In the neural network context, used in a variety of applications, binarised networks, which describe both weights and activations as single-bit binary values, provide computationally attractive solutions. A lightweight binarised neural network system can be constructed using only logic gates and counters together with a two-valued activation function unit. However, binarised neural networks represent the weights and the neuron outputs with only one bit, making them sensitive to bit-flipping errors. Binarised weights and neurons are manipulated by the utilisation of bitstream processing with regard to stochastic computing to cope with this error sensitivity. Stochastic computing is shown to provide robustness for bit errors on data while being built on a hardware structure, whose implementation is simplified by a novel subtraction-free implementation of the neuron activation.

### Table 1. Multiplication in decimal and binary values

| Multiplication in Decimal | Multiplication in Binary (XNOR) | Decimal versus Binary Equivalency |
|--------------------------|---------------------------------|----------------------------------|
| +1 x +1 = +1             | 1⊙1 = 1                         | (+1)\(_{10} = (1)\(_2       |
| −1 x −1 = +1             | 0⊙0 = 1                         | (−1)\(_{10} = (0)\(_2       |
| +1 x −1 = −1             | 1⊙0 = 0                         |                                |
| −1 x +1 = −1             | 0⊙1 = 0                         |                                |

### Background: In bitstream processing, the numbers are represented by a binary encoding scheme, before being processed as a bit flow by simple logic hardware, to implement numerical operations. In this work, arguments in italic fonts are used to denote scalar numbers, while bold italic fonts denote their corresponding binary stream. Let \( x \) be a scalar value. Its binary stream representation is composed of \( N \) binary elements \( x = [x_1, \ldots, x_N] \) where \( N \in \mathbb{Z}^+ \), \( N \geq 2 \) define the stream size, and any ith element out of \( N \), \( x_n \), is either binary 1 or binary 0. In a bitstream, let \( K \) be the number of 1s denoted by \( K = \sum_{n=1}^{N} x_n \). Unipolar encoding (UPE) and bipolar encoding (BPE) are generally considered to encode any scalar into a bitstream, but the fractional number encoding is essential in neural network applications. A positive-only fractional number \( x \), in the range \([0, 1]\), is encoded via UPE such that \( x = \frac{x}{2^N} \). However, in BPE, signed number \( x \), in \([-1, +1]\), is represented by \((x + 1)/2 \equiv K/N, \) so that decoding \( x \) stream is given represented by \( x = \left(\left(\sum_{n=1}^{N} x_n\right) / N\right) * N \). In the BPE stream, a majority of 1s, that is, \( \text{high value} \), is an indication of the positive sign, whereas a majority of 0s is an indication of the negative sign. Thus, BPE is superior to UPE in terms of sign estimation robustness. In both the schemes, at encoding, a Bernoulli sequence can be considered to convert the scalar number into a random sequence for perfect randomness of each bit [3]. In this work, data are represented using the BPE format in the proposed framework.

### Proposed framework: In this study, the acronym BNN refers to the conventional deterministic binarised neural network [7]. In contrast, the ‘BSBNN’ acronym denotes the bitstream-processing BNN. We first revisit how BNN works, and then introduce the proposed BSBNN implementation.

In any deterministic neural network, \( S_{n}^{(i)} \) denotes the \( n \)th neuron pre-activation related to the \( n \)th hidden layer (HL). Pre-activation as \( S_{n}^{(i)} = (x_i \times w_{ni}^{(i)}) + \ldots + (x_N \times w_{nN}^{(i)}) \) is computed with the multiplied and accumulated values of preceding neuron (or input) values, \( x_i, \ldots, w_{nN} \), where \( p \) is the number of neurons in the \((n-1)\)th layer. In the hidden layers of the deterministic BNN, decimal weight and neuron output values, restricted to \( 1 \) during training, are mapped into 1-bit binary values such as \((+1)_{10} \equiv (1), \) and \((−1)_{10} \equiv (0)\). This makes it possible to implement bitwise multiplication using a single XNOR gate. Table 1 compares decimal and bitwise multiplications and demonstrates how decimal multiplication is degraded into a simple logic operation via XNOR gates.

However, in deterministic BNN, accumulation is performed by counting the population of 1s. For this operation, a modulus (MOD) digital counter circuit is utilized. Let \( T \) be the total number of states of the counter. Including a zero initial accumulation value, the counter can count up to the final state, \( T - 1 \). The term 'popcount', short for population count, is generally used to denote the basic counter logic hardware [7, 8]. Figure 1 depicts an example of an asynchronous counter based on the D-type flip-flops (FFs) for popcount. Binary inputs to be accumulated are fed sequentially to the first flip-flop clock input, and the base-2 output.
put, \((Q_i|Q_j|Q_k)_1\), gives the total number of 1s in the input binary word. It naturally resets to zero when the total number of 1s presented to the input has reached a value that is the number of ffs to the power of two, that is, \(2^{\text{ord}}(f f)\).

When considering our BSBBN framework, we implement multiplication and accumulation operations using bitstream sourced logic elements. Distinct from the prior art [8], bitstreams are performed not only to represent the image pixel values but also to define all neuron outputs and weights, that are handled throughout hidden layers. Thus, a \(-1\) weight value in deterministic BNN is converted into \(w = [0, 0, \ldots , 0]\) for bitstream processing. Likewise, \(+1\) is converted into \(w = [1, 1, \ldots , 1]\) for bitstream processing. Our experiments show that this way of representing weights and hidden values improves the bit-flip data error robustness.

Next, the implementation of the proposed BSBBN bitstream operation is detailed. In bitstream processing, the multiplier is either an AND gate for UPE bitstreams or an XNOR gate for BPE bitstreams. Since the weights can be \(+1\) or \(-1\) in the binarised network case, BPE streams are adopted, and the multiplication is implemented by the XNOR gate. Figure 2 depicts an example of XNOR-based multiplication for X and W BPE bitstreams.

In BSBBN, the neuron activation can be computed directly by processing the concatenation of all multiplication bitstreams, denoted as \(\{(x_i|w_{i,j}|x_{p,q})\}_{i,j}\), where \(x_i|w_{i,j}|x_{p,q}\) represents the multiplication stream related to the \(i\)th neuron from the preceding layer. The concatenated bitstream is inputted to a counter, which decides if the output has reached a value that is the number of ffs to the power of two, that is, \(2^{\text{ord}}(f f)\).

To conclude, hardware units in BSBBN perform compared to BNN in case of input or model bit-flip errors. implemented efficiently, with reduced hardware logic, we analyse how Network on the simulation: After the discussion on how BSBBN can be implemented efficiently, with reduced hardware logic, we analyse how BSBBN performs compared to BNN in case of input or model bit-flip errors. This section simulates our proposed network, using the MNIST handwritten digit dataset [11], and presents the obtained results. Figure 6 presents the accuracy obtained on the MNIST digit dataset by previous handwritten digit dataset [11], and presents the obtained results. Figure 6 presents the accuracy obtained on the MNIST digit dataset by previous handwritten digit dataset [11], and presents the obtained results. Figure 6 presents the accuracy obtained on the MNIST digit dataset by previous handwritten digit dataset [11], and presents the obtained results. Figure 6 presents the accuracy obtained on the MNIST digit dataset by previous handwritten digit dataset [11], and presents the obtained results. Figure 6 presents the accuracy obtained on the MNIST digit dataset by previous handwritten digit dataset [11], and presents the obtained results. Figure 6 presents the accuracy obtained on the MNIST digit dataset by previous handwritten digit dataset [11], and presents the obtained results. Figure 6 presents the accuracy obtained on the MNIST digit dataset by previous handwritten digit dataset [11], and presents the obtained results.
An exponential learning rate decay is applied in the range of 10^−4 to 10^−6. The network is made of four layers, each of them containing 784-1024-1024-10 neurons, respectively. Squared hinge loss is used as the loss function. For the case in which errors affect pixel values, some binary symbols are manipulated in 8-bit traditional binary representation for deterministic BNN. For BSBNN, the image data is represented in a floating-point version of the weights, which are then binarised to deterministic weights. However, the weight update is performed considering a floating-point version of the weights, which are then binarised to define the forward path. The network is made of four layers, each of them including 784-1024-1024-10 neurons, respectively. Squared hinge loss is chosen, which outperforms cross-entropy loss in terms of validation accuracy. An exponential learning rate decay is applied in the range of \([3 \times 10^{-7}, 3 \times 10^{-7}]\). The batch size is set to 100. Dropout is implemented with \(p = 0.2\) for the input layer, and with \(p = 0.5\) for hidden ones. The best performing model is saved by considering the best validation accuracy, 98.43\%, at the 968th epoch over 1000 epochs. The deterministic BNN test accuracy is 98.27\%.

From a data representation point of view, traditional binary data in BNN is known to be quite vulnerable to bit-flipping errors. This is attributed to it being quite sensitive to errors affecting the most significant bits of input data and the 1-bit only weights. In contrast, we expect increased robustness from the bitstream-processing paradigm. Hence, bit-flip error injection has been considered to measure the robustness of the proposed bitstream processing framework to errors that can happen during memory accesses of images and weights [4, 8]. Since binary symbol errors affect the weights and pixel values differently, our study investigates those two kinds of errors separately.

For the case in which errors affect pixel values, some binary symbols representing the pixels are randomly selected. The pixels of test images are manipulated in 8-bit traditional binary representation for deterministic BNN. For BSBNN, the image data is represented in \(N = 8\) (BSBNN8) or \(N = 16\) (BSBNN16) sized streams. Figure 7(a) depicts how error injection is defined in terms of the percentage of flipped binary symbols. (a) BNN model trained without error is tested for image bit-flips. (b) BNN model trained without error is tested for weight bit-flips. (c and d) Models are trained in the presence of (image or weight) errors, with the processing paradigm used at test time.

With 2 HL, as has been considered in this study. During our simulations, we first consider a single model. It is obtained using deterministic BNN in the training phase. The forward path of the network uses binarised weights. However, the weight update is performed considering a floating-point version of the weights, which are then binarised to define the forward path. The network is made of four layers, each of them including 784-1024-1024-10 neurons, respectively. Squared hinge loss is chosen, which outperforms cross-entropy loss in terms of validation accuracy. An exponential learning rate decay is applied in the range of \([3 \times 10^{-7}, 3 \times 10^{-7}]\). The batch size is set to 100. Dropout is implemented with \(p = 0.2\) for the input layer, and with \(p = 0.5\) for hidden ones. The best performing model is saved by considering the best validation accuracy, 98.43\%, at the 968th epoch over 1000 epochs. The deterministic BNN test accuracy is 98.27\%.

From a data representation point of view, traditional binary data in BNN is known to be quite vulnerable to bit-flipping errors. This is attributed to it being quite sensitive to errors affecting the most significant bits of input data and the 1-bit only weights. In contrast, we expect increased robustness from the bitstream-processing paradigm. Hence, bit-flip error injection has been considered to measure the robustness of the proposed bitstream processing framework to errors that can happen during memory accesses of images and weights [4, 8]. Since binary symbol errors affect the weights and pixel values differently, our study investigates those two kinds of errors separately.

For the case in which errors affect pixel values, some binary symbols representing the pixels are randomly selected. The pixels of test images are manipulated in 8-bit traditional binary representation for deterministic BNN. For BSBNN, the image data is represented in \(N = 8\) (BSBNN8) or \(N = 16\) (BSBNN16) sized streams. Figure 7(a) depicts how error injection is defined in terms of the percentage of flipped binary symbols. We observe in Figure 7(b) that BNN and BSBNN achieve close performance, while BSBNN16 demonstrates a non-negligible, but significantly reduced sensitivity to weight bit errors.

As the second round of simulations, we have investigated how accounting for the errors and the processing paradigm at training time affects the robustness to errors at the test time. Therefore, different models have been trained with different bit-flipping rates (5, 10, or 20\%) at training. This is done using either the conventional or the bitstream processing computational paradigm, and the errors affect either the image pixels or the weights. The resulting models are tested with the same computational paradigm than the one used at training, and their test accuracy is reported in Figure 7(c) and (d), as a function of the bit-flip percentage at test time. In Figure 7(c), we observe that injecting errors on image pixels at training increases the robustness to errors at test time by a large margin (about 30\% accuracy improvement at 30\% error rate) and that the bitstream processing paradigm is significantly more robust to errors than the conventional one. With regard to the errors affecting weights, we observe in Figure 7(d) that accounting for errors is largely beneficial when they are also present at test time but the performance is penalized in absence of errors at test time.

To compare the complexity of the conventional and bitstream-based computational paradigms, a hardware simulation framework was constructed in MATLAB Simulink with design primitives from Xilinx System Generator. For both BNN and BSBNN hardware simulations, the input layer data were copied from MATLAB workspace. BPE-based random bitstreams were prepared on the software side, and they adopted a Bernoulli distribution for BSBNN following the procedure in [4]. Parallel co-simulation was performed by feeding the hardware model associated with the feed-forward hidden layers. For single neuron hardware shown in Figure 4, the total count of the look-up table and flip logic was 101 for \(N = 8\)-bit BSBNN, and 147 for BNN, which represents a gain of approximately 30\%, according to the design synthesis in the Vivado tool setting the target device as Zyq5-7000.

**Conclusion:** A stochastic bitstream processing binarised neural network is presented. The first contribution of the study is proposing subtraction-free activation using the truncated MOD-T counter in the presence of bitstream usage. The second contribution is underlining data error robustness of bitstream processing over the deterministic approach. Injecting random bit-flips into the data of deterministic BNN decreases the accuracy values significantly. Stochastic bitstream representation of image pixels and weights provides noticeable robustness for soft errors on data.

**Acknowledgments:** This Ph.D. dissertation is supported by ITU BAP with grant MKD-2018-41532. The study is being collaboratively continued in UCLouvain, ICTEAM during Sercan Aygın’s Ph.D.-related research visit in the group lead by Prof. C. De Vleeschouwer.

© 2021 The Authors. Electronics Letters published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

Received: 17 September 2020 Accepted: 2 November 2020 doi: 10.1049/el2.12045

**References**

1. Miller, A.J., Brown, A.W., Mars, P.: Adaptive logic circuits for digital stochastic computers. *Electron. Lett.* 9(21), 500–502 (1973)
2. Li, B., Najafi, M.H., Lilja, D.J.: Low-cost stochastic hybrid multiplier for quantized neural networks. *ACM J Emerg. Technol. Comput. Syst.* 15(2), 18 (2019)
3. Alaghi, A.: The logic of random pulses, Dissertation, University of Michigan (2015)
4. Canals, V., et al.: A new stochastic computing methodology for efficient neural network implementation. *IEEE Trans. Neural Networks Learn. Syst.* 27(3), 551–564 (2016)
5. Ren, A., et al.: SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing. *ASPLoS' 17: Proceedings of the Twenty-Second International Conference on Architectural Support for
Ardakani, A., et al.: VLSI implementation of deep neural network using integral stochastic computing. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **25**(10), (2017)

Courbariaux, M., et al.: Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1. Preprint, arXiv: 1602.02830 (2016)

Hirtzlin, T., et al.: Stochastic computing for hardware implementation of binarized neural networks. *IEEE Access* **7**, 76394–76403 (2019)

Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. Preprint, arXiv:1708.07747 (2017)

Godse, A.P., Bakshi, U.A.: *Digital and Linear Integrated Circuits*. Technical Publications, Pune, Maharashtra (2009)

LeCun, Y., Cortes, C., Burges, C.J.: MNIST handwritten digit database. AT&T Labs, [Online]. http://yann.lecun.com/exdb/mnist (Accessed: 24th August, 2020)

Liu, Y., et al.: An energy-efficient stochastic computational deep belief network. *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1175–1178. IEEE, Piscataway, NJ (2018)

Sanni, K., et al.: FPGA implementation of a deep belief network architecture for character recognition using stochastic computation. *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, March 2015, Baltimore, MD, pp. 1–5. IEEE, Piscataway, NJ (2015)

Babu, A.V., et al.: Stochastic learning in deep neural networks based on nanoscale PCMO device characteristics. *Neurocomputing* **321**, 227–236 (2018)

Li, B., Najafi, M.H., Lilja, D.J.: An FPGA implementation of a Restricted Boltzmann Machine classifier using stochastic bit streams. *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2015, Toronto, Canada. IEEE, Piscataway, NJ (2015)

Kim, K., et al.: Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, Austin, TX. IEEE, Piscataway, NJ (2016)

Xie, Y., et al.: Fully-parallel area-efficient deep neural network design using stochastic computing. *IEEE Trans. Circuits Syst. II Express Briefs* **64**(12), 1382–1386 (2017)

Liu, Y., et al.: A stochastic computational multi-layer perceptron with backward propagation. *IEEE Trans. Comput.* **67**(9), 1273–1286 (2018)