An Binary Weight Neural Network Accelerator Based on FPGA

1. Introduction

   This paper modifies a CNN-based speech recognition algorithm into a binary weight neural network model where weight value is +1 or -1. Also, this paper uses Matlab quantify functions to turn float-type feature data into fix-point data level by level, whose loss of accuracy will be covered by the performance of fix-point computing on FPGA platforms. Last but not least, this paper designs a multi-PE BWN accelerator on FPGA and achieve over 300x accelerating ration compared with Matlab codes on i7-8700K.

2. Background
3. Speech Recognition Model

3.1 Model Architecture and Weight Binarization

This CNN-based speech recognition model is trained on Tensorflow speech command data set and can recognize the six sort of short speech segment "up", "down", "yes", "right", "left" and unknown words. First of all, this model uses MFCC algorithm to change an audio file into a float-type tensor whose dimension is 20*49*1. Then this tensor will be sent into a convolution neural network which contains two convolution layers, three full connected layers. The detail of model architecture is shown in Fig.1. To be noticed, weight is still in float mode at this stage. Finally, this model outputs the possibility of six type of labels via softmax function.

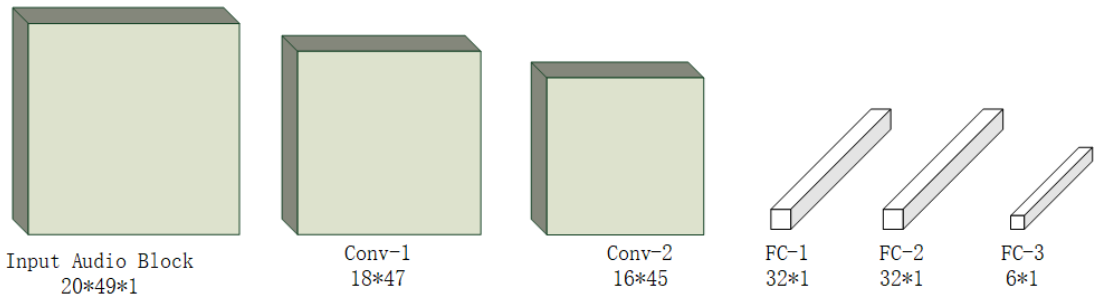

Fig.1 Convolution Neural Network Architecture.
This model contains two convolution layers and three full-connect layers. All layer's kernel size is 3 and convolution stride is 1. There is no padding and expansion in this network, which is convenient for us to accelerate.

When model parameter is fixed in the training process, we transfer float weight value into (-1, 1) by tanh function and then uses series scale method discretizing them to 0 or 1, and finally to -1 or +1. Fig.2 shows how we handle all weight data. This stage's binary-weight-float-data model can provide the accuracy no less than 85%.

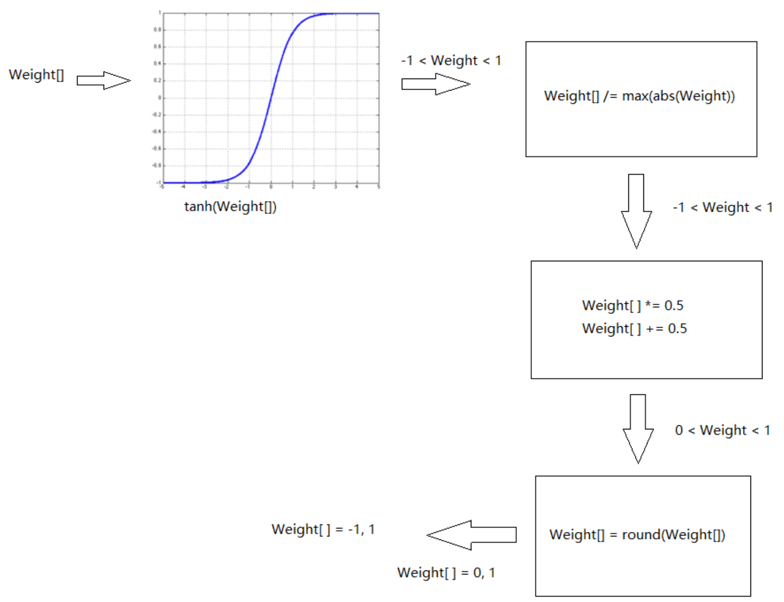

Fig.2 Weight Binarization Process

3.2 Feature Data Quantification

The data computed by each level in the neural networks can be divided to two parts: middle results after MAC operations and before batch-normalization and the output data be batch-normalized. In some level, absolute value and variance value of middle results can be huge, but both of them will be narrowed down after batch-normalization. So it is necessary to take different quantification data type on middle results and output results within one level, besides the results of each level distribute in different range. To turn feature data into fix-point with accuracy's loss as less as possible, we ensure data range in different stages within a level and data range between different levels, then use matlab quantizer function library to find the best quantization format combination in each level an in each stage. After this step, we get a binary-weight-fix-point-feature CNN based speech recognition model. The quantization result will be discussed in Experiment part.

4. Accelerator Architecture

The target model is small, shallow and binarized-weight, so the main method we use is reducing storage latency by putting shared binarized parameter on chip, accelerating neural network level by level and designing pipeline between levels.

4.1 Parameter Storage

Unlike some neural network accelerators having to use DRAM instead of on-chip memory to store parameter, our model has small-sized parameter. Table.1 shows the detail information of this model's parameter.

type	layer	filter number	kernel	bit-width	size
convlution	conv-1	32	3*3*1	1	36B
batch-normalization	conv-1	—	—	22/22/32/22	6464B
convlution	conv-2	32	3*3*32	1	1152B
batch-normalization	conv-2	—	—	21/21/21/21	2688B
full-connect	fc-1	32	16*45*32	1	90KB
batch-normalization	fc-1	—	—	28/28/32/28	3712B
full-connect	fc-2	—	—	1	128B
batch-normalization	fc-2	—	—	21/21/21/21	2688B
full-connect	fc-3	—	—	1	24B

Table.1 The Detail Information of Parameter.
FC-1 layer occupies most of the parameter size, while other layers' data is rather tiny and can be directly stored on chip.

Considering the scale of FC-1 parameter, it is naturally to share them between multi-PEs. We set all PEs to work synchronously and fetch exactly the same pretrained data at the FC-1 computing step. Shared memory block is consisted of 32 block memory generators, each for one kernel in FC-1.

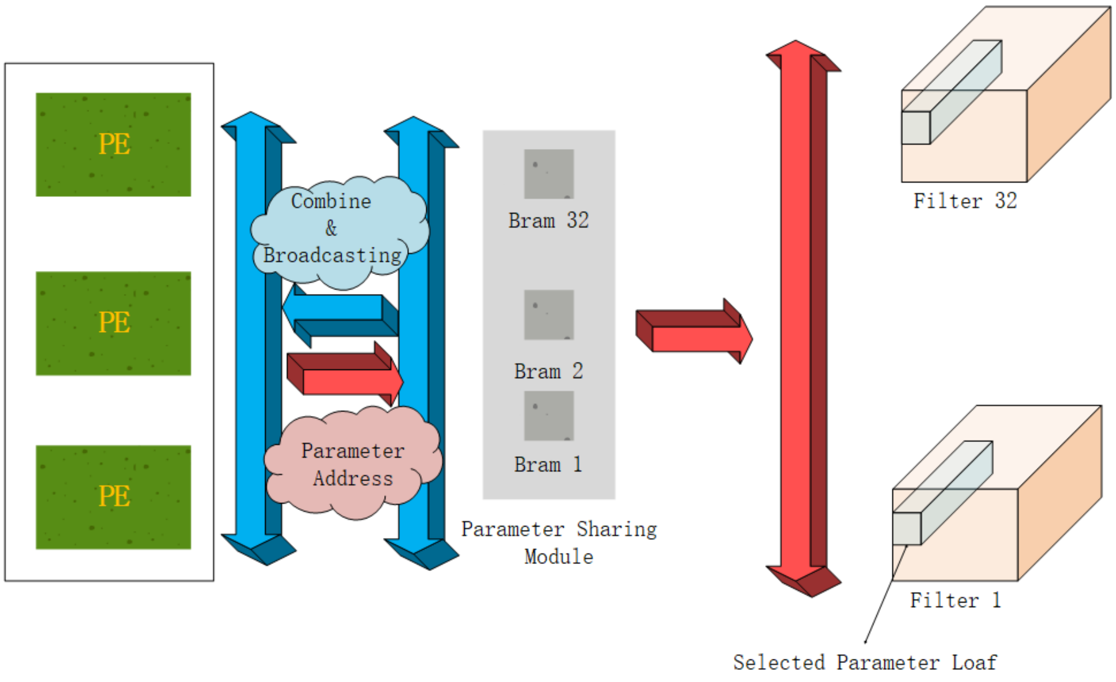

Fig.3 FC-1's Parameter Sharing Structure.
PE sends target parameter loaf address to parameter sharing module, which is a line address to Bram blocks.

Parameters are stored along the third dimension in BRAM, so it can be consistent to the input data's organization method of FC-1 layer. By broadcasting, shared-parameter data is sent to all PEs.

4.2 Bit-width Expansion

Inside one convolution or full-connected layer, the input fix-point feature data is usually under normal distribution and varies in a small range, however, after MAC, the data's variance can be huge and irregular. Batch-normalization relies on these middle results' mean and variance to modify the data distribution after MAC, which is vital to final accuracy. If we still apply the same data format as the input feature to normalization step, it will bring unnecessary loss to final result.

To handle this situation, we introduce bit-width expansion by giving extra decimal number width to both feature data and normalization parameter when computing. After DSP outputting multiply results, those extra decimal bit-width will be cut and the data will return to original input format.

Also, the middle results of MAC process have the similar needs for bit-width expansion. The different point is MAC's results need more bit-width in integer part instead of decimal number. We apply expansion to both normalization step and MAC accumulator, it turns out that this method ensure computing accuracy on hardware.

4.3 Level-by-level Pipeline

The key point to level-by-level pipeline's implementation is to balance running periods between levels. In deep convolution neural networks such as VGG-16 and AlexNet, it is difficult to keep this balance because as networks going deeper, the deep level will demand levels ahead to generate feature result at a faster speed, which is beyond current computing power limit.

The target neural network is shallow and tiny, so it is relatively easy to keep balance between two convolution layers. We change Conv-1 layer's data-fetching address generating method to cooperate with Conv-2 layer's computing mode. Also, we expand the scale of parallelism of Conv-1 layer's function module in order to generate one series of Conv-2 layer's input data in one level pipeline's beat.

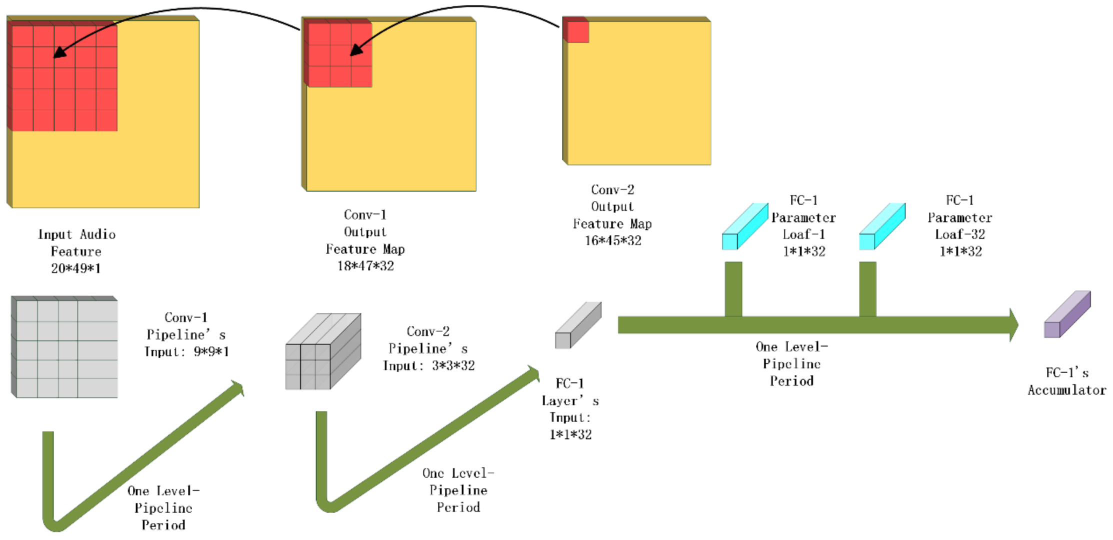
Input Audio Feature 20*49*1
Conv-1 Output Feature Map 18*47*32
Conv-2 Output Feature Map 16*45*32
FC-1 Parameter Loaf-1 1*1*32
FC-1 Parameter Loaf-32 1*1*32
Conv-1 Pipeline' s Input: 9*9*1
Conv-2 Pipeline' s Input: 3*3*32
FC-1 Layer' s Input: 1*1*32
One Level-Pipeline Period
FC-1' s Accumulator
One Level-Pipeline Period
One Level-Pipeline Period

Fig.4 Balancing Conv-1 and Conv-2.

To produce one result in Conv-2 layers, this neural network has to compute one 3*3 slide window on Conv-1's output map and compute nine 3*3 slide windows from a 5*5 area on input audio feature. We expand the scale of Conv-1's computing array to make it generate nine 32*1 vectors in one macro pipeline period, and in next period these vectors will be sent to Conv-2 and generate one 32*1 vector for FC-1 layer. Conv-2's function is running in one macro period as well. FC-1 layer is the bottleneck of this accelerator, it is unbearable to keep balance this layer with two convolution layers for its huge computation utilization, but we can ensure FC-1 layer keep getting one vector in each macro period for accumulation.

By adapting pipeline on levels, target neural networks can be accelerated without putting between-levels-result into DRAM and thus reducing memory cost, in another word, we keep data stream in level pipeline from input audio feature to final predict results without stop. The accelerator only communicates with DRAM in fetching and final writing back. Inside each level's pipeline, we also divide all computing into some function parts like vector computing unit, normalization unit in pipeline method, which help to rise hardware running frequency.

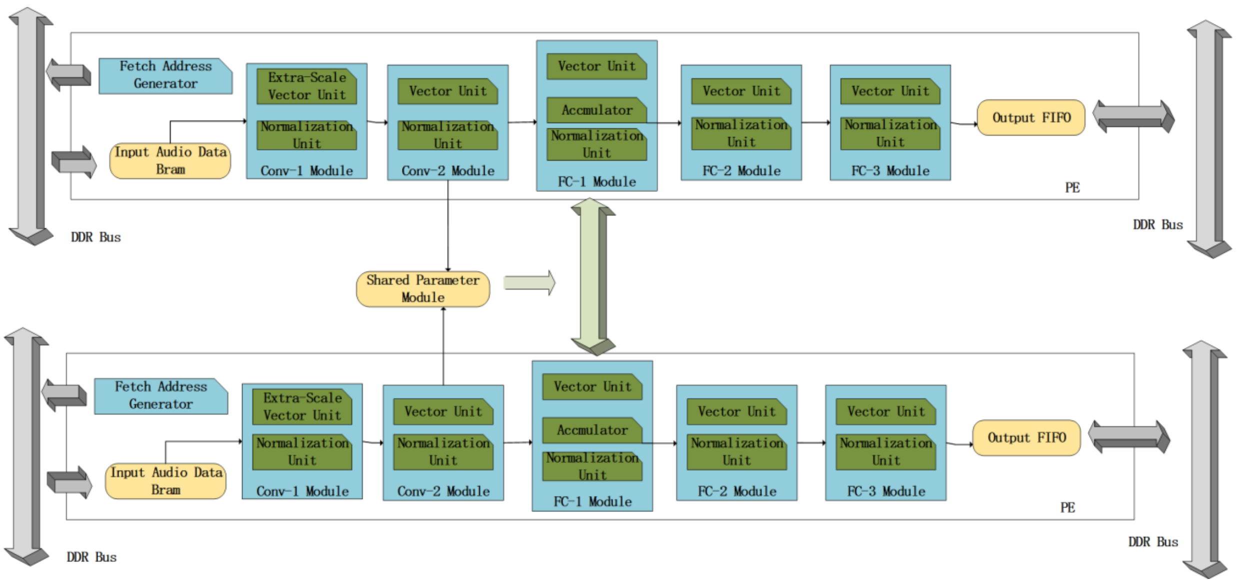

Fig.5 Hardware Architecture.

An accelerator contains one or multi PEs, one DDR bus and one DRAM (not shown in this figure). Every single PE has own function modules including five layers neural networks process unit, address generator, input and output data storage. Conv-1 module, Conv-2 module and FC-1 module's accumulator works in a pipeline method.

5. Experiment

5.1 Quantified Model's Performance

We use Matlab-2018a's quantizer function to turn feature and batch-normalization parameter into fix-point data with saturate mode. To find the best bit-width setting method, we run a couple of quantification experiments and compare their accuracy performance.

Decimal Bit-width of Feature	Decimal Bit-width of Normalization Parameter	Accuracy
13	12	83.32%
12	12	83.20%
11	12	83.38%
9	10	83.58%
8	9	83.73%
7	8	82.91%

Table.2 Accuracy Performance Under Different Bit-width

It is shown that when feature data adapts eight bit-width and normalization parameter adapts nine bit-width, this model will achieve best performance. To be noticed that hardware cannot handle division operation as easy as Matlab code, so we turn variance's division in normalization function into reciprocal multiplication and expand decimal bit-width to ensure accuracy.

We run the whole dataset (contains 1512 audio segments) under 8700K and Matlab-2018a circumstance, 8700K and Matlab-2018a with parallelism library circumstance and multi-node 9700K circumstance with MPI. We set their running time as baseline to compute accelerating ratio of our design.

8700K without Parallelism			8700K with Parallelism		
Function Segment	Time (Seconds)	Ration in Total Time	Function Segment	Time (Seconds)	Ration in Total Time
Data Loading & Pre-process	9.413601	7.40%	Data Loading & Pre-process	9.804942	19.74%
Conv-1's Quantizer	0.000508	<1%	Conv-1's Quantizer	0.000195	<1%
Conv-1	3.084229	2.43%	Conv-1	1.803836	3.63%
Conv-2's Quantizer	0.000694	<1%	Conv-2's Quantizer	0.000307	<1%
Conv-2	114.582	90.14%	Conv-2	38.02353	76.56%
FC-1's Quantizer	0.006863	<1%	FC-1's Quantizer	0.006184	<1%
FC-1	0.023377	<1%	FC-1	0.025257	<1%
FC-2's Quantizer	0.000834	<1%	FC-2's Quantizer	0.000588	<1%
FC-2	0.001659	<1%	FC-2	0.001316	<1%
FC-3's Quantizer	0.000945	<1%	FC-3's Quantizer	0	0
FC-3	0.001589	<1%	FC-3	0.000981	<1%
Total	127.1163	100%	Total	49.66721	100%

Table.3 Running Time on CPU Platform

5.2 Accelerator's Performance

We implement our single-PE version accelerator design on Xilinx KU-115 FPGA platforms and test running time, accuracy and power performance. To measure time and power better, we set accelerator to compute dataset circularly for at most two hours.

Running Time (Seconds)	0.4116
Static Power (W)	9.63
Dynamic Power (W)	10.06
Accuracy	?
Utilization	38%
Accelerating Ratio (8700K)	308.83
Accelerating Ratio (8700K & Parallelsim)	120.67
Accelerating Ratio (9700K & MPI)	?
Throughput (Audio Segment Per Sec)	3675.9
Running Frequency (MHz)	150

Table.4 Accelerator's Performance

Table.3 shows that Conv-2 layer is the most time-costing function, however, by using balanced level-by-level pipeline, our accelerator can eliminate bottlenecks in the original neural networks and achieve excellent accelerating performance. Also, the data stream inside the pipeline reduce the DDR bus communication tremendously, eliminate the DRAM limit on CPU platform.

6. Conclusion

Our accelerator is a low-power, high-speed and high-throughput hardware for a specific speech recognition model, it has excellent performance compared with state-of-art CPU platforms. We perform pressure test on our design, and it turns out a reliable and high-efficiency accelerator.