S_1 \bar{I} + S_1 \bar{S}_0$
19	4-SIMPLY (M ₁ , M ₂ , M ₄ , S ₁)	$S_1 = S_1 \bar{I} + S_1 \bar{S}_0 + I S_0 \bar{S}_1$

(a)

(b)

Figure S2. 4-SIMPLY-based implementation of two concatenated 1-bit HAs that are used to implement the popcounting operation for BNNs. **(a)** Sketch of the two serially connected half-adders (HAs). **(b)** Sequence of IMPLY and FALSE operations implementing the result of the operation. The use of 4-SIMPLY reduces the number of computing steps from 26 (when using only the 2-SIMPLY Figure S1a) to 19.

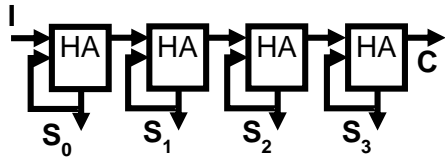


# Step	Operation	Equivalent Operation
1-8	sFALSE (C, M ₁ , M ₂ , M ₃ , M ₄ , M ₅ , M ₆ , M ₇)	$C = M_1 = M_2 = M_3 = M_4 = M_5 = M_6 = M_7 = 0$
9	2-SIMPLY (I, M ₁)	$M_1 = \bar{I}$
10	2-SIMPLY (S ₀ , M ₂)	$M_2 = \bar{S}_0$
11	2-SIMPLY (S ₁ , M ₃)	$M_3 = \bar{S}_1$
12	2-SIMPLY (S ₂ , M ₆)	$M_6 = \bar{S}_2$
13	5-SIMPLY (M ₁ , M ₂ , M ₃ , M ₆ , C)	$C = S_0 S_1 S_2 I$
14	2-SIMPLY (M ₂ , M ₄)	$M_4 = S_0$
15	2-SIMPLY (M ₃ , M ₅)	$M_5 = S_1$
16	2-SIMPLY (M ₆ , M ₇)	$M_7 = S_2$
17-19	sFALSE (S ₀ , S ₁ , S ₂)	$S_0 = S_1 = S_2 = 0$
20	3-SIMPLY (I, M ₂ , S ₀)	$S_0 = S_0 I$
21	3-SIMPLY (M ₁ , M ₄ , S ₀)	$S_0 = \bar{I} S_0 + S_0 \bar{I}$
22	3-SIMPLY (I, M ₃ , S ₁)	$S_1 = S_1 \bar{I}$
23	3-SIMPLY (M ₄ , M ₅ , S ₁)	$S_1 = S_1 \bar{I} + S_1 \bar{S}_0$
24	4-SIMPLY (M ₁ , M ₂ , M ₄ , S ₁)	$S_1 = S_1 \bar{I} + S_1 \bar{S}_0 + I S_0 \bar{S}_1$
25	3-SIMPLY (M ₅ , M ₂ , S ₂)	$S_2 = S_0 \bar{S}_1$
26	3-SIMPLY (I, M ₆ , S ₂)	$S_2 = S_0 \bar{S}_1 + \bar{I} S_2$
27	3-SIMPLY (M ₆ , M ₄ , S ₂)	$S_2 = S_0 \bar{S}_1 + \bar{I} S_2 + S_2 \bar{S}_0$
28	5-SIMPLY (M ₁ , M ₂ , M ₃ , M ₇ , S ₂)	$S_2 = S_0 \bar{S}_1 + \bar{I} S_2 + S_2 \bar{S}_0 + I S_0 S_1 \bar{S}_2$

(a)

(b)

Figure S3. 5-SIMPLY-based implementation of three concatenated 1-bit HAs that are used to implement the popcounting operation for BNNs. (a) Sketch of the three serially connected half-adders (HAs). (b) Sequence of IMPLY and FALSE operations implementing the result of the operation. The use of 5-SIMPLY reduces the number of computing steps from 39 (when using only the 2-SIMPLY Figure S1a) to 28.

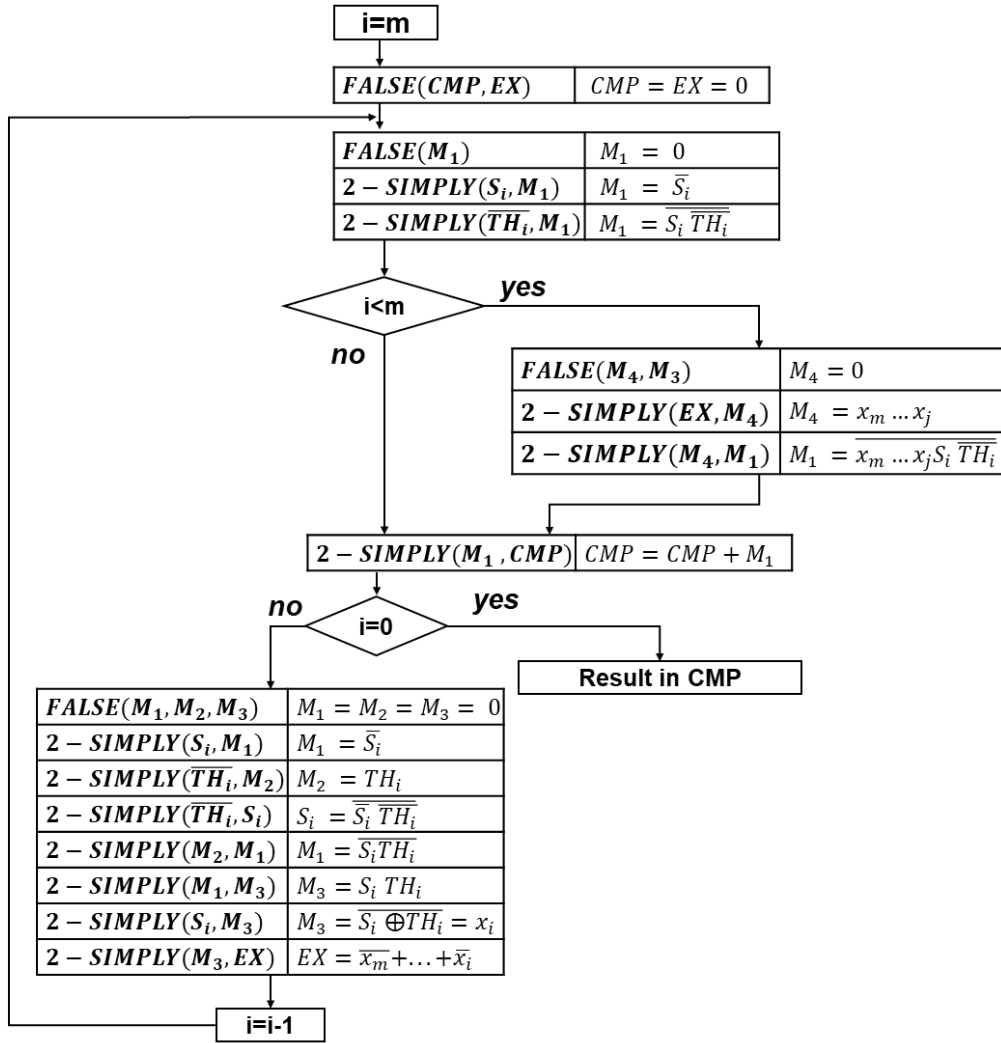


(a)

# Step	Operation	Equivalent Operation
1-10	sFALSE (C, M ₁ , M ₂ , M ₃ , M ₄ , M ₅ , M ₆ , M ₇ , M ₈ , M ₉)	$C = M_1 = M_2 = M_3 = M_4 = M_5 = M_6 = M_7 = M_8 = M_9 = 0$
11	2-SIMPLY (I, M ₁)	$M_1 = \bar{I}$
12	2-SIMPLY (S ₀ , M ₂)	$M_2 = \bar{S}_0$
13	2-SIMPLY (S ₁ , M ₃)	$M_3 = \bar{S}_1$
14	2-SIMPLY (S ₂ , M ₆)	$M_6 = \bar{S}_2$
15	2-SIMPLY (S ₃ , M ₈)	$M_8 = \bar{S}_3$
16	6-SIMPLY (M ₈ , M ₁ , M ₂ , M ₃ , M ₆ , C)	$C = S_0 S_1 S_2 S_3 I$
17	2-SIMPLY (M ₂ , M ₄)	$M_4 = S_0$
18	2-SIMPLY (M ₃ , M ₅)	$M_5 = S_1$
19	2-SIMPLY (M ₆ , M ₇)	$M_7 = S_2$
20	2-SIMPLY (M ₈ , M ₉)	$M_9 = S_3$
21-24	sFALSE (S ₀ , S ₁ , S ₂ , S ₃)	$S_0 = S_1 = S_2 = S_3 = 0$
25	3-SIMPLY (I, M ₂ , S ₀)	$S_0 = S_0 I$
26	3-SIMPLY (M ₁ , M ₄ , S ₀)	$S_0 = \bar{I} S_0 + S_0 \bar{I}$
27	3-SIMPLY (I, M ₃ , S ₁)	$S_1 = S_1 \bar{I}$
28	3-SIMPLY (M ₄ , M ₃ , S ₁)	$S_1 = S_1 \bar{I} + S_1 \bar{S}_0$
29	4-SIMPLY (M ₁ , M ₂ , M ₄ , S ₁)	$S_1 = S_1 \bar{I} + S_1 \bar{S}_0 + I S_0 \bar{S}_1$
30	3-SIMPLY (M ₅ , M ₂ , S ₂)	$S_2 = S_0 \bar{S}_1$
31	3-SIMPLY (I, M ₆ , S ₂)	$S_2 = S_0 \bar{S}_1 + \bar{I} S_2$
32	3-SIMPLY (M ₆ , M ₄ , S ₂)	$S_2 = S_0 \bar{S}_1 + \bar{I} S_2 + S_2 \bar{S}_0$
33	5-SIMPLY (M ₁ , M ₂ , M ₃ , M ₇ , S ₂)	$S_2 = S_0 \bar{S}_1 + \bar{I} S_2 + S_2 \bar{S}_0 + I S_0 S_1 \bar{S}_2$
34	3-SIMPLY (I, M ₈ , S ₃)	$S_3 = S_3 \bar{I}$
35	4-SIMPLY (M ₁ , M ₇ , M ₈ , S ₃)	$S_3 = S_3 \bar{I} + I S_3 \bar{S}_2$
36	4-SIMPLY (M ₁ , M ₄ , M ₈ , S ₃)	$S_3 = S_3 \bar{I} + I S_3 \bar{S}_2 + I S_3 \bar{S}_0$
37	4-SIMPLY (M ₁ , M ₅ , M ₈ , S ₃)	$S_3 = S_3 \bar{I} + I S_3 \bar{S}_2 + I S_3 \bar{S}_0 + S_3 \bar{I} + I S_3 \bar{S}_2 + I S_3 \bar{S}_1$

(b)

Figure S4. 6-SIMPLY-based implementation of four concatenated 1-bit HAs that are used to implement the popcounting operation for BNNs. (a) Sketch of the four serially connected half-adders (HAs). (b) Sequence of IMPLY and FALSE operations implementing the result of the operation. The use of 6-SIMPLY reduces the number of computing steps from 52 (when using only the 2-SIMPLY Figure S1a) to 37.

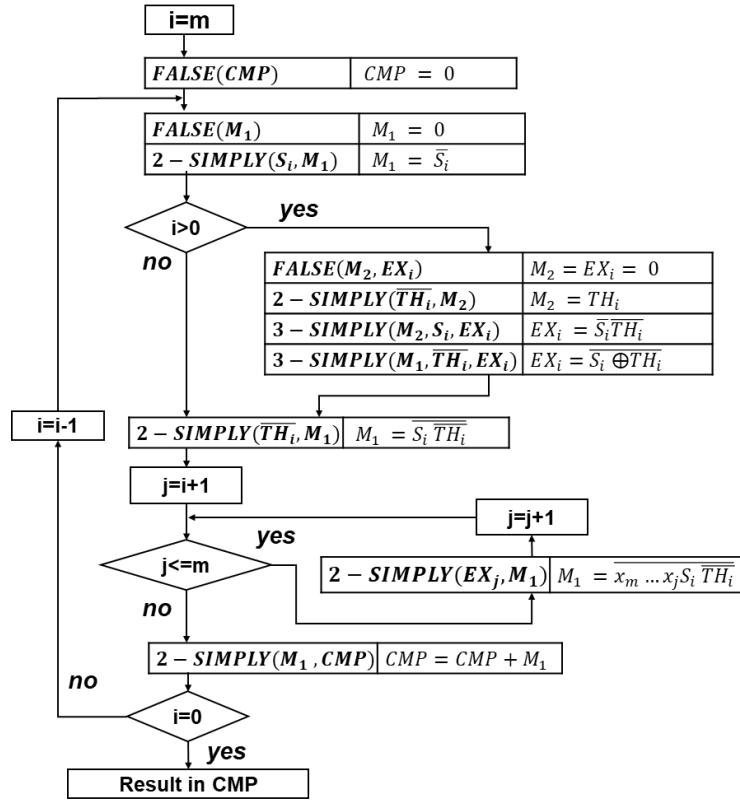


$$n = m + 1$$

if $n = 1 \rightarrow \#Steps = 5$

else $\rightarrow \#Steps = 18n - 12$

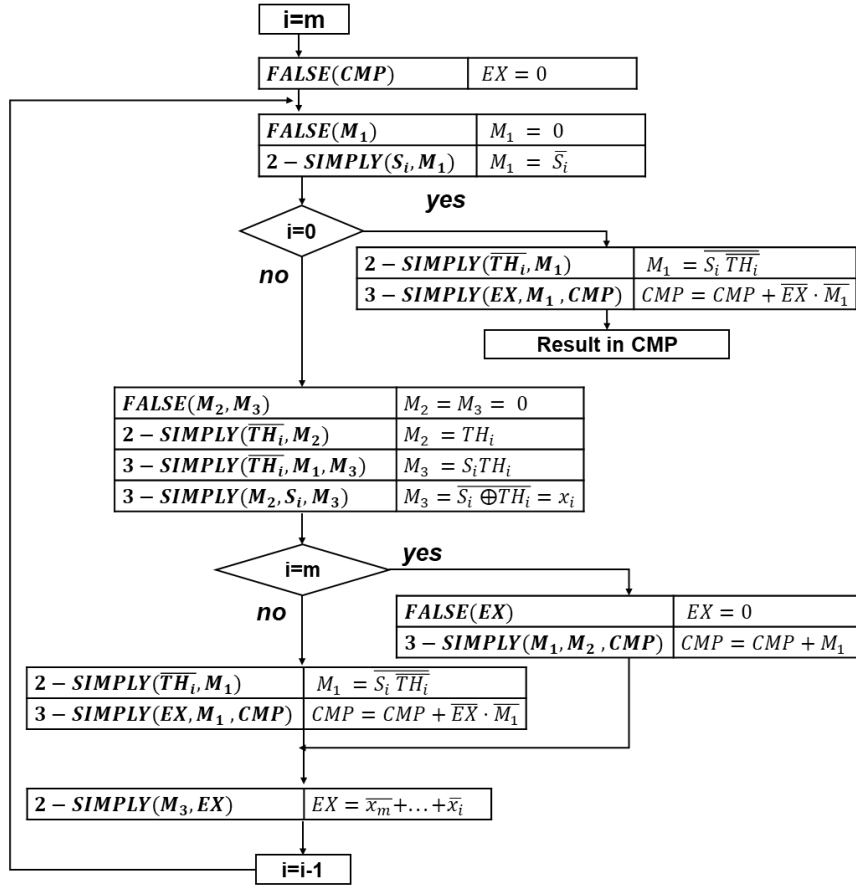
Figure S6. Flowchart building the sequence of computing steps optimized for large number of input bits, that is used to implement the activation function of a BNN neuron, i.e., the comparison with a threshold (TH), when using the 2-SIMPLY operation (i.e., 2-SIMPLY Opt. Wide Words in Figure 11). The formula for the comparison operation is reported together with the scaling of the number of computing steps (i.e., #steps) as a function of the number of compared input bits (n).



$$n = m + 1$$

$$\#Steps = 0.5 n^2 + 8.5n - 4$$

Figure S7. Flowchart building the sequence of computing steps that is used to implement the activation function of a BNN neuron, i.e., the comparison with a threshold (TH), when using the 3-SIMPLY operation and optimizing only the steps required for computing the intermediate XNOR operations (i.e., 3-SIMPLY Only XNOR in Figure 11). The formula for the comparison operation is reported together with the scaling of the number of computing steps (i.e., #steps) as a function of the number of compared input bits (n).



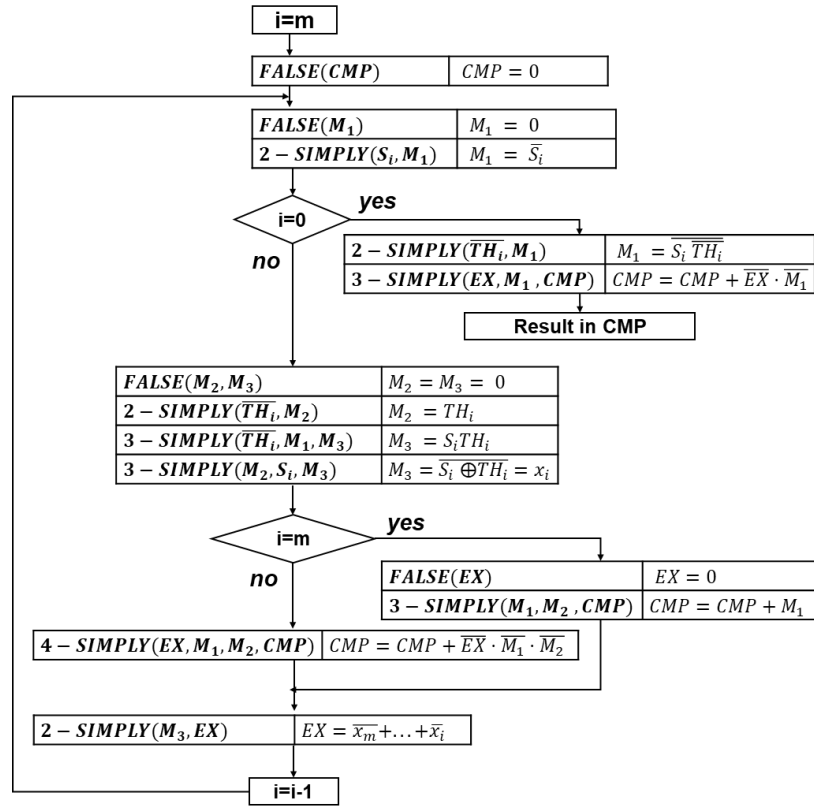
$$n = m + 1$$

$$\text{if } n = 1 \rightarrow \#Steps = 5$$

$$\text{if } n = 2 \rightarrow \#Steps = 15$$

$$\text{else} \rightarrow \#Steps = 10n - 5$$

Figure S8. Flowchart building the optimized sequence of computing steps that is used to implement the activation function of a BNN neuron, i.e., the comparison with a threshold (TH), when using the 3-SIMPLY operation (i.e., 3-SIMPLY Opt. in Figure 11). The formula for the comparison operation is reported together with the scaling of the number of computing steps (i.e., #steps) as a function of the number of compared input bits (n).



$$n = m + 1$$

$$\text{if } n = 1 \rightarrow \#Steps = 5$$

$$\text{if } n = 2 \rightarrow \#Steps = 15$$

$$\text{else} \rightarrow \#Steps = 9n - 3$$

Figure S9. Flowchart building the optimized sequence of computing steps that is used to implement the activation function of a BNN neuron, i.e., the comparison with a threshold (TH), when using the 4-SIMPLY operation (i.e., 4-SIMPLY Opt. in Figure 11). The formula for the comparison operation is reported together with the scaling of the number of computing steps (i.e., #steps) as a function of the number of compared input bits (n).