2.2

Efficient Deep Learning Approach for Fault Detection in the Semiconductor Industry

Liliana Andrade[1], Thomas Baumela[1], Frédéric Pétrot[1], David Briand[2], Olivier Bichler[2] and Marcello Coppola[3]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, France
[2]CEA-LIST, France
[3]STMicroelectronics, France

Abstract

The semiconductor industry is a very cost sensitive industry and yield is key to profitability. The ability to analyse and detect the faulty parts at several manufacturing steps is also very important to ensure the quality of the delivered integrated circuits. Several factors as alignments, shifts or masks rotations can lead to errors during the front-end step (wafer fabrication), or others causes such as fingerprints, scratches and stains can cause cosmetic damage during the back-end step (silicon packaging). Therefore, an automatic visual inspection is required to ensure that the parts are free of any defects. In this chapter, we focus specifically on classifying wafer maps according to predefined defaults. We propose a platform which aims at making the classification process more energy efficient, by means of the interconnection of two hardware parts. The first one, the microprocessor STM32MP1, is responsible for image pre-processing and for offloading inference to a dedicated hardware accelerator. The second one, the hardware accelerator, is implemented in a Xilinx Zybo Z7-20 FPGA and uses a quantized neural network model. Preliminary results show that, for this low throughput applications that has a limited number of classes, the solution presented in this article can classify in real-time with accuracy above 80% using limited resources.

Keywords: classification, wafer maps, deep learning, quantized neural networks, embedded artificial intelligence, HW/SW integration, hardware acceleration of AI, high-level synthesis, field-programmable gate array.

2.2.1 Motivation: The Wafer Fault Classification Problem

Defect inspection and classification are significant steps of most manufacturing processes in the semiconductor industry. These steps are needed during the front-end process, when wafers come out of the foundry, and during the back-end process, when packaging individual chips. Having a proper inspection to partition wafers, dies, or packages in correct vs incorrect items is needed not only to ensure delivering working packaged chips to customers, but also to improve the quality of the manufacturing process. Similarly, accurate classification of the defaults, be they during the front-end or back-end process, is key to high yield. Indeed, yield management and yield learning help the manufacturing process engineers in determining the causes of abnormal fabrication. To ensure a high-level of quality and yield, still today, many of the inspection phases are performed visually, by humans [1]. Given the throughput on the production lines, the actual inspection can only be done on samples, which leaves quite some room for improvement.

In this chapter, we focus on the front-end process, and more specifically on the detection and classification of wafer defects using deep neural networks (DNN), which have proven to be efficient for classifying images. Early detection of defects at the back-end process, during which wafers are produced and electrically tested, can help to readjust certain parameters in the production line, to increase yield and thus reduce costs. This explains why a lot of effort has been devoted to this topic since the infancy of mass production of integrated circuits.

Once wafers have been fabricated and the electrical inspection step carried out, different 2D images are generated indicating which dies are working properly and which are not. These images, known as wafer maps, must then be inspected to extract their features and classified into different categories. It will allow to determine if all chips on the wafer can go to packaging, in the case of lack of defects, or if the wafer presents a specific failure pattern indicating an issue during some of the production stages. Specific failure patterns may indicate the root cause of a problem. For instance, when several faulty dies appear randomly on a wafer without

following any specific pattern, the problem may come from the presence of dust particles transferred to the wafer surface; when a ring of faulty dies is present, the issue is due to a misalignment of several layers during photolithography; when defects are located in the centre of the wafer or following a donut pattern they may indicate a non-uniform application of forces to smooth and flatten, silicon wafer, a non-uniform temperature distribution or also a problem during oxidation; when some streaks run across the wafer surface it may be due to a human error in handling equipment or due to an issue during the chemical-mechanical polishing stage; or even when falling dies are located near to the edge of the wafer, this can indicate an issue during etching or a non-uniform cleaning.

Wafer data, because it could leak information on the process and possibly its yield, is very sensitive, and manufacturer are unwilling to share it. The work presented in this chapter will therefore be based on a public and open access dataset that has been "anonymized" and then donated to the community by TSMC: the WM-811K wafer map dataset [2]. The wafer production line throughput is low compared to general purpose computer vision applications, and the construction of the wafer maps is the result of a process which also involves test equipment [3]. However, since the machines are working 24/7, 365 days a year for continuous monitoring of the production quality, we seek solutions that are both accurate and low power. In addition to these constraints, and given the confidentiality of the data processed, being able to perform on-device classification instead of sending the data to a remote server is also of importance. To that end, we choose to investigate the use of highly quantized artificial neural networks to be implemented on small industry grade micro-controllers, possibly enhanced with hardware accelerators on FPGA. After having defined in this section the problem at hand, we organize this chapter as follows. Section 2.2.2 presents past works related to it while section 2.2.3 details the requirements and functional specifications of the system. Section 2.2.4 presents the method we propose and some preliminary results. Finally, section 2.2.5 wraps-up the chapter and presents possible extensions to this work.

2.2.2 Related Works

Recently, many authors have proposed different techniques for automatic detection and classification of failure wafer patterns, either using Machine Learning (ML) or Deep Learning (DL) techniques.

On the one hand, we find some approaches where a feature extraction is performed on wafer maps to obtain a reduced representation ready to be analysed and classified. In this context, different authors have highlighted the use of ML techniques, generally applied in computer vision. Some of these techniques allow for example the feature extraction using the Hough transform, the generation of probability distributions used to define specific-faulty regions, or the use of k-nearest-neighbour classifiers to distinguish faulty patterns. A key research implementing ML techniques is presented in [2]. It introduces a new set of features which, requiring low computation and storage, is used to obtain a reduced representation of wafer maps, to identify wafer maps failure patterns and to support recovery of similar failures in other wafer maps. The proposed approach applies support vector machines as classifier, preserves the rotation-invariant attribute in wafers maps and reduces the computational cost with respect to different approaches carrying spatial analysis between features maps. This work is considered as a reference because it is at the origin of the WM-811K dataset used for the experiments presented in this chapter.

On the other hand, we find several approaches implementing DL techniques, which have seen an exponential growth in the last years. For example, Alawieh et al. [4] propose a wafer map classification using deep selective learning and implement a reject technique where model refrain from predicting class label when the miss-risk is high. This can usually happen when during classification some wafers show default patterns that have never been seen during training. Also, Convolutional Neural Networks (CNN) have demonstrated great potential to recognize and classify patterns without carrying manual feature extractions. Using convolution layers, they can perform automatic feature extraction; using pooling layers they can summarize the last extraction by reducing the features maps size; and using fully-connected layers they can efficiently classify patterns into well-separated categories. As described below, several studies have been conducted to detect and classify wafer defect patterns using CNN. Kyeong and Kim [5] address the problem of detecting mixed-type defect patterns, this means to have different defect patterns combined in the same wafer. Authors propose a single approach building individual CNN-based classification models for each pattern and determining the final class by combining the results of multiple individual models. Jang et al. [6] implement a one-vs-one model that uses a CNN as base classifier. Their technique consists of determining a weighted mean score from failure bit count wafer maps (grey-scale images) and then, based on this score, they determine the presence

or absence of failures. Failure detection is performed calculating the score proximity in relation to a data group learned in a feature space. In principle to address a high-quality classification of wafers maps using CNN, having a set of examples containing a large quantity of labelled patterns is required. These patterns are the key to fit the parameters in DNN before inference. Sometimes, even having a large quantity of patterns is still not enough, but it is also required to have as much as possible a balanced dataset. That is, a set of examples where the proportion of wafer maps in each class is almost the same. As it is difficult to achieve, domain expert engineers are required for labelling data coming from the manufacturing process in the form of wafer maps. As this process represents a significant cost, several authors [7], [8], [9], [10], [11] have worked on different techniques to avoid the use of unbalanced datasets and to automatically increase the sets of examples reducing the intervention of experts.

Although many of the presented solutions achieve high performance, none of them have been specifically designed to be implemented on small embedded devices. We are interested to address this challenge by using quantized neural networks, which in our knowledge have never been used for classifying faulty wafer maps.

2.2.3 Target Platform Requirements

Based on these experiences and considering the increasing need to detect and classify defects in an automatic, real-time and power-efficient way, we define the requirements for an automatic wafer defect detection platform targeting high-power efficiency and real-time inference. While this platform should be generic enough to support different applications, it will primarily target detection and classification of faulty wafer maps. We rely on the requirements presented below to enable industrial scanning equipment to efficiently address the aforementioned problem.

- *Define a deep learning classification platform that can be programmed and its hardware partially reconfigured*: When we refer to deep learning, we evoke the new programming paradigm where humans provide input data and expected responses, and a layered system, better known as DNN, processes inputs and stores a meaningful representation that can be later used to perform tasks automatically, for example recognizing a set of images. The stage where the system transforms input data and stores a representation of it in form of parameters, also known as

weights, is called *learning*. The appropriate selection of the weights associated with each neural network layer is performed by first assigning random values and computing a temporal prediction of the network from a set of inputs, then comparing that prediction with respect to the expected response (through a loss function), and using a back-propagation algorithm (usually implemented by an optimizer) to adjust the weights in the correct direction [12]. Once the system has learned an enough representative input dataset, it can be used to perform automatic classification tasks also called *inference*. For learning, we will follow the approach in which the neural network parameters are computed and refined off-line, before implementing a HW/SW model in the industrial equipment via micro-controllers and small reconfigurable devices.

- *Design an efficient DNN model to be implemented in hardware*: We will focus on neural network models with small number of parameters and on quantization techniques to increase power efficiency during classification, without neglecting high-throughput. On modern high-end front-end equipment, the throughput in terms of wafer per hour is between 150 and 300. Assuming that the electrical characterization and test equipment is dimensioned to work at that same throughput, this means that the analysis must be performed in a 20 s to 40 s time frame. It is thus neither useful nor economically sensible to reach for throughputs like those required by general purpose video processing.
- *Use real images which undergo classical linear time pre-processing before being fed to a DNN implemented by the classification platform*: We will use the WM-811K public dataset provided by the Multimedia Information Retrieval (MIR) laboratory. It contains 811457 real wafer maps collected from 46393 lots of real-world fabrication. The 2D images provided in this dataset have different sizes and 172951 ($\sim$20%) of these images were manually labelled by domain experts using nine patterns (Figure 2.2.1): no-defects (85.2%), edge-ring (5.6%), edge-local (3.0%), center (2.5%), local (2.1%), scratch (0.7%), random (0.5%), donut (0.3%) or near-full (0.1%). As observed by different authors, the challenge with this dataset is that it is unbalanced, then image pre-processing and data augmentation will be required to improve classification accuracy.

2.2.4 HW/SW System and Methodology

2.2.4.1 Industrial HW/SW System for On-Device Inference

We propose a platform allowing the integration, in a reconfigurable device, of a neural network model trained upstream with a set of reference wafer maps, as well as the classification of new faulty wafers by means of a dedicated HW/SW architecture.

The hardware architecture of the platform consists of two main boards (Figure 2.2.2). One STM32MP1 board interfacing with the physical world (i.e., the wafer production line), and one Zybo-Z7 board for the wafer fault classification. The STM32MP1 is an industrial grade master board including an ARM dual-core Cortex-A7 and an additional Cortex-M4, DDR memory and a good set of peripherals, in particular a 1GBps Ethernet chipset, USB device connectors and an HDMI output connector. The Zybo-Z7 embeds the XC7Z020 SoC from Xilinx, featuring 667MHz dual-core Cortex-A9 processor, 1GB DDR3L memory, a 1GBps controller as well as an FPGA. Both boards communicate through a GBps Ethernet link, allowing the STM32MP1 to send input image to the Zybo-Z7 taking care of the inference and sending back the results through the Ethernet link. An inference cycle thus consists of: (1) receive an image from the test equipment, (2) apply scale and crop filters to get the image to the correct dimensions, (3) send it to the Zybo-Z7, (4) make the inference on the Zybo-Z7 and (5) get back the results to the STM32MP1. The Zybo-Z7 is the heart of the inference process. It integrates a hardware implementation of a neural network, taking advantage of the high parallelisation capabilities the FPGA offers. The network is integrated with a message-based interface consisting of a pair of RX/TX FIFOs connected to high-performance Scatter-Gather (S/G) DMA engine. These FIFOs are used by the network to receive configuration weights and inputs as well as send inference outputs. The DMA engine is used by the software to efficiently exchange those data and control and status commands to drive the neural network.

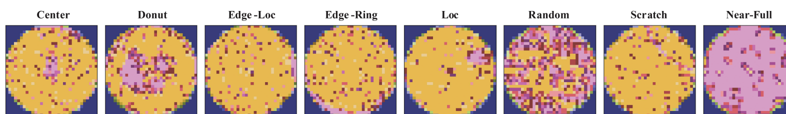

Figure 2.2.1 Example of classified wafer maps.

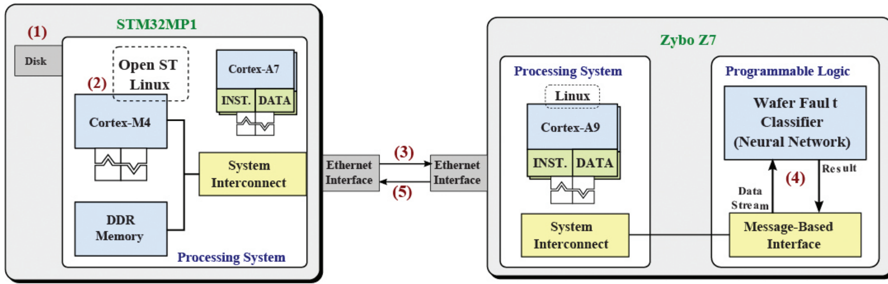

Figure 2.2.2 The platform hardware architecture.

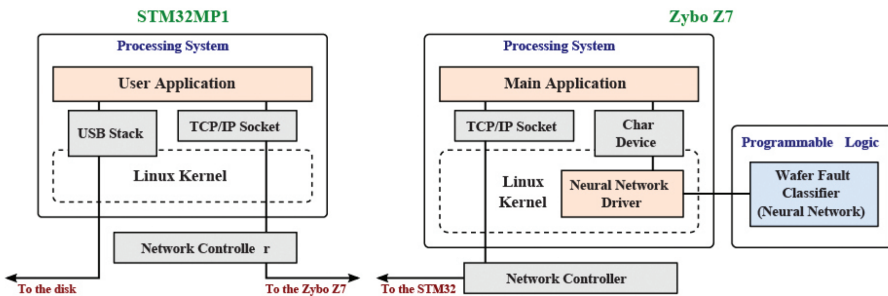

Figure 2.2.3 The platform software architecture.

The software architecture of the platform is also composed of two parts (Figure 2.2.3). The STM32MP1 runs a Linux operating system pre-configured by ST tools. It includes an application accessing the file system to send inputs to the Zybo-Z7 through the Ethernet link. It also configures the neural network by sending the weights to the Zybo-Z7. The Zybo-Z7 runs a minimal Linux operating system built using PetaLinux tools. It includes a custom driver integrating the neural network in the Linux environment, allowing user applications to control it. To do so, the driver provides a char device interface in which applications can read and write into to control the neural network. The driver implements those read and writes by driving the S/G DMA engine included in the neural network interface.

With the neural network accessible by user applications, the main application configures the Ethernet link with the STM32MP1. Once the communication with the STM32MP1 is established, the application forwards weights and inputs to the neural network and receives outputs from it.

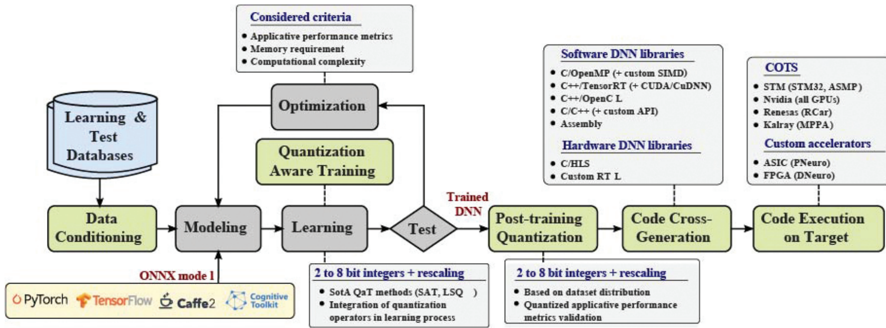

Figure 2.2.4 System design and optimization process using N2D2.

2.2.4.2 Neural Network Building and Training Using N2D2

Current research on deep neural networks extensively uses different frameworks, such as PyTorch [13] and Tensorflow [14], which allow the development of neural networks and ML algorithms. Some other frameworks, such as Keras [15], QKeras [16] or Larq [17], are interfaces to these frameworks. Unfortunately, once a network has been designed and trained, there is a gap towards its optimized deployment in an embedded and hardware constrained system. The recent development of libraries built to ease the deployment and optimization of DNN, such as TensorRT [18] or TFLite [14], reflects the rising trend of hardware integration requirements. The main weakness of these libraries remains the limitation to certain target platforms and the possible optimizations that can be applied, which are greatly linked to proprietary solutions. To train and generate a first neural model, we use the N2D2 deep learning framework [19], which reduces this gap by providing an innovative optimization method to the system designer.

N2D2 is hardware agnostic while being able to directly target most common computing architectures and parallel run-time software. As shown in the high-level view of the system design process enabled by N2D2 (Figure 2.2.4), this framework integrates a generic database handling and data processing dataflow.

The N2D2 learning core is close to the standard deep learning frameworks with the support of typical layers, operators and learning rules. Its execution on x86 and ARM processor is accelerated thanks to C++/OpenMP kernels while execution on NVidia GPUs is supported thanks to cuDNN and custom CUDA Kernels. Moreover, the N2D2 core also supports spikes simulations modelling. The input model representation of the N2D2 framework can be

made through an INI description file or thanks to the Open Neural Network Exchange (ONNX) format [20], allows the user to load a pre-trained neural network from another deep learning framework.

Among the key features of the N2D2 framework, the integrated quantization module remains one promising technique to optimize a deep learning model for a wide range of hardware accelerators. Quantization refers to the process of reducing the number of bits that represent a number, without performance degradation. In the context of deep learning, the predominant numerical format used for research and for deployment has so far been 32-bit floating point, or FP32. However, the desire for reduced bandwidth and compute requirements of models has driven research into using lower-precision numerical formats. It has been extensively demonstrated that weights and activations can be represented using 8-bit integers (or INT8) without incurring significant loss in accuracy. The use of even lower bit-widths, such as 4/2/1-bits, is an active field of research that has also shown great progress. The more obvious benefit from quantization is significantly reduced bandwidth and storage. For instance, using INT8 for weights and activations consumes 4x less overall bandwidth compared to FP32. Additionally, integer compute is faster than floating point compute. It is also much more area and energy efficient. Note that very aggressive quantization can yield even more efficiency. If weights are binary{-1, 1} [21], [22] or ternary{-1, 0, 1} [23], [24], then convolution and fully-connected layers can be computed with additions and subtractions only, removing multiplications completely. A lot of techniques have been proposed recently to quantize neural networks. These techniques can be classified into two types: Post Training Quantization (PTQ), which quantizes both weights and activations for faster inference, without requiring to re-train the model; Quantization Aware Training (QAT), which models quantization during training and can provide higher accuracies than post quantization training schemes. Both techniques are integrated into N2D2. However, QAT is currently the best technique to provide highest accuracies for heavily quantized networks, with bit-widths as low as 4/2/1-bits for weights and/or activations. N2D2 integrates both Learned Step Size Quantization (LSQ) [25] and Scale-Adjusted Training (SAT) [26] [27] state-of-the art QAT algorithms, the latter one being one of the most promising solutions, both in term of implementation complexity, flexibility and accuracy. The quantization aware training in N2D2 is performed by a full precision learning phase with weights clamping; and quantization learning phase, with the same hyperparameters

by using a transfer learning method from the previously clamped weights.

We use N2D2 to build and train a first neural network model. Three kinds of networks are tested, all based on convolutional layers, with various complexity. First, a simplified AlexNet made of 3 convolutional layers with MaxPooling, followed by 2 fully connected layers and a SoftMax activation layer. Second, a simplified VGG with 5 convolutional blocks (made of 2 or 3 convolutional layers) of increasing size, with MaxPooling, followed by 2 fully connected layers and a SoftMax activation layer. Third, a MobileNet V1, which uses depthwise separable convolutions in place of the standard convolutions to provide lighter models. The version used here has 27 layers (26 groups of 'convolution + batch normalization' and 1 fully connected + softmax layers).

Before training, the images in the WM-811K dataset are homogenized. They are rescaled to a common size. Sizes of 42x42 pixels and 64x64 pixels were tested. As mentioned above, the dataset is very unbalanced, as the 'no defect' class has much more images than the others. We then decided to limit the scope of the application to the first eight classes and discard this last class, for a total of 17625 training images and 7894 test images. We help the network converge to a correct solution, by applying data augmentation (Random rotation and horizontal/vertical image flipping) during the training phase. To decrease the memory usage, images are also transformed to Grayscale and normalized (colour range moved from 0-255 to 0-1) before applying the data augmentation strategy.

After training, we observed that although the topology of the simplified AlexNet is much simpler, it uses much more weights and biases than the other networks for a total of 2,478,632 numbers to store. The performances were not better, so this network was abandoned. The simplified VGG network requires around 600,000 parameter storage (depending on the presence of batch normalization layers) which is much lighter. In comparison, the MobileNet V1 requires a little bit more with 823,752 parameters. The best performances were obtained with the VGG network with 98.2% recognition on the training set and 81.0% on the test set (1 hour and 38 minutes training and 2000 training epochs). After applying post-training quantization (8-bits), the performance on the test set remained at 80.4% (0.6% loss) for images size of 42x42 pixels.

2.2.4.3 Neural Network Export and FPGA Implementation Used for Inference

The inference platform is implemented in hardware, targeting an FPGA based platform (the Zybo-Z7 board in our proposed platform). The implementation is performed using Vivado-HLS, Xilinx's High-Level-Synthesis tool from C++ programs. The neural network implementation process is divided in two phases. First, a C++ export, which could be for example the reference N2D2 export. This export is expected to provide an implementation in which the structural parameters of the layers, e.g., loop boundaries, are passed as template parameters. This allows heavy compile-time inlining, optimization and loop unrolling. Second, the neural network FPGA implementation, which consists of modifying the C++ implementation to make it fit for HLS synthesis. Indeed, the C++ implementation does not target HLS due to language and library limitations such as dynamic memory allocation, printfs, file system access. In addition, a set of pragmas must be added to guide the synthesis tool in order to get a properly optimized network. The most important pragmas are array modifiers and loop unrolling directives. Array modifiers (called *array_map*, *array_partition* and *array_reshape* in Vivado-HLS) are used to optimize the structure of arrays by splitting them in smaller arrays. It also allows to merge small words, for instance our 2-bit wide weights, into bigger words allowing to fetch several small words in the same cycle. Having efficiently structured arrays, in particular the one storing weights in Block RAMs (BRAMs), allows to completely unroll some loops as data contained in these arrays can be accessed in one clock cycle by all the parallelized iterations. Unrolling loops is performed using the *unroll* pragma. It requires reordering the nested loops in a way that will allow the synthesis tool to properly unroll and take advantage of the array structures we defined. Optimizing the usage of BRAM is key to make a network fit entirely in an FPGA, even with low parallelization settings, to avoid time and power consuming external memory accesses.

We report the resource usage for a preliminary experiment on a fully-connected input layer for 42x42 wafer maps (Figure 2.2.5). These results show the evolution of resource usage against the bit-size of weights. The important point enlighten by these experiments is that lowering the size of weight is key to make a network fit inside a given FPGA. Of course, reducing the weight size leads to a loss of accuracy, though this loss can be mitigated by increasing the number of neurons in the network. For instance, reducing the weight size on a 100-neuron network with 8-bit weights to a 2-bit weights

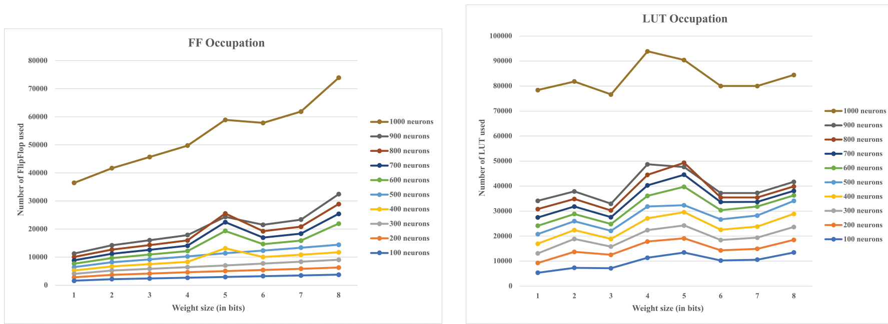

Figure 2.2.5 Resource occupation (FF and LUT) for 42x42 wafer maps.

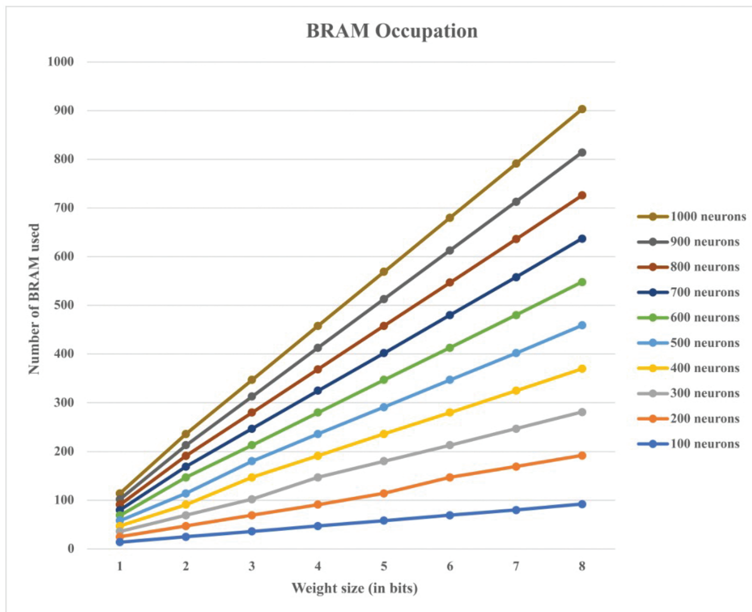

Figure 2.2.6 BRAM occupation for 42x42 wafer maps.

network allows to increase the number of neurons to 400 while occupying the same amount of BRAMs (Figure 2.2.6).

2.2.5 Conclusion

Process control in the semiconductor industry is a major issue. In this article, we present the approach we propose, that is suited to the low throughput of the

wafer production line. It is an AI hardware/software based solution running on a small industry grade device which aims at analysing wafer maps in real-time. We report preliminary experiments showing first that highly quantized neural networks, when trained appropriately, can reach high accuracy, and second, that the hardware implementation of these networks can be very resource efficient.

The next step is to smooth the integration between the tools and generalize hardware support to larger classes of network layers.

Acknowledgements

This work is conducted under the framework of the ECSEL AI4DI "Artificial Intelligence for Digitising Industry" project. The project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826060. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Austria, Czech Republic, Italy, Latvia, Belgium, Lithuania, France, Greece, Finland, Norway.

References

[1] M.-J. Wang and C.-L. Huang, "Evaluating the Eye Fatigue Problem in Wafer Inspection," IEEE Transactions on Semiconductor Manufacturing, vol. 17, no. 3, pp. 444-447, 2004.

[2] M.-J. Wu, J.-S. Jang and J.-L. Chen, "Wafer Map Failure Pattern Recognition and Similarity Ranking for Large-Scale Data Sets," IEEE Transactions on Semiconductor Manufacturing, vol. 28, no. 2, pp. 1-12, 2015.

[3] F. Duvivier, "Automatic detection of spatial signature on wafermaps in a high volume production," in International Symposium on Defect and Fault Tolerance in VLSI Systems, Albuquerque, NM, USA, 1999.

[4] M. B. Alawieh, D. Boning and D. Z. Pan, "Wafer Map Defect Patterns Classification Using Deep Selective Learning," in ACM/EDAC/IEEE Design Automation Conference, Virtual Event, USA, 2020.

[5] K. Kyeong and H. Kim, "Classification of Mixed-Type Defect Patterns in Wafer Bin Maps Using Convolutional Neural Networks," IEEE Transactions on Semiconductor Manufacturing, vol. 31, no. 3, pp. 395-402, 2018.

[6] J. Jang, M. Seo and C. O. Kim, "Support Weighted Ensemble Model for Open Set Recognition of Wafer Map Defects," IEEE Transactions on Semiconductor Manufacturing, vol. 33, no. 4, pp. 635-643, 2020.

[7] T. Nakazawa and D. V. Kulkarni, "Wafer Map Defect Pattern Classification and Image Retrieval Using Convolutional Neural Network," IEEE Transactions on Semiconductor Manufacturing, vol. 31, no. 2, pp. 309-314, 2018.

[8] M. Saqlain, Q. Abbas and J. Y. Lee, "A Deep Convolutional Neural Network for Wafer Defect Identification on an Imbalanced Dataset in Semiconductor Manufacturing Processes," IEEE Transactions on Semiconductor Manufacturing, vol. 33, no. 3, pp. 436-444, 2020.

[9] J. Shim, S. Kang and S. Cho, "Active Learning of Convolutional Neural Network for Cost-Effective Wafer Map Pattern Classification," IEEE Transactions on Semiconductor Manufacturing, vol. 33, no. 2, pp. 258-266, 2020.

[10] R. Wang and N. Chen, "Defect Pattern Recognition on Wafers using Convolutional Neural Networks," Quality and Reliability Engineering International, vol. 36, no. 4, pp. 1245-1257, 2020.

[11] T. -H. Tsai and Y. -C. Lee, "A Light-Weight Neural Network for Wafer Map Classification Based on Data Augmentation," in IEEE Transactions on Semiconductor Manufacturing, vol. 33, no. 4, pp. 663-672, Nov. 2020. Available online at: https://doi.org/10.1109/TSM.2020.3013004.

[12] F. Chollet, Deep Learning with Python, USA: Manning Publications Co., 2017.

[13] A. Paszke, S. Gross et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Advances in Neural Information Processing Systems, Vancouver, Canada, 2019.

[14] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. Available online at: http://tensorflow.org/.

[15] "Keras: The Python Deep Learning API." Available online at: https://keras.io/. [Accessed 2021].

[16] "QKeras: A Quantization Deep Learning Library for Tensorflow Keras," Available online at: https://github.com/google/qkeras. [Accessed 2021].

[17] "Larq: Python Library for Training BNN," 2021. Available online at: https://larq.dev/.

[18] "NVidia TensorRT: Programmable inference accelerator." Available online at: https://developer.nvidia.com/tensorrt.

[19] O. Bichler, D. Briand et al., “N2D2-neural network design & deployment,” CEA LIST, 2017. Available online at: https://github.com/CEA-LIST/N2D2/raw/master/manual/manual.pdf.

[20] J. Bai, F. Lu et al., “ONNX: Open Neural Network Exchange.” Available online at: https://github.com/onnx/onnx. [Accessed 2021].

[21] I. Hubara, M. Courbariaux et al., “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activation,” J. Mach. Learn. Res., vol. 18, no. 1, p. 6869–6898, 2017.

[22] M. Rastegari, V. Ordonez et al., “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks,” in European Conference in Computer Vision, 2016.

[23] L. Fengfu and L. Bin, “Ternary Weight Networks,” 2016. Available online at: https://arxiv.org/abs/1605.04711.

[24] H. Alemdar, V. Leroy et al., “Ternary Neural Networks for Resource-Efficient AI Applications,” in 30th International Joint Conference on Neural Network, Training code available at https://github.com/slide-lig/tnn-train, 2017.

[25] S. K. Esser, J. L. McKinstry et al., “Learned Step Size Quantization,” 2019. Available online at: http://arxiv.org/abs/1902.08153.

[26] J. Qing, Y. Linjie and L. Zhenyu, “Towards Efficient Training for Neural Network Quantization,” 2019. Available online at: https://arxiv.org/abs/1912.10207.

[27] J. Qing, Y. Linjie and L. Zhenyu, “Rethinking Neural Network Quantization,” 2020. Available online at: https://openreview.net/forum?id=HygQ7TNtPr.