# A Fast and Low Power Hardware Accelerator for ANN Working at Near Threshold Voltage

Tianbao Chen[1, a], Shouyi Yin[2,b] and Shaojun Wei[3,c]

[123]Room 3-330, Building FIT, Tsinghua University, Beijing, 100084, China

[a]ctb13@mails.tsinghua.edu.cn, [b]yinsy@tsinghua.edu.cn, [c]wsj@tsinghua.edu.cn

**Keywords:** ANN; Hardware accelerator; NTV

**Abstract.** Artificial neural network (ANN) are widely applied in machine learning and artificial intelligence. But ANN usually requires large data throughputs and induces high power consumptions. This paper proposes an accelerator design guideline for ANN with full consideration of the hardware scale, performance and power consumption. We apply multiple clocks in our design to get a high data throughput and introduce the near threshold voltage (NTV) to get a lower power consumption. We further optimize the multiplication operation in critical path obtaining a higher performance.

## Introduction

Artificial neural network (ANN) [1], an adaptive nonlinear dynamic system composed of a large number of neurons through a very rich and perfect connection, is a simulation of biological neural networks in the structure, realization and function. ANN can do so well in pattern recognition that it is widely used in word, voice and face recognition. Works in [2] proposed a handwriting recognition system based on ANN. Work in [3] mapped ANN directly to an accelerator inducing a low energy and hardware efficiency. Work in [4] used neuron processing units to implement network with large scale, unfortunately they do not pointed out how to determine the hardware scale given an arbitrary ANN. They introduce the page-mirror memory, but for each neuron too much memory access is required.

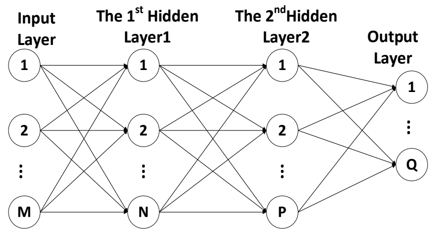

Fig. 1. A typical 4-layer network with 2 hidden layer

In this paper, we propose a hardware design guideline to map ANN with arbitrary scales and depths to accelerators. We optimize the computing way of ANN on a hardware accelerator to avoid too much SRAM access. Novel interface architectures working at multiple clocks for data and weights transmission are proposed to achieve a high throughout. We introduce the NTV [5] technology and optimize the multiplication in our design to obtain a lower power consumption.

## ANN hardware determination

ANN works in the way that each neuron in the active layer gets all the outputs of neurons in the former layer and provides one result. A typical 4-layer network is shown in Fig. 1. We define $x_n$ as the $n^{th}$ data in the former layer and $w_n$ as the $n^{th}$ weight. The operations on the $k^{th}$ neuron in layer $i$ can be represented by Eq. 1.

$$y_k = f(\sum_n^{N_{i-1}} w_n x_n - \theta) \qquad (1)$$

Here $N_{i-1}$ is the number of neurons in the former layer and $\theta$ is the threshold value. $f$ is a sigmoid function which works as an activation function. From Eq.1 and Fig. 1, we can see the computing time of one certain layer depends on two factors: the number of neurons in the former adjacent layer and the number of neurons in the current layer. For one layer, if there is only one neuron to be mapped to a single PE, then, the calculation time for the neuron will be $N_{i-1} * t$ and that for the current whole layer will be $N_i * N_{i-1} * t$. Here $t$ is the time to do a multiply-add operation. $N_{i-1}$ and $N_i$ equals to the number of neurons of the adjacent former layer and the current active layer. In such an extreme case, the accelerator will be resources-saving but extremely time-cost. At the other extreme, $N_i$ PEs are applied to the calculation operations of the active layer, it will take $N_{i-1} * t$ for the current layer to work out all the results. There is no doubt that this is the fastest way to accomplish the calculations of every layer but not available because the network scale may be huge.

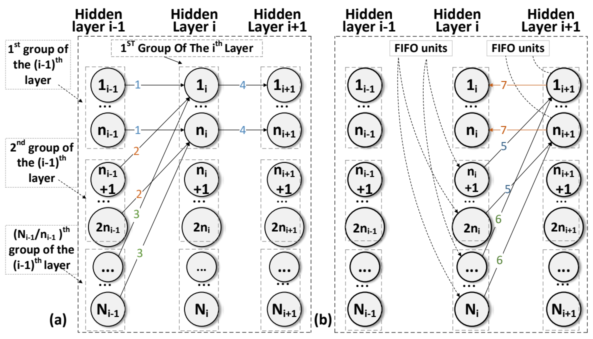
Hidden layer i-1
Hidden Layer i
Hidden Layer i+1
1st group of the (i-1)th layer
2nd group of the (i-1)th layer
(Ni-1/ni-1)th group of the (i-1)th layer
1ST Group Of The ith Layer
FIFO units
FIFO units
(a)
(b)

Fig. 2. Data flow between adjacent layers

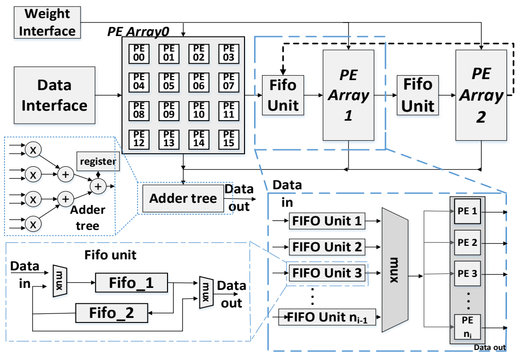

Fig. 3. The system architecture

We divided the neurons in the same layer into a few groups and the ones within a group will be processed at the same time. Then groups are processed one by one, as shown in Fig. 2(a). Neurons not only in the same layer but also in the adjacent layers can be processed in parallel, as shown in Fig. 2. As long as the results of the *1st* group of layer *i-1* are passed to the next layer, both neurons of the *1st* group of layer *i* and the *2nd* group in layer *i-1* will be processed. Outputs from the *1st* group of layer *i-1* will be stored in FIFO units when outputs of the other neurons of layer *i-1* available. Data in FIFO unit will be read out when neurons in other groups are activated. Meanwhile outputs of the *1st* group of the layer *i* are passed to the next layer and the 1st group of the layer *i+1* are mapped to the accelerator in parallel with the *2nd* group of layer *i*, as shown in Fig. 2(b).

We define *n* as the number of neurons in each group and neurons in the same layer will be divided into N/n groups. Neurons in the same group will give the results after $N_{i-1} * t$ and the time required for a layer is $N_{i-1} * N_i / n_i * t$. Where $N_i$ equals to the number of neurons in layer *i* and $n_i$ is defined as the number of neurons in one group. The larger $n_i$ is, the less time will be taken in layer *i*. If $N_{i-1} * N_i / n_i * t$ is greater than $N_{i-2} * N_{i-1} / n_{i-1} * t$, $n_i$ PEs will be left unused. In the other side, $N_{i-1} * N_{i-1} / n_{i-1} * t$ is small, that means too many PEs are heaped up here, left unused after we get the outputs. So we should achieve a balance between the calculation times of adjacent layers, as shown in Eq. 2. $n_{it}$ is the temporary number of neurons in one group in layer *i* before we set it.

$$N_{in} * N_{0t} / n_{0t} * t = N_0 * N_1 / n_{1t} * t = ... = N_{i-1} * N_i / n_{it} * t \qquad (2)$$

Thus determining the number of PEs for the *1st* hidden layer we can determine the number of PEs for each latter layer. Through the formula above we can get $n_1$, $n_2$, $n_3$..., the number of PEs of the *ith* layer is $n_i = \min(N_i, n_{it})$.

## Hardware Implementation

The architecture of our design has been shown in Fig. 3. There are 3 clocks in the system named clk-4, clk-2 and clk-1 with the relationship of $f_{clk-4} = S/2 f_{clk-2} = 4 f_{clk-1}$. *S* is the number of SRAMs that work at clk-2. Here clk-4 is applied to the data and weight interface module specially to achieve a high throughput. The other parts work at clk-1.

**PE and PE Array**

PE Arrays fulfil the multiply-add operations. We set 3 PEAs at most and 1 at least. For a network with 3 or less hidden layers, the number of the PE Arrays is the same as that of the hidden layers. For the ones with more than 3 hidden layers, we map each layer of the networks in the 3 PEAs.

Neurons in the *2nd* and *3rd* hidden layers are mapped to PE arrya1 and PE array2 respectively. As long as outputs from the *3rd* hidden layer are available, all operations on the *2nd* layer are done. So we map neurons in the *4th* hidden layer to PE array1, i.e. the *7th* step shown in Fig. 2(b). Same as the way of the *4th* hidden layer, we can have odd layers and even layers mapped to PE Array2 and PE Array3 respectively. Thus networks with arbitrary depths can be mapped to our design. To accommodate all the hidden layers, the number of PEs in PE array1 is $max(n_2, n_4, n_6)$ and that of PE array2 is $max(n_3, n_5, n_7)$. In our design PE array1 and PE array2 may not appear in some cases though shown in the picture.

**Interface architectures**

Data from off chip will be reused by the neurons in different groups. We store data in SRAMs in data interface module. *S* SRAMs with smaller capacity working at clk-2 with lower power consumption are combined as one SRAM working at clk-4. We take $S = 4$ as an example presented in Fig. 4(a). Data in the *S* SRAMs flow in parallel in clk-2 and then exported serially in clk-4. By changing the number of SRAMs we can change the relationship between clk-4 and clk-2, as $s = 2 * clk_{-4} / clk_{-2}$. Weights from off chip are read into the system at clk-4, shown in Fig. 4(b). The data flow is divided into four registers marked REG A. Data in REG A are refreshed every four clk-4 cycles and read out to REG B at clk-1, then we use four FIFOs synchronized by clk-1 to cache the data. In weight interface module, the logic elements working at clk-1 work at a near threshold voltage and those working at clk-4 are supplied with a standard voltage range (SVR).

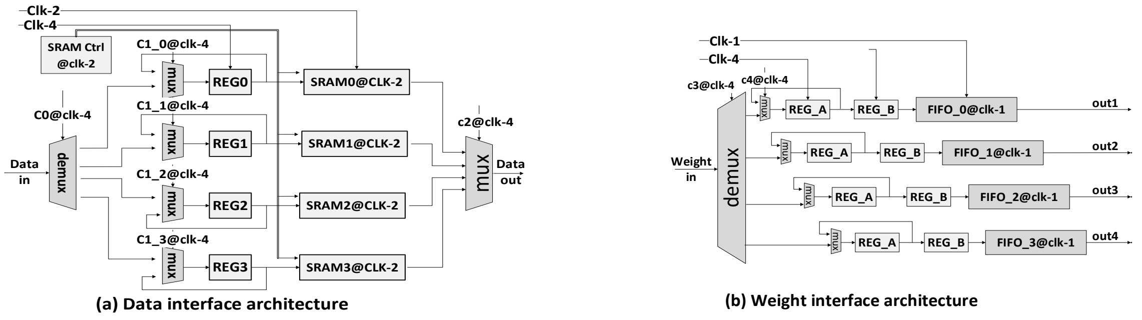

Fig. 4. Interface architectures

**FIFO Unit and adder tree**

Data will be reused by neurons in different groups. To avoid much SRAM access, all the intermediate results are stored in FIFO units composing of two FIFOs. Data from the former layer are stored in FIFO1 first. When being read out, all the data are written to FIFO2 at synchronously. In the same way, when data in FIFO2 are pulled out, they are written to FIFO1. Data flow between the two FIFOs forth and back until all the neurons in the next layer get the results. Neurons of the output layer are mapped to the adder tree shown in Fig. 3 in pipeline. There are $N_{i-1}$ data being divided to $n_i$ groups. They are weighted and accumulated group by group. Partial sum of some groups is stored in a register which is refreshed every cycle until the result has been got.

**Optimization in NTV range**

NTV is a potential technology when power consumption and heat density have been a bottleneck in IC design. Systems working in the near threshold voltage of opening transistors, about 0.6v, can get the highest power utilization efficiency but at an expense of low working frequency [6]. Here we optimize the multipliers in the critical path to get a high working frequency.

We replace a multiplier with *4* smaller-bit multipliers working in parallel. For a multiplier consisting of full adders, half adders and *"and gates"*, the delay time of M[*m-1:0*]*N[*m-1:0*] will be $[(m-1)+(n-2)]_{tcarry}+(n-1)t_{sum}+t_{and}$. Here *M*[*m-1:0*] is an m-bit number and *N*[*n-1:0*] is an n-bit one. Here $t_{carry}$ is the delay time of a full adder. $t_{sum}$ is defined as the delay time between the adjacent adders. $t_{and}$ equals the delay time of an *"and gate"*. We take *M*[*m-1:0*]**N*[*n-1:0*] as an example replacing *M*[*m-1:0*] with $\{M_h, M_l\}$ and *N*[*n-1:0*] with $\{N_h, N_l\}$. $M_h$ and $M_l$ are m/2-bit-wide and $N_h$ and $N_l$ are n/2-bit-wide. The conversion process is shown in Eq. 4. So decreasing the bit width of the multiplicators can decrease the delay time efficaciously.

$$M * N = \{M_h, M_l\} * \{N_h, N_l\} = \left\{(M_h * N_h), \overbrace{0,\dots,0}^{nzeros}\right\} + \left\{(M_h * N_l + M_l * N_h), \overbrace{0,\dots,0}^{n/2zeros}\right\} + M_l * N_l \tag{4}$$

## Experimental result

### Benchmarks and our design architectures

We synthesize our final design using the Synopsys Design Compiler (DC) and simulate our design VCS. The power consumption is estimated by PrimeTime PX.We train 7 networks called fft, inversek2j, jmeint, jpeg, kmeans, sobel and nettalk as our benchmarks. These networks are used in various situations with varied scales and depths shown in Table 1.

The benchmarks are expressed in the format of $x_{in} \times x_1 \times ... \times x_i \times x_{out}$. $x_{in}$ and $x_{out}$ are the numbers of neurons in the input layer and output layer. $x_i$ is the number of neurons in the $i^{th}$ hidden layer. We provide a corresponding architecture for each network above. Our design are described in the format of $X_0 \times X_1 \times X_{ADD}$. $X_0, X_1$ and $X_{ADD}$ are the numbers of PEs in the PEA Array0, PEA Array1 and adder tree. As the deepest depth of the nine networks is just 4, the number of PEs in PEA array2 is always 0 which is not shown in Table 1. For the 7 benchmarks, we define the number of PEs in PEA Array0 as 16 just according to the numbers of neurons in the first hidden layers, most of which are multiples or submultiples of 16.

Table 1. BENCHMARKS AND CORRESPONDING DESIGN

| Benchmarks | Benchmarks topology | Our design |
|---|---|---|
| fft | 1x4x4x2 | 16x4x4 |
| inversek2j | 2x8x2 | 16x0x16 |
| jmeint | 18x32x8x2 | 16x8x8 |
| jpeg | 64x16x64 | 16x0x16 |
| kmeans | 6x8x4x1 | 16x4x4 |
| sobel | 9x8x1 | 16x0x16 |
| nettalk | 203x120x26 | 16x0x16 |

Table 2. POWER CONSUMPTION IN NTV AND SVR

| Benchmar | NTV | | SVR | |
|---|---|---|---|---|
| | $P_{dyn}$(mw) | $P_{sta}$(uw) | $P_{dyn}$(mw) | $P_{sta}$(nw) |
| fft | 0.618 | 34.28 | 2.689 | 3.21 |
| inversek2j | 0.866 | 29.36 | 4.419 | 3.19 |
| jmeint | 3.672 | 61.77 | 17.88 | 5.74 |
| jpeg | 6.824 | 51.22 | 31.356 | 4.82 |
| kmeans | 0.804 | 34.52 | 3.809 | 3.21 |
| sobel | 0.870 | 34.63 | 3.286 | 3.19 |
| nettalk | 208.6 | 51.66 | 952.839 | 4.77 |

### Performance and power consumption

We use multiple process database to do the synthesis. Parts working at clk-4, 280M, are synthesized using the SMIC 40nm SVR database. As SRAMs occupy most of the power consumption, we use the SMIC 40nm NTV process database to do the synthesis at a frequency of 70M. We define the number of SRAMs in data interface part as *8*, i.e. *S = 8*, clk-2 = 70M. Modules working at clk-1 synthesized using the SMIC 40nm NTV process database with a frequency of 70M, i.e. clk-1 = 70M.

We compare our design with the work in [7] who trained a hardware architecture to mimic a region of code of approximate programs. Work in [7] does not take an optimization on the scheme of the data and weights uploading into consideration. Thanks to our weight and data interface architectures, the parallel PEs in each PEA and the application of multi clocks, although working at a low frequency, our design gets a higher performance in each benchmark than that of work in [7], shown in Fig. 5.

Then we synthesize our design using the SMIC 40nm SVR database. The dynamic Power ($P_{dyn}$) and static power ($P_{sta}$) in NTV and SVR are shown in Table 2. In each benchmark, the static power consumption accounts for most of the power consumption. The benchmarks working in NTV has a lower power consumption than those in SVR. From Fig. 6, we can see the total power consumption of benchmarks working in NTV ($P_{ntv}$) is an order of magnitude lower than those ($P_{std}$) in SVR.

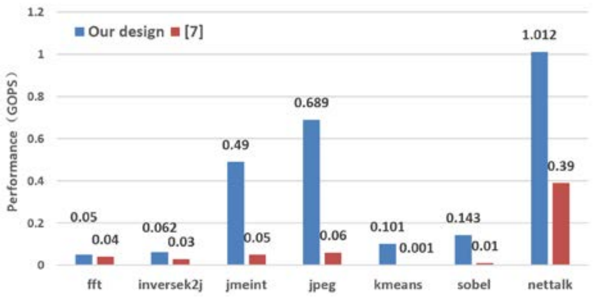

Fig. 5. The performance of our design and [7]

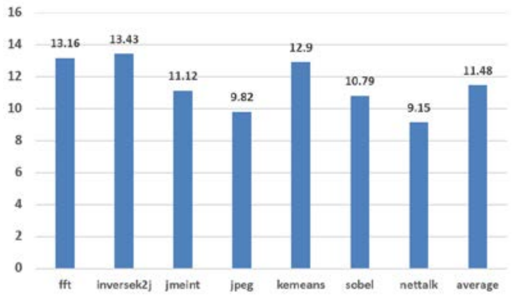

Fig. 6. $p_{sta}/p_{ntv}$ in each benchmark

## Optimization on multipliers

We replace a full-bit multiplier with 4 half-bit ones. The latency of a 16-bit multiplier at NTV is about 9.051 ns and so the theoretical highest frequency is about 100M. But considering the effects of process deviations, it will be some lower. But replacing the 16-bit multiplier with four 8-bit ones can decreases the latency to about 4.492ns and the highest frequency on the critical path will be about 140M, i.e. clk-1 = 140M, clk-4 = 560M. But considering the limit of the data ports in our design, we define clk-1 as 100M and so clk-4 = 400M. Considering the SRAMs working at NTV, we define $S$ as 16 insuring the SRAMs to work at 50M, lower than 70M.

## Conclusion

We propose an accelerator design guideline for ANN with arbitrary scales and depths. We train 7 networks and define an architecture for each benchmark. Multiple processing databases and clocks are applied in our design to get a high throughput and performance. We improve the working frequency by optimizing the multiplier on the critical path and change the compound mode of the SRAMs to ensure that they are working at a valid clock domain.

## References

[1] Domingos P O, Silva F M, Neto H C, “An efficient and scalable architecture for neural networks with backpropagation learning.” In Field Programmable Logic and Applications, 2005, pp.89–94

[2] N. Chumuang et al, “Intelligent handwriting thai signature recognition system based on artificial neuron network,” in TENCON 2014-2014 IEEE Region 10 Conference. IEEE, 2014, pp. 1–6

[3] A. R. Omondi et al, “FPGA implementations of neuralnetworks. Springer”, 2006, vol. 365

[4] M. Pietras, “Hardware conversion of neural networks simulation models for neural processing accelerator implemented as fpga-based soc,” in Field Programmable Logic and Applications (FPL), 2014 24th Interna-tional Conference on. IEEE, 2014, pp. 1–4.

[5] L. Kou and W. H. Robinson, “Impact of process variations on reliability and performance of 32-nm 6t sram at near threshold voltage,” in VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on. IEEE, 2014, pp. 214–219.

[6] I.-C. Wey et al, “Near-threshold-voltage circuit design: The design challenges and chances,” in SoC Design Conference (ISOCC), 2014 International. IEEE, 2014, pp. 138–141.

[7] Esmaeilzadeh H. et al. Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on. IEEE, Vol.33, pp.16 - 27.