Triple Fixed-Point MAC Unit for Deep Learning

Madis Kerner*, Kalle Tammemäe*, Jaan Raik*, Thomas Hollstein*†
*Tallinn University of Technology, Tallinn, Estonia
†Frankfurt University of Applied Sciences, Frankfurt, Germany
email: madis.kerner@taltech.ee, kalle.tammemae@taltech.ee, jaan.raik@taltech.ee, hollstein@fb2.fra-uas.de

***Abstract*—Deep Learning (DL) algorithms have proved to be successful in various domains. Typically, the models use Floating Point (FP) numeric formats and are executed on Graphical Processing Units (GPUs). However, Field Programmable Gate Arrays (FPGAs) are more energy-efficient and, therefore, a better platform for resource-constrained devices. As the FP design infers many FPGA resources, it is replaced with quantized fixed-point implementations in state-of-the-art. The loss of precision is mitigated by dynamically adjusting the radix point on network layers, reconfiguration, and re-training. In this paper, we present the first Triple Fixed-Point (TFxP) architecture, which provides the computational precision of FP while using significantly fewer hardware resources and does not need network re-training. Based on a comparison of FP and existing Fixed-Point (FxP) implementations in combination with a detailed precision analysis of YOLOv2 weights and activation values, the novel TFxP format is introduced.**

I. Introduction

Deep Learning (DL) algorithms have been successfully deployed in various domains, including image recognition, natural language detection, among others [1], [2]. These algorithms automatically extract the input signal's essential features and do not rely on domain expert knowledge and manual pre-processing.

A DL algorithm has a layered structure and comprises various types of layers. Each layer includes neurons that receive and process data from the previous layer and feeds the next layer. This kind of build-up provides excellent possibilities for acceleration: all the neurons can execute in parallel. Therefore, typical platform for running DL is PC based, using Graphical Processing Unit (GPU).

Due to the success of DL, contemporary research explores the possibilities to execute these algorithms in resource constrained devices as well: Field Programmable Gate Arrays (FPGAs) form an excellent platform for this. While FPGAs are power efficient compared to GPUs, the DL algorithms rely on Floating Point (FP) representations of parameters, which infer a lot of Hardware (HW) resources. Despite the search for efficient FP support in FPGAs [3], the available Digital Signal Processing (DSP) slices are still more suitable for fixed-point operations.

Although it is undoubtedly possible to perform FP calculations on FPGAs, the inferred HW resources are high: constructing a half-precision multiply-accumulator, which is a typical computational unit in Artificial Neural Networks (ANNs), requires three DSPs and several hundred LUTs and registers [4].

Contemporary research tries to overcome this obstacle and explores different approximation techniques to get rid of FP representations. Typically, it means quantizing the network parameters to fixed-point numbers of some sort, or binary values in extreme cases [5]. Many works have achieved reasonable inference accuracy using quantized networks, suggesting that the precision of FP is not required. However, there are proposals that better precision than the deep-quantization is necessary [6].

In this paper, we propose a novel Triple Fixed-Point (TFxP) based Multiply-Accumulate (MAC) unit for ANNs. TFxP extends the Dual Fixed-Point (DFxP) format [7] by introducing one additional range. This extra middle range allows extending the usable dynamic range of the format, while the added HW cost is small.

We show that TFxP can be used as the drop-in replacement for FP. To justify the proposal, we first analyzed the required representation range of the YOLOv2 network [8]. This network comprises 23 convolutional layers, which all make heavy use of MAC operations. Further, as our simulations show, the YOLOv2 network achieves the same inference precision with TFxP format as with FP and does not require retraining to accomplish that.

The rest of the paper is organized as follows: Section II performs the design space exploration by analysing the weight and activation values of YOLOv2 network, Section III presents the TFxP format, Section IV analysis and compares the average precision of converted network, Section V presents the XILINX DSP48E1 based TFxP MAC unit and synthesis results, and Section VI provides the conclusions.

II. Design Space Exploration

This section provides an analysis of a Deep Neural Network (DNN) to determine the numeric type requirements. The DNN of our choice was YOLOv2; it is a well-known Convolutional Neural Network (CNN) and comprises 23 convolutional layers, among others.

Convolutional layers make heavy use of MAC operations: kernels move across the input feature map, and input values multiplied by the corresponding weight values are accumulated to form the output. Fig. 1 illustrates this operation.

In a typical CNN, 90% of the execution time goes to the convolutional layers during the inference phase [9]. I.e., the MAC unit has to be efficient; it should execute fast and not infer too much of HW to allow the maximum amount of parallelism.

978-3-9819263-5-4/DATE21/©2021 EDAA

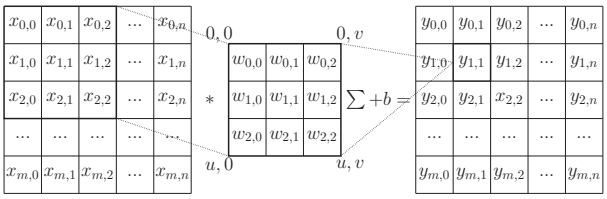

Figure 1. Convolution: m*n input features X are convolved with u*v weight matrixes to form the output Y.

To analyze the required range of the network parameters, we first performed a statical analysis and determined the minimum, maximum, and median of all the network's weight values. After this, we run the inference using a realistic photo, white image, and black image while recording all the layer outputs' extreme and median values. Table I presents the analyses results.

Table I
ANALYSIS OF THE WEIGHTS AND ACTIVATION VALUES OF YOLOV2.

	Minimum	Maximum	Median
Weights	-18.6	99.5	13.7
Photo	-113.9	106.3	0.7
All white	-57.9	31.6	1.4
All black	-23.1	28.6	1.2

As the analysis shows, most activation and weight values are low in magnitude. However, there are larger values present as well. The candidate drop-in replacement for FP should cope with the range and keep the maximum precision for median values. Additionally, the chosen data type has to use a minimum amount of bits to cope with memory bandwidth's limitations.

According to Table I, a numeric type with an 8-bit integer part can fit all the extreme values without over- or underflow. However, median values suggest that this range is not required for most of the calculations. While FP representations inherently solve this problem, fixed-point numbers have to use other means to overcome this.

A typical approach found in literature either makes use of dynamically adjusting the radix point in fixed-point numbers [10], or uses FPGA re-configuration to change the numeric format [11]. Both of these approaches have drawbacks: dynamically adjusting the radix point requires arbitrary shifters in Processing Elements (PEs), while re-configuration slows down the algorithm's execution.

This paper proposes the drop-in replacement for FP representations, which does not require re-configuration or dynamic adjustments: Triple Fixed-Point (TFxP).

III. TRIPLE FIXED-POINT

In search of a numeric format that does not sacrifice small numbers' precision to the range as much as the fixed-point representation does, the authors in [7] propose DFxP. DFxP makes use of a single exponent bit to select the radix point location. Authors in [12] use the format to replace the FP for CORDIC calculations and extend the work to use dynamic DFxP in [13].

Dynamic DFxPs undoubtedly improve the accuracy of computations as it takes a step towards FP representations. While the format is more HW friendly, the dynamic nature calls for arbitrary shifters.

Reserving one extra bit for exponent extends the DFxP to Triple Fixed-Point (TFxP). It bears the same objective as dynamic DFxP: develop the precision for a specific numeric range, but infers less FPGA resources.

Fig. 2 presents the proposed TFxP format, where a_x and b_x are the bit length of integer and fractional parts respectively. Depending on the range, (1) defines the numeric value D the representation is holding, where X is the significand, and E denotes the value of the exponent field.

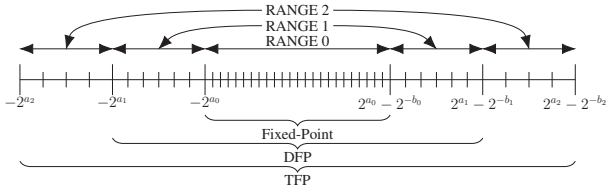

Figure 2. Triple Fixed-Point (TFxP) representation. Ranges 1 and 2 increase the range while sacrificing the precision.

$$D = \begin{cases} X \cdot 2^{-b_0} & \text{if } E = 0 \\ X \cdot 2^{-b_1} & \text{if } E = 1 \\ X \cdot 2^{-b_2} & \text{if } E = 2 \end{cases} \quad (1)$$

In order to convert FP to TFxP, a suitable target range has to be selected. Equation (2) defines the range selection: the first range capable of accomodating the value without the over- or underflow is chosen, and the exponent field E is set accordingly. The value 3 indicates over- or underflow, based on the sign bit S.

$$E = \begin{cases} 0 & \text{if } -2^{a_0} < D < 2^{a_0} - 2^{-b_0} \text{ else} \\ 1 & \text{if } -2^{a_1} < D < 2^{a_1} - 2^{-b_1} \text{ else} \\ 2 & \text{if } -2^{a_2} < D < 2^{a_2} - 2^{-b_2} \text{ else} \\ 3 & \text{overflow} \end{cases} \quad (2)$$

Table II shows the bit fields of the TFxP format 16_13_9_5, where the notion of format is $n\_b_0\_b_1\_b_2$ and n is the total number of bits the representation uses.

Table II
TRIPLE FIXED-POINT (TFXP) FORMAT 16_13_9_5.

Mode		**Signed significand**												
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	S	fraction												
0	1	S	integer					fraction							
1	0	S	integer									fraction			

A numeric format's critical property is the dynamic range: the ratio of the absolute values of the largest and smallest numbers the format can accommodate (3).

$$\text{Dynamic Range} = 20\log_{10}(2^{a_2+b_0}) \text{ (dB)} \quad (3)$$

The maximum dynamic range for the TFxP is the same as for DFxP. However, constructing such a range with DFxP loses the precision in the middle range entirely. TFxP mitigates this problem by using the middle range, and therefore, ensures a more comprehensive usable dynamic range.

IV. TFxP YOLOv2 inference precision

This section provides the results of YOLOv2 inference precision using the TFxP and DFxP and FP formats. The analysis was performed using MATLAB, while mex functions were used to add support for DFxP and TFxP.

Before the experiment, all the network's weight values were converted to the data type under test, and the AP@[.5:.95] on COCO 2014 validation dataset [14] was used for comparison. Table III presents the network accuracy for different formats, including the FP. The length of bitfields is marked using the same notation as for TFxP (Table II).

Table III
PRECISION OF THE YOLOV2 NETWORK USING FIXED-POINT (FXP), DFXP, TFXP, AND FP FORMATS.

	FxP 16_13	**DFxP 16_13_9**	**DFxP 16_13_5**	**TFxP 16_13_9_5**	**FP**
AP@[.5:.95]	0.45	0.47	0.49	0.52	0.52

The first format presented in Table III, FxP 16_13, has only three bits for the signed integer part. However, range analysis in Table I suggests that seven bits plus the sign bit are required. Therefore, FxP 16_13 can not reach the precision of FP.

The second format, DFxP 16_13_9, extends the maximum integer part to five bits while preserving the lower range accuracy. The precision increases, but as there are still overflows present, it can not reach the level of FP either.

The DFxP 16_13_5 format sets the upper range to nine bits, ensuring no overflows. The precision increases but does not reach the level of FP. Here, there is a more significant gap between upper and lower ranges than DFxP 16_13_9. TFxP addresses that issue by introducing one additional range.

As the results show, TFxP is the only format that reaches the precision of FP. Compared to DFxP 6_13_5, which can avoid overflows as well, the TFxP extends the precision of middle range values.

V. MAC unit

This section presents the TFxP MAC unit. It has been synthesized to XILINX System On Chip (SoC) device Z-7020 and makes use of HW DSP48E1 slices.

The DSP48E1 has four inputs and can natively perform various operations on them using integers, including MAC. However, using the FxP format requires additional considerations like radix point alignment.

Multiplication of two TFxP numbers produces the output O with shifted radix point, (4).

$$O = D_0 \cdot 2^{b_0} \cdot D_1 \cdot 2^{b_1} = D_0 \cdot D_1 \cdot 2^{b_0+b_1} \quad (4)$$

In the case of FxP multiplications, both operands have the same radix point location. Therefore, the result always has the same shift, and the internal accumulator can directly be used. The same does not hold for TFxP format: the radix point location of the multiplication output is the sum $b_0 + b_1$ (4). The total number of possibilities equals the *combinations with repetitions*: there are six different shifts possible.

Operands with different radix point locations cannot directly be summed; correction logic is required in the accumulator loop. This either reduces the maximum operating frequency of the MAC unit or increases the latency if a pipeline is used.

The proposed MAC unit (Fig. 3) ensures the fixed shift in multiplication output, independent of the input operands. The maximum possible shift in the multiplication output equals $2b^0$: double the shift of the lowest range. In case one of the inputs does not belong to the lowest range, an additional pre-shift has to be applied: the proposed MAC unit uses input multiplexers to achieve that. Compared to FP, the TFxP MAC does not require full-featured shifters as the total amount of possible input combinations is limited.

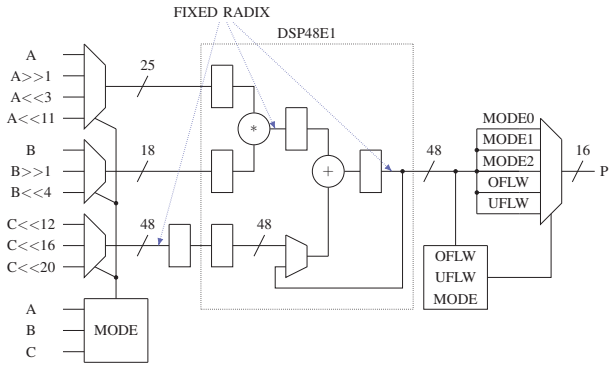

Figure 3. TFxP MAC. The design wrapes XILINX DSP48E1 HW slices. Blue arrows mark the locations where the radix point positions are matched.

The DSP48E1 slice has an additional input C, which can be added to the multiplication result instead of the internal accumulation. This input's radix point is set to the same position as the multiplication output.

The maximum pre-shift is limited to the DSP slice capabilities. Given that the width of the input A is 25 bits, the maximum possible left-shift for that input is $25 - 14 = 11$. For input B, the maximum shift is $18 - 14 = 4$, yielding $11 + 4 = 15$ bits total.

The maximum shift for the format presented in Table II is $2b^0 = 26$ bits. Similarly, the shortest fractional part in multiplication output is $2b^2 = 10$. Therefore, the total required pre-shift is $26 - 10 = 16$, one bit more than the maximum of 15. The proposed MAC unit mitigates this problem by setting the common fixed shift to 25, and performs the 1-bit right shift to one of the operands if both multiplication inputs belong to the lowest range.

Table IV presents the required pre-shifts to fix the multiplication output radix point. The least significant bit of input A selects the behavior if both of the operands are from the

lowest range: if A0 $= 0$, input A is right-shifted; otherwise, the input B is right-shifted, and one or zero bits of data is lost.

Table IV
RANGES OF THE TFXP MULTIPLICATION INPUTS AND REQUIRED PRE-SHIFTS FOR THE FIXED RADIX POINT LOCATION IN THE OUTPUT.

A Range	B Range	$\sum$ Shift	A Pre-Shift	B Pre-Shift
0	0	26	0/-1 [a]	0/-1 [b]
0	1	22	3	0
0	2	18	3	4
1	0	22	3	0
1	1	18	3	4
1	2	14	11	0
2	0	18	3	4
2	1	14	11	0
2	2	10	11	4

[a] -1 if A0 $= 0$, 0 otherwise.
[b] -1 if A0 $= 1$, 0 otherwise.

Figure 4 presents the DSP slice's output: signed fixed-point number with 25 fractional bits. The bits in the *OF GUARD* field have to match the sign bit; over or underflow has occurred otherwise. The range selection logic is similar to overflow check: if all the bits in field R_2 equal to the sign bit, the highest range is not needed. The same holds for the field R_1.

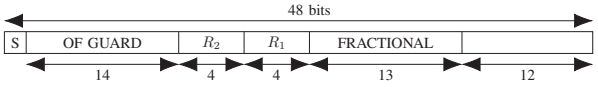

Figure 4. DSP slice output. Fields R_1 and R_2 determine the range. 14-bit *OF GUARD* is used to check the over- and underflows.

Table V presents the synthesis results of the MAC unit. Additionally, all the formats infer a single DSP slice.

The first format, 16_14_9_5, uses a maximum 14-bit fractional part: restricting the exponent field to 1 bit for the range 0 allows it. The maximum output radix point is 28 in that case, which yields to 18 bits pre-shift, 3 bits more than allowed. Additional logic to analyze and loose three least significant bits from the input if both operands belong to the lowest range infers a lot of additional HW compared to 16_13_9_5 format. Therefore, the proposed MAC unit uses the latter.

Table V
INFERRED HW OF DFXP AND TFXP FORMATS.

Format	LUTs	Regs	Slices	Power (W)	WNS (ns)	clk (MHz)
TFxP 16_14_9_5	124	23	35	0.148	0.834	393
TFxP 16_13_9_5	80	22	23	0.147	0.909	393
DFxP 16_13_5	72	18	29	0.133	0.680	393

Comparing the TFxP 16_13_9_5 and DFxP 16_13_5 formats reveals that additional middle range has mild impact on iferred HW. As an interesting observation, the synthesizer managed to combine the TFxP MAC to fewer HW slices compared to the DFxP.

Despite the positive Worst Negative Slack (WNS), the maximum operating frequency is limited by the DSP slice. All the designs can execute at 393 MHz.

VI. CONCLUSIONS

The novel contribution of this work is to provide the Triple Fixed-Point (TFxP) format for Deep Neural Networks (DNNs); it can directly replace the Floating Point (FP) format without the network re-training. The format is proposed based on analyzing the YOLOv2 network parameters and activation values during the inference phase, followed by the converted network's precision analysis. From the application point of view, direct conversion of FP to TFxP allows training the network using a Graphical Processing Unit (GPU), and deployment on Field Programmable Gate Array (FPGA), for example.

In addition to the format proposal, TFxP based Multiply-Accumulate (MAC) unit has been presented. The proposed MAC unit wraps the XILINX DSP48E1 slice to achieve the best performance and power efficiency. Compared to the FP, TFxP MAC infers much less HW resources, while the inference precision of the network is retained without the re-training.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 5 2015.
[2] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.
[3] M. Langhammer and B. Pasca, "Design and Implementation of an Embedded FPGA Floating Point DSP Block," Altera, Tech. Rep., 2014. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01089172
[4] Xilinx, "Performance and Resource Utilization for Floating-point." [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ru/floating-point.html
[5] E. Wang, J. J. Davis, R. Zhao, H. C. Ng, X. Niu, W. Luk, P. Y. Cheung, and G. A. Constantinides, "Deep neural network approximation for custom hardware: Where We've Been, Where We're going," *ACM Computing Surveys*, vol. 52, no. May, pp. 1–39, 2019.
[6] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Computing and Applications*, vol. 32, no. 4, pp. 1109–1139, 2020.
[7] C. Te Ewe, P. Y. K. Cheung, and G. A. Constantinides, "LNCS 3203 - Dual Fixed-Point: An Efficient Alternative to Floating-Point Computation," in *Field Programmable Logic and Application*. Springer Berlin Heidelberg, 2004, pp. 200–208.
[8] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
[9] M. P. Véstias, R. P. Duarte, J. T. de Sousa, and H. C. Neto, "A fast and scalable architecture to run convolutional neural networks in low density FPGAs," *Microprocessors and Microsystems*, vol. 77, 2020.
[10] C. Su, S. Zhou, L. Feng, and W. Zhang, "Towards high performance low bitwidth training for deep neural networks," *Journal of Semiconductors*, vol. 41, no. 2, 2020.
[11] G. A. Vera, M. Pattichis, and J. Lyke, "A dynamic dual fixed-point arithmetic architecture for FPGAs," *International Journal of Reconfigurable Computing*, vol. 2011, 2011.
[12] A. Jacoby and D. Llamocca, "Dual fixed-point CORDIC processor: Architecture and FPGA implementation," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2016, pp. 1–8.
[13] ——, "Dynamic dual fixed-point CORDIC implementation," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 235–240.
[14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.