NUCL SCI TECH (2020) 31:46
https://doi.org/10.1007/s41365-020-00756-z

# FPGA implementation of neural network accelerator for pulse information extraction in high energy physics

Jun-Ling Chen[1] · Peng-Cheng Ai[1] · Dong Wang[1] · Hui Wang[1] · Ni Fang[1] · De-Li Xu[1] · Qi Gong[1] · Yuan-Kang Yang[1]

Received: 6 January 2020 / Revised: 22 March 2020 / Accepted: 23 March 2020 / Published online: 27 April 2020


**Abstract** Extracting the amplitude and time information from the shaped pulse is an important step in nuclear physics experiments. For this purpose, a neural network can be an alternative in off-line data processing. For processing the data in real time and reducing the off-line data storage required in a trigger event, we designed a customized neural network accelerator on a field programmable gate array platform to implement specific layers in a convolutional neural network. The latter is then used in the front-end electronics of the detector. With fully reconfigurable hardware, a tested neural network structure was used for accurate timing of shaped pulses common in front-end electronics. This design can handle up to four channels of pulse signals at once. The peak performance of each channel is 1.665 Giga operations per second at a working frequency of 25 MHz.

**Keywords** Convolutional neural networks · Pulse shaping · Acceleration · Front-end electronics

## 1 Introduction

In modern particle physics experiments, the analysis and discrimination of physical events depend on the observation of the types and kinetic information of the final state particles . This information is usually extracted by measuring the energy, time, and position of particles in the detector [1–3]. With the development of digital nuclear spectrometer systems, digital signal processing algorithms have been applied to estimating pulse parameters [4, 5]. Mark A. Nelson et al. evaluated a curve fitting method to calculate the relative arrival times of pulses [6]. Huang et al. proposed an estimation method for parameters of overlapping nuclear pulse signals based on a population technique [7]. However, the exponential signal and the non-Gaussian noise of the detector system complicate the problem, making the traditional methods insufficient to achieve optimal accuracy and efficiency. Moreover, the long-term drift and short-term changes in the detector cause systematic variations of the analog-to-digital converters (ADC) [8].

Neural networks were applied to the field of high energy physics in the 1980s [9] because of their advantages in pattern recognition. Recently, the deep learning technology represented by convolutional neural networks has progressed rapidly [9, 21]. It has been successfully applied to high energy physics, and has been proven to perform well in pulse shape identification and particle discrimination [10, 11]. In a recent paper [8], a deep learning architecture was proposed for a time series of nuclear pulse signals and demonstrated favorable performance in practice.

Deep learning networks require a large number of calculations to realize the forward propagation. Hardware accelerators based on graphics processing units (GPU),

This work was supported by the National Natural Science Foundation of China (Nos. 11875146 and 11505074) and National Key Research and Development Program of China (No. 2016YFE0100900).

✉ Peng-Cheng Ai
pengcheng.ai@mails.ccnu.edu.cn

✉ Dong Wang
dongwang@mail.ccnu.edu.cn

[1] Central China Normal University, Wuhan 430079, China

application-specific integrated circuits (ASIC), and field programmable gate arrays (FPGA) are widely used to implement neural network calculations [12–15].

The GPU provides parallel computing and floating-point computing capabilities, which make it a good choice to train and apply to deep neural networks. However, because of their high power consumption, GPUs are not suitable for low power platforms [16]. On the other hand, the ASIC has many advantages, such as low power consumption, high performance, and small size [17]. Nevertheless, the development cycle of an ASIC is long, and the customized design also makes an ASIC less flexible in adapting it to various networks. As a programmable device, an FPGA has reconfigurable logic resources based on a look-up-table (LUT) [20]. According to the operational requirements of the network model, the hardware circuit of an FPGA can be adjusted and optimized, providing high flexibility in the hardware framework. However, an FPGA has no advantage in floating-point operations. FPGA has the ability of parallel and pipeline processing. In recent years, Microsoft, Amazon, and many other companies have deployed a huge number of FPGAs in their data centers to upgrade their computing facilities [26]. The FPGA, with its short development cycle and flexible logic resources, is also used as a preliminary step for ASICs. To preprocess the nuclear pulse signals in the front-end electronics of the detectors, we designed a customized neural network accelerator for pulse timing based on the neural network architecture described in [8].

## 2 Background

In the process of nuclear signal digitization, there are two major converters: time-to-digital converters (TDC) and ADC. Generally, the TDC is used for high precision time measurement (picosecond scale), while the ADC is used for amplitude information and lower time accuracy measurements (nanosecond scale). The previous network structure in [8] is used to process a one-dimensional time series of nuclear pulse signals from ADC-based detectors. In this section, we first introduce the shaped pulse, citing the photon spectrometer (PHOS) calorimeter [18]. We then review the previous work in [8].

### 2.1 Shaped pulse

The front-end electronics (FEE) of the PHOS calorimeter has 32 independent readout channels, each of which is connected to a detector unit composed of PWO ($PbWO_4$) crystals, avalanche photodiodes (APD), and charge sensitive pre-amplifiers (CSA). The APD detects the light induced by incident particles and the CSA converts the resulting charge into a voltage signal. The signal from a CSA is fed to a channel of the FEE card and shaped by two CR-RC2 signal shapers with high gain and low gain. At the terminations of the two shapers are two 10-bit ADCs for digitizing with a fixed sampling rate of 10 MS/s [19]. Based on the shaping circuit, the output pulse shape of the shaper in the time domain is shown in Fig 1, and the representation of the shaper can be formulated as the equation below:

$$V(t) = \begin{cases} K \cdot (\frac{t - t_0}{\tau_p})^2 \cdot e^{(-2 \cdot \frac{t-t_0}{\tau_p})} + b & \text{for } t \geq t_0 \\ b & \text{for } t < t_0 \end{cases}. \quad (1)$$

In this equation, $K$ is defined as $\frac{2Q \cdot A^2}{C_f}$, where $Q$ is the APD charge, $A$ is the shaper gain, and $C_f$ is the charging capacitance of the CSA. $t_0$ is the start time and $b$ is the pedestal. $\tau_p$, as the peaking time, is equal to twice the shaping time which is 2 μs here. $Q, A, c_f$ represent the APD charge, shaper gain, and charging capacitance of the CSA, respectively.

### 2.2 Pulse timing estimation

#### 2.2.1 Neural network

A network model, shown in Fig. 2, is based on a model proposed in [8] and verified to be feasible for a time series of nuclear pulse signals. This network consists of a denoising autoencoder (DAE) and a regression network. An autoencoder is an unsupervised neural network that attempts to approximately copy its input to output to learn useful properties of the data [21]. The DAE is an

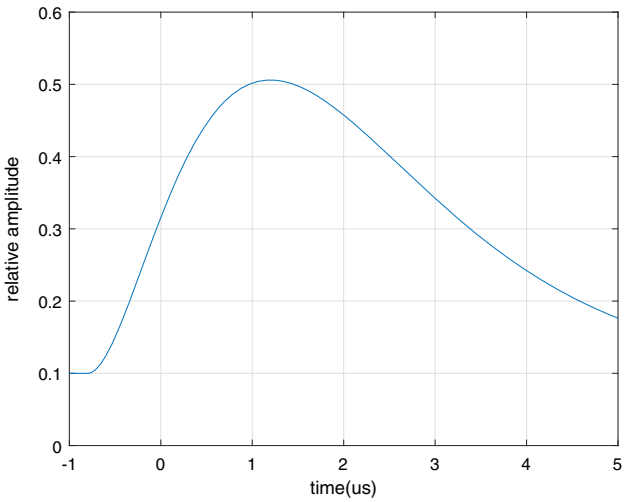



**Fig. 1** Typical diagram of the shaped pulse. The values of $K$ and $b$ are related to the detector readout electronics. In this figure, the values of $K$ and $b$ are set to 3 and 0.1, respectively. The start time $t_0$ is randomly taken as − 0.7

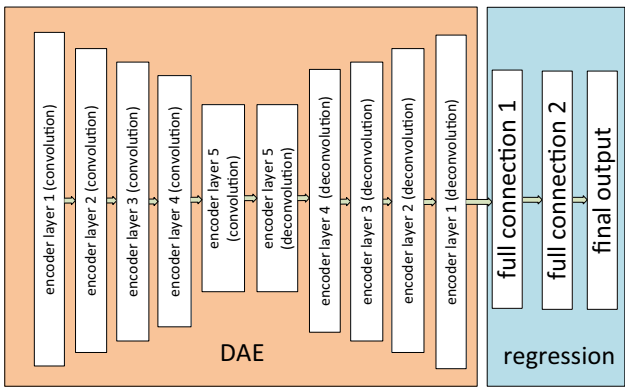



**Fig. 2** Architecture of the network

autoencoder that receives a noise-added data as input and is trained to recover the original data as its output [22]. The regression network composed of two full connection layers is used to output the parameters of interest.

To improve performance of the network, convolution and deconvolution are added to replace full connection in the encoder and decoder layers in the DAE and to enhance the process of feature extraction. In [8], the network consists of eight convolution layers and eight deconvolution layers in the DAE, whereas in this design, the number of convolution and deconvolution layers is reduced to five. Skip connections are also used between the encoder and the decoder. This means that, except for the last layer of the encoder and the first layer of the decoder, every encoder layer and the corresponding decoder layer are directly connected, which solves the problem of gradient vanishing/exploding in the network optimization. The parameters of all layers of the network are shown in Table 1.

**Table 1** Architecture of the network

| Layer | Kernel[$N_C$] | Feature map[$N_C$] | Output[$N_C$] |
|---|---|---|---|
| Conv1 | 4[16] | 32[1] | 16[16] |
| Conv2 | 4[32] | 16[16] | 8[32] |
| Conv3 | 4[64] | 8[32] | 4[64] |
| Conv4 | 4[128] | 4[64] | 2[128] |
| Conv5 | 4[128] | 2[128] | 1[128] |
| Deconv5 | 4[128] | 1[128] | 2[128] |
| Deconv4 | 4[64] | 2[128] | 4[64] |
| Deconv3 | 4[32] | 4[64] | 8[32] |
| Deconv2 | 4[16] | 8[32] | 16[16] |
| Deconv1 | 4[1] | 16[16] | 32[1] |
| Fc1 | — | 32[1] | 256[1] |
| Fc2 | — | 256[1] | 256[1] |

$N_C$ is the number of channels. Conv1 is the first layer of convolution and Deconv1 is the last layer of deconvolution. Fc1 and Fc2 are fully connected layers

#### 2.2.2 *Data processing*

The following steps are used to process the pulse data for the neural network.

1. Generate the data set. For the pulse signal in equation 1, different values of K and $t_0$ are taken to generate multiple sets of pulses. Thirty-two points with a fixed interval are sampled for each example in the data set. These examples are divided into the training data set and the test data set in a ratio of 4 to 1.
2. Train the network. This step is performed on the GPU. First, noise is added to the input data based on the noise level that is expected to appear in the pulse signal, and the DAE is pre-trained with the data. The DAE receives noisy data as the input and the original data as the label. By this means, the DAE can develop the ability to filter out noise. Then, the network is finetuned end-to-end.
3. Predict $t_0$. We use the trained model on the test data set and the values of the start time $t_0$ are predicted.

In the experiment of [8], the difference between the predicted values and the actual values was fitted as a Gaussian function. The standard deviation of Gaussian fitting is a measure of time resolution, and the average is a measure of system bias. The experimental results show that, in time resolution, the performance of the neural network method is 27.3 percent higher than the curve fitting method. In addition, the system bias is greatly reduced by the neural network. This means that it is feasible to use this network to estimate the start time of a nuclear pulse, and this method can greatly improve the time accuracy.

## 3 Firmware design and implementation

To preprocess the pulse data in the FEE of the detector in real time and reduce the off-line data storage, we designed a customized neural network accelerator based on an FPGA for the network model in Sect. 2. In this section, we introduce the overall architecture and the implementation of each module of the accelerator.

### 3.1 Firmware

This design is composed of an external central processing unit (CPU) and an FPGA. The external CPU is used to generate instructions, while the FPGA is responsible for the layer calculations. To connect with a RISC-V (an instruction set architecture) CPU in a future design, we adopt the internal chip bus (ICB) [23] to communicate with the outside. The convolution, deconvolution, and full

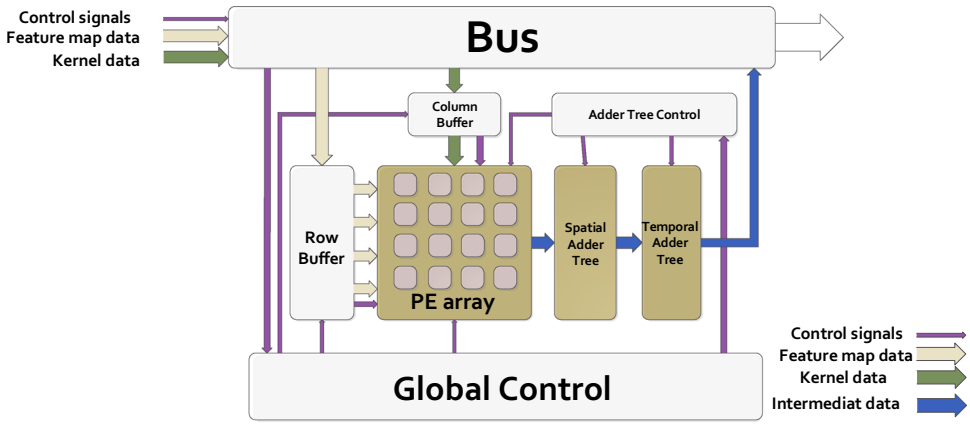



**Fig. 3** (Color online) Architecture of the accelerator

connection operations are configured by the instructions of the external CPU.

For one layer of the network, the data flow of the FPGA is shown in Fig. 3:

- After the accelerator is started, the external CPU sends instructions through the ICB. There are four devices loaded on the ICB, global controller with the process element (PE) array, adder tree (AT), row buffer (RB), and column buffer (CB). The ICB selects appropriate devices for reading and writing operations according to the addresses.
- Through the ICB bus, the feature map data and convolution kernel data are each sent to the RB and CB. The RB and CB adjust the order of the feature map data and kernel data separately, and add different zero-padding depending on the calculation types. When there are more than 16 channels in the feature map, the channels will be spliced in order.
- The preprocessed feature map data and kernel data are sent to a $4 \times 4$ PE array for multiply-and-accumulate (MAC) calculations.
- The calculations from the PE array are fed into the AT for the following operations. The AT performs 0 to 8 levels of adder operations on the data and decide whether to use the activation function depending on the configurations.
- The calculations from the AT are transmitted through the ICB.
- Finally, the operations of a single turn in a network layer are complete. Since each PE can store up to 16 kernels, when the stored kernels are used up, the feature map data will be unchanged in the PEs, while the kernel data will be input and the operations in steps 3-5 will be repeated until all kernels are calculated.

### 3.2 Data preprocessing in the accelerator

Two buffers, RB and CB, are placed between the ICB and the arithmetic module to preprocess the feature map data and the kernel data to simplify external manipulations.

Owing to the 32-bit bus width of the feature map in the PE array, the feature map data in the RB should be 4 bytes, aligned through the data buffer to connect with the PE array lines. Owing to the different calculations between convolution, deconvolution, and the fully connected operation, it is necessary to add zero-padding for the feature map data. The convolution and deconvolution operations are shown in Figs. 4 and 5. For the convolution layers (left), zeros are added at the ends of the feature map data. For the deconvolution layers (right), zeros are added at both ends of the data and interleaved between the data. For the fully connected layers, no zero-padding is added. The CB caches 8-bit kernel data and then outputs $16 \times 8$-bit data to match the 16 PEs. In addition, RB and CB temporarily store input data while the PEs are working, which can improve the parallel efficiency of the accelerator.

### 3.3 Details of RB and CB

The RB is not only used to cache and align the feature map data, but also used to add zero-padding to the input data, which is used to simplify the external data operation and reduce the operations in the PE unit. First, four 8-bit

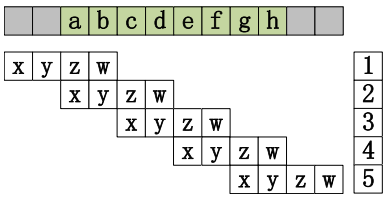



**Fig. 4** Diagram of convolution calculation. The letters a-h represent the feature map data and the shadow cubes stand for zero-padding. The letters x, y, z, and w constitute the kernel data and the numbers 1 to 5 represent the result of the operation

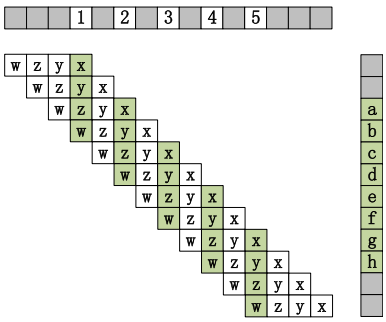



**Fig. 5** Diagram of deconvolution calculation. This figure shows the zero-padding method and the calculation of the deconvolution

RAMs, corresponding to four rows of the PE array, are used to cache data. The feature map data loads into the RAMs row by row. The feature map data is then zero-padded under the control of a state machine.

The zero-padded data is transmitted as four rows individually to four deserializers. After that, four rows and four columns of data are sent to the PE array.

The CB is used to cache and align the kernel data. Similar to the front of RB, sixteen 8-bit RAMs, corresponding to 16 PEs, are used to cache the kernel data.

### 3.4 Calculation module

The calculation module is composed of the PE array and the AT. The PE array is a $4 \times 4$ array consisted of 16 PEs, which is mainly responsible for MAC calculations. The AT includes spatial AT and temporal AT. The spatial AT is used to accumulate the calculations of different PEs at the same time, and the temporal AT accumulates the results of the spatial AT at different times.

As shown in Fig. 6, each PE consists of a controller, an Arithmetic Logic Unit (ALU) and memory. The memory stores the feature map data and kernel data and punches them into the ALU. In the ALU, 4 bytes of feature map

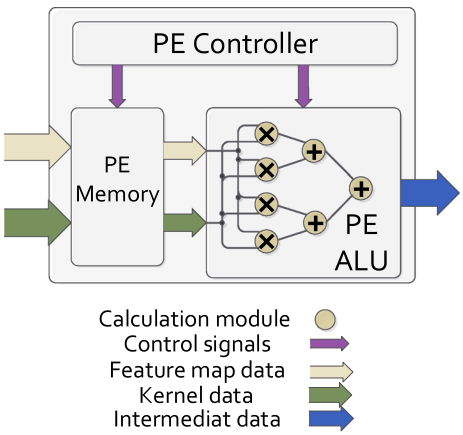



**Fig. 6** (Color online) Structure of a PE

data and 4 bytes of kernel data read from the memory are multiplied by element, and then, the calculations are added and fed to the AT as the result of one PE manipulation.

The accelerator is equipped with 16 PEs, and so is able to calculate 16 channels of input data at a time. When there are fewer than 16 channels of input data, the output data from the PEs is stored by the spatial AT and then sent back through the ICB. Otherwise, the calculations of the PEs are stored by the spatial AT and then sent to the cache of the temporal AT. The data cached in the temporal AT is stored with other data until data from all channels are calculated.

### 3.5 Details of PE and AT

A PE ALU consists of three carry ahead adders and four multipliers designed as a four-stage pipeline. A PE calculates the multiplications and additions of four feature map values and four kernel values in six clock periods. Both the spatial adder tree and the temporal adder tree are four-stage adder trees composed of 15 carry ahead adders, which can be up to eight-stage in total.

Owing to the large number of kernels, a PE is equipped with two areas of RAM to store all the feature map data and part of the kernel data, respectively. After the stored kernel data is calculated, the next batch of kernel data is transmitted into the PE, while the feature map data is kept in the RAM. When there are greater than 16 feature map data channels, the feature map data calculated in a PE needs to be concatenated between channels. For example, the first channel of the feature map data is concatenated with the 17th channel of data. The concatenation is completed in the RB. When the convolution window moves to the connection of two channels of data, the calculation of the ALU is invalid. The PE controller recognizes the ALU result as valid or invalid by counting the times of the PE calculation.

The 8-stage AT is used to accumulate the convolution results of all channels of a feature map. Firstly, 16 FIFOs are designed in the spatial adder tree to store the results of 16 PEs. The spatial adder tree takes one value from each of the 16 FIFOs and adds them. Then, the results from the spatial adder tree are stored in RAM in the temporal adder tree. The data stored in the RAM is read in a particular order to be added up together by channels. An example is shown in Figs. 7 and 8.

### 3.6 Parallelization and data reuse

Because of the large number of calculations in convolution operations, implementing parallel calculations will significantly improve the efficiency of the accelerator. Moreover, in convolution and deconvolution, the feature map data corresponds to multiple sets of kernel data, and

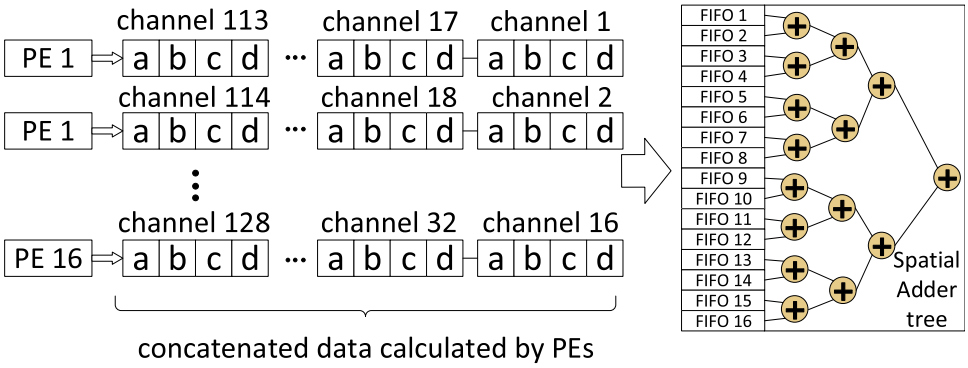



**Fig. 7** Diagram of the data calculated in the spatial adder tree. This figure shows the concatenation method of the data after PEs and the structure of the spatial adder tree

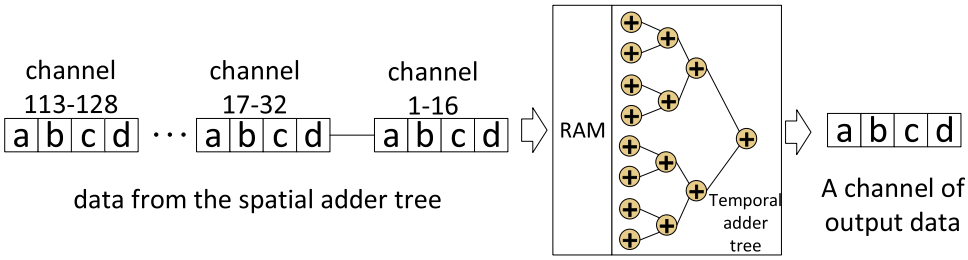



**Fig. 8** Diagram of the data calculated in the temporal adder tree. This figure shows the structure of the temporal adder tree

one set of kernel data corresponds to several convolution subregions. All of these computations are independent, and thus, data reuse can effectively reduce the data movement and improve efficiency. This design adopts the following methods to improve the operating efficiency.

**Parallel calculation in the channels:** The feature map data from multiple channels can be calculated in parallel. The PE array can calculate up to 16 channels of data at a time, while the AT adds the multi-channel data in a pipeline to realize multi-channel internal parallel calculation.

**Parallel calculation in the kernels:** The length of the kernel in the network is 4. To accommodate the kernel length, the PE is designed to perform calculations for 4 units of 8-bit feature map data and 4 units of 8-bit kernel data at a time. Therefore, the kernel data corresponding to the same subregion of convolution can be calculated in parallel.

**Kernel data reuse:** In one round of convolution, after being taken out, the kernel data is stationary while the feature map is moved until all calculations for this kernel data have been performed. After that, another set of kernel data is fed into the PE and calculations continue with the previous round.

**Feature map data reuse:** In the first layer of the encoder, there is one input channel. At this time, 16 PEs can share the same set of feature map data and implement the spatial reuse of feature map data.

## 4 Experiment

To verify the function of the design and evaluate its performance, we analyzed three aspects of the design. Firstly, we measured the FPGA resources occupied by the accelerator; secondly, we evaluated the computing performance of the accelerator when it ran the network; finally, we assessed the impact of different quantization precision on the time resolution of the network estimating pulse time. This design was implemented on the KC705 board, equipped with a Xilinx FPGA chip XC7K325T.

**Table 2** Resource usage of this design

| Resources | LUTs | LUTRAM | FF | IO |
|---|---|---|---|---|
| Used | 47927 | 319 | 56170 | 86 |
| Available | 203800 | 64000 | 407600 | 500 |
| Utilization (%) | 23.50 | 0.50 | 13.78 | 17.20 |

Look-up-table (LUT), LUTs used for RAM (LUTRAM), flip flop (FF), and input or output (IO) statistics are reported after placement and routing is completed with the Vivado set tool

### 4.1 Resource usage

We synthesized the RTL code on the Vivado platform, and obtained the resource utilization shown in table 2 as provided by Vivado. Compared with general neural network accelerators [14, 24], this design uses less resources. The utilization shows that this board can provide four independent channels for pulse signals.

### 4.2 Calculation performance

To verify the function of the RTL code logic and evaluate the performance of accelerator, we performed behavioral simulation of the RTL code on Vivado at a working frequency of 25 MHz, and measured the time required for each layer of the network to calculate a set of data. Based on the time and the number of operations at each layer, which can be easily calculated based on the size of the network model, we obtained the performance of this accelerator. As shown in Table 3, the peak performance is 1.665 Giga operations per second (GOPS).

**Table 3** Performance of the accelerator

| Layer | Running time ($\mu$s) | Performance (GOPS) |
|---|---|---|
| Conv1 | 27.3 | 0.075 |
| Conv2 | 34.6 | 0.947 |
| Conv3 | 107.5 | 0.610 |
| Conv4 | 220.2 | 0.595 |
| Conv5 | 256.0 | 0.512 |
| Deconv5 | 297.0 | 0.883 |
| Deconv4 | 366.1 | 1.432 |
| Deconv3 | 162.6 | 1.612 |
| Deconv2 | 78.7 | 1.665 |
| Deconv1 | 5.2 | 1.575 |
| Fc1 | 204.8 | 0.081 |
| Fc2 | 337.9 | 0.389 |

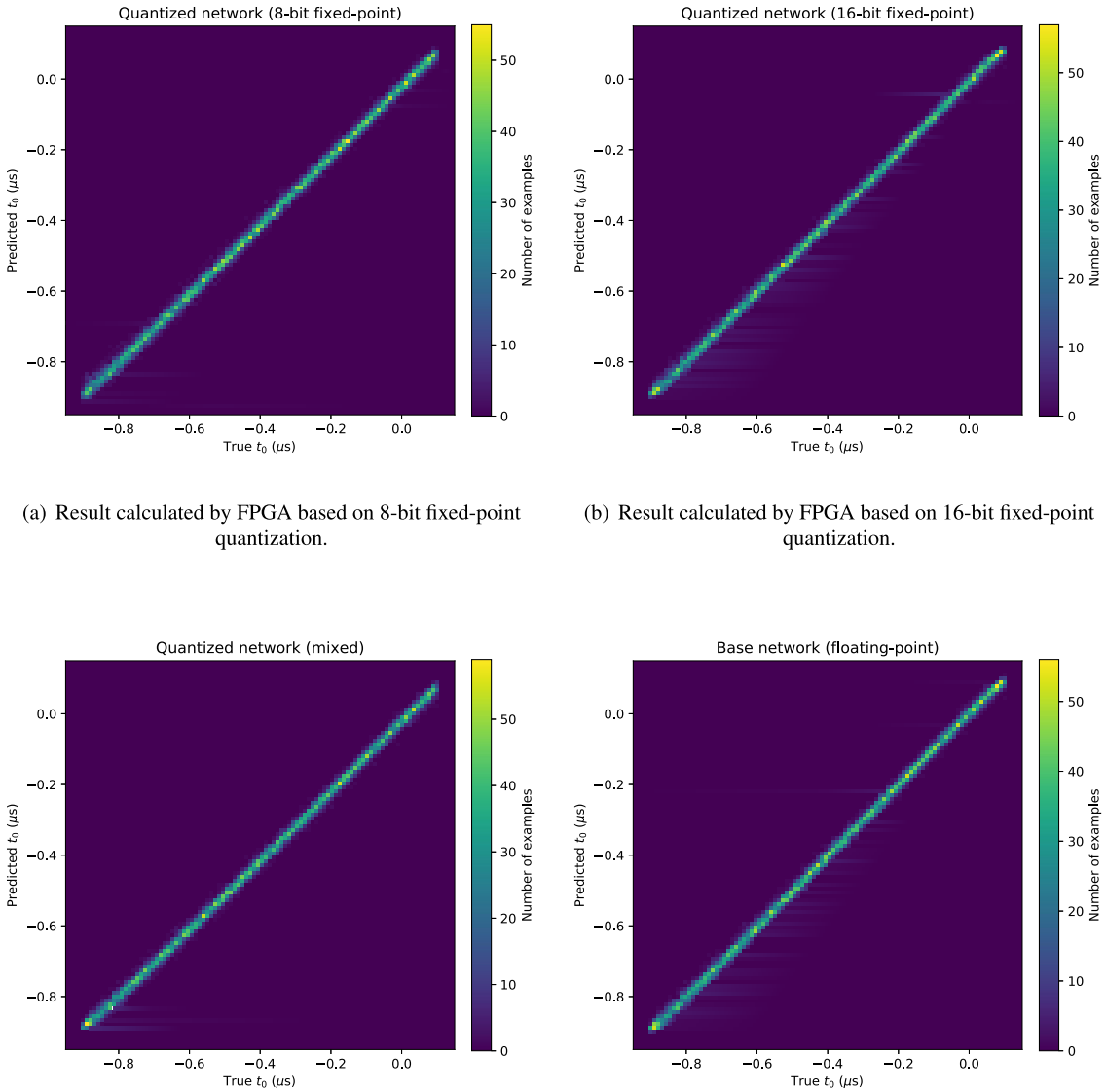
The performance shows the operation speed of the accelerator, which indicates the number of operations performed by the accelerator per second [25]. The number of operations at each layer of the network can be easily calculated based on the size of the network model

### 4.3 Pulse timing estimation

We used this design to estimate the pulse time and compare the time resolution with a GPU. In the case of the same network model and training method, the difference between using an FPGA and a GPU for pulse time estimation is mainly due to different quantization conditions. The FPGA is not suitable for floating-point operation, which is what the GPU excels at. We therefore measured the difference in the time resolution between a fixed-point quantization network on the FPGA and a floating-point quantization network on the GPU when estimating pulse time. Based on the pulse model discussed in Sect. 2, we added Gaussian noise to the pulse and sampled it to obtain 40000 samples in the training data set and 10000 samples in the test data set. The DAE was trained according to the different quantization precisions of 8-bit fixed-point, 8-bit and 16-bit mixed fixed-point, 16-bit fixed-point, and 32-bit floating-point, respectively. We then calculated the root mean square error (RMSE) between the DAE output and the true value, and gathered the statistics of the RMSE distribution. The RMSE indicates the recovery of the true pulse sequence by the DAE network. The lower the RMSE, the stronger the DAE network's ability to find the true noiseless signal from the signal with noise added. Lastly, the regression network was trained according to the different quantization precisions. According to the regression output and the true value, we mapped the distribution of the result and measured the time resolution. The result is shown in Fig. 9 and Table 4

Compared with 8-bit fixed-point DAE, the 8-bit encoder and 16-bit decoder perform slightly better in recovering the true pulse signal, while the 16-bit fixed-point DAE shows a significant improvement. For the regression task, the time resolution of the 16-bit network is 13% better than that of the 8-bit network, whereas that of the mixed network is 1.1% worse than that of the 8-bit network. The time resolution of the 32-bit floating-point network is 1.9% better than that of the 16-bit network.

The result shows that the improvement of quantization precision can improve time resolution, but mixed quantization has no advantage. The time resolution measured by the FPGA is indeed worse than that of the GPU owing to the quantization precision. This difference, however, is not as evident as the difference between 8-bit fixed-point quantization and 16-bit fixed-point quantization.




(a) Result calculated by FPGA based on 8-bit fixed-point quantization.

(b) Result calculated by FPGA based on 16-bit fixed-point quantization.

(c) Result calculated by FPGA based on 8-bit and 16-bit mixed fixed-point quantization.

(d) Result calculated by GPU based on 32-bit floating-point quantization.

**Fig. 9** (Color online) The results of the network for different quantization. The 8-bit and 16-bit mixed fixed-point quantization indicates that 8-bit fixed-point quantization is applied to the encoder of the DAE and 16-bit is used on the decoder. We plotted the results of the test samples and showed the true $t_0$ and the predicted $t_0$. The color of each square in the two-dimensional histogram shows the number of examples (with true $t_0$ and predicted $t_0$) that fall into that region

## 5 Conclusion

Neural networks have excellent learning abilities when processing time series of nuclear pulse signals. The DAE, owing to its structure, has some advantages compared with the traditional method in terms of its anti-noise properties. Currently, there are many studies on the application of neural networks to pulse parameter estimation and particle identification, but few of them are applied to FEE for real-time data processing. For a verified network model, we designed a customized accelerator applied to pulse time recovery. The customized design means that the accelerator consumes less resources while still ensuring the performance, so as to provide support for multiple data channels. This design provides a possibility to preprocess the data in the FEE of detectors in real time, reduce the amount of data storage and improve the precision of pulse time extraction. It is also a preview step for ASIC design. The study comparing our design with a GPU showed the influence of quantization on the final result. Higher quantization precision resulted in more accurate data representation and less error propagation during the neural network

**Table 4** RMSE of the DAE output and the time resolution of the result measured by FPGA and GPU in different quantization types

| Quantization type | RMSE of DAE output | Time resolution ($\mu$s) |
|---|---|---|
| 8-bit fixed-point | $0.00863 \pm 0.00281$ | 0.01437 |
| 8-bit 16-bit mixed | $0.00846 \pm 0.00280$ | 0.01453 |
| 16-bit fixed-point | $0.00473 \pm 0.00190$ | 0.01243 |
| Floating-point | $0.00412 \pm 0.00172$ | 0.01219 |

The differences between the predicted output of the network and the true values were fitted as a Gaussian function. The standard deviation of the fitted Gaussian function measures time resolution

training. However, higher quantization precision also reduced computing efficiency. Thus, the balance between quantization precision and computing efficiency is worth considering for future research. Methods for reducing the impact of low-precision quantization on networks trained with floating-point weights and feature maps are also worth studying.

## References


1. R. Grzywacz, Applications of digital pulse processing in nuclear spectroscopy. Nucl. Instrum. Meth. B **204**, 649–659 (2003). https://doi.org/10.1016/S0168-583X(02)02146-8
2. ALICE Collaboration, Performance of the ALICE experiment at the CERN LHC, Int. J. Mod. Phys. A **29**: 1430044 (2014). https://doi.org/10.1142/S0217751X14300440
3. ATLAS collaboration, Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC, Phys. Lett. **B716**:1–29 (2012) https://doi.org/10.1016/j.physletb.2012.08.020
4. N. Petrick, A.O. Hero, N.H. Clinthorne et al., A fast least-squares arrival time estimator for scintillation pulses. IEEE Trans. Nucl. Sci. **41**(4), 758–761 (1994). https://doi.org/10.1109/23.322802
5. G. Ripamonti, A. Geraci, Towards real-time digital pulse processing based on least-mean-squares algorithms. Nucl. Instrum. Meth. A **400**(2–3), 447–455 (1997). https://doi.org/10.1016/S0168-9002(97)01012-7
6. M.A. Nelson, B.D. Rooney, D.R. Dinwiddie et al., Analysis of digital timing methods with BaF2 scintillators. Nucl. Instrum. Meth. A **505**(1–2), 324–327 (2003). https://doi.org/10.1016/S0168-9002(03)01078-7
7. H.Q. Huang, X.F. Yang, W.C. Ding et al., Estimation method for parameters of overlapping nuclear pulse signal. Nucl. Sci. Tech. **28**, 12 (2017). https://doi.org/10.1007/s41365-016-0161-z
8. P.C. Ai, D. Wang, G.M. Huang et al., Timing and characterization of shaped pulses with MHz ADCs in a detector system: a comparative study and deep learning approach. J. Instrum. **14**, E03001 (2019). https://doi.org/10.1088/1748-0221/14/03/P03002
9. B. Denby, Neural networks and cellular automata in experimental high energy physics. Comput. Phys. Commun. **49**(3), 429–448 (1998). https://doi.org/10.1016/0010-4655(88)90004-5
10. J. Griffiths, S. Kleinegesse, D. Saunders, et al. Pulse shape discrimination and exploration of scintillation signals using convolutional neural networks. 2018. arXiv:1611.03180
11. MicroBooNE collaboration, Deep neural network for pixel-level electromagnetic particle identification in the MicroBooNE liquid argon time projection chamber. arXiv:1808.07269v1
12. T. Roska, G. Bártfai, P. Szolgay et al., A digital multiprocessor hardware accelerator board for cellular neural networks: CNN-HAC. Int. J. Circuit Theory Appl. **20**(5), 589–599 (1992). https://doi.org/10.1002/cta.4490200512
13. Z. Du, R. Fasthuber, T. Chen, et al. ShiDianNao: shifting vision processing closer to the sensor. ISCA '15: Proceedings of the 42nd Annual International Symposium on Computer Architecture, vol 43, 3, pp 92–104 (2015). https://doi.org/10.1145/2749469.2750389
14. J. Qiu, J. Wang, S. Yao, et al. Going deeper with embedded FPGA platform for convolutional neural network. Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, (2016). https://doi.org/10.1145/2847263.2847265
15. P. Judd, A. Delmas, S. Sharify, et al. Cnvlutin2: Ineffectual-Activation-and-Weight-Free Deep Neural Network Computing. (2017). arXiv:1705.00125
16. A. Coates, P. Baumstarck, Q. Le, et al. Scalable learning for object detection with GPU hardware. 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 10: 4287–4293 (2009). https://doi.org/10.1109/IROS.2009.5354084
17. A. Boutros, S. Yazdanshenas, V. Betz, You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference. ACM Trans. Reconfig. Technol. Syst. **11**(3), 1–23 (2018). https://doi.org/10.1145/3242898
18. H. Torii, The ALICE PHOS calorimeter. J. Phys: Conf. Ser. **160**, 012045 (2009). https://doi.org/10.1088/1742-6596/160/1/012045
19. H. Muller, R. Pimenta, Z. Yin et al., Configurable electronics with low noise and 14-bit dynamic range for photodiode-based photon detectors. Nucl. Instrum. Meth. A **565**(2), 768–783 (2006). https://doi.org/10.1016/j.nima.2006.05.246
20. C. Farabet, C. Poulet, J.Y. Han, et al. CNP: An FPGA-based processor for Convolutional Networks. 2009 International Conference on Field Programmable Logic and Applications. https://doi.org/10.1109/FPL.2009.5272559
21. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**, 436–444 (2015). https://doi.org/10.1038/nature14539
22. P. Vincent, H. Larochelle, I. Lajoie et al., Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**(12), 3371–3408 (2010)
23. T.C. Deng, Z.B. Hu, An ultra-low-power processor pipeline-structure. Appl. Electron. Tech. **45**(6), 50–53 (2019). (in Chinese)
24. C. Zhang, P. Li, G.Y. Sun, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA2015). 161-170 (2015). https://doi.org/10.1145/2684746.2689060
25. P. Molchanov, S. Tyree, T. Karras, et al. Pruning convolutional neural networks for resource efficient inference. 2016. arXiv:1611.06440
26. A. Putnam. Large-scale reconfigurable computing in a microsoft datacenter. 2014 IEEE Hot Chips 26 Symposium (HCS). https://doi.org/10.1109/HOTCHIPS.2014.7478819