CNN-LSTM accelerator design

Aditya Shukla[1], Guoqin Yin[1], Jiajun Cao[1], Wenbo Duan[1], Rui Sun[1] and Xiuneng Lu[1]

[1]Department of Electrical and Computer Engineering, University of Michigan
{*aditshuk, guoqin, jiajunc, wbduan, rayss, xiuneng*}*@umich.edu*

I. Background & motivation

Neural networks have been demonstrated to be capable of achieving high performance in tasks like speech recognition and classification, currently CNN and LSTM networks are two mainstream in neural network design. Because the CNN and LSTM networks have different advantages in dealing with language modeling tasks, our project intends to combine the advantages of these two networks and design an accelerator to solve speaker recognition tasks.

II. Model design

We use the Librispeech dataset and get the data of 10 different speakers for out project, the data is first processed using MFCC to extract 16 different frequency data in a single timestep, a total of 40 timesteps is generated, the output is then sent into a CNN layer. In the CNN layer, the kernel size we use is 4 and 8 filters are used to generate the output. The output data from CNN layer is then sent into a LSTM layer, after the feature extraction process in the previous layer, 8 features are extracted to be the important information in solving the tasks, the LSTM layer expands these important features into 16 through modeling. The output data from LSTM module is then sent into a FC layer, the fc layer will utilize all the information processed in the previous module, and generate 10 numbers indicating the class that the sample data belongs to.

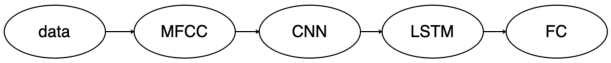

Fig. 1. Module Data Flow

In the accelerator, we change to floating point data into fixed point according to the table II:

Quantization Table

Layer	Weight		
	sign	integer	decimal
CNN Weight	1	0	7
CNN Input	1	7	0
Cnn Inside	1	8	7
LSTM Weight	1	0	7
LSTM Input	1	7	0
LSTM Inside	1	1	14
FC Input	1	0	7
FC Weight	1	0	7
FC Inside	1	5	14

III. Top Level Architechture

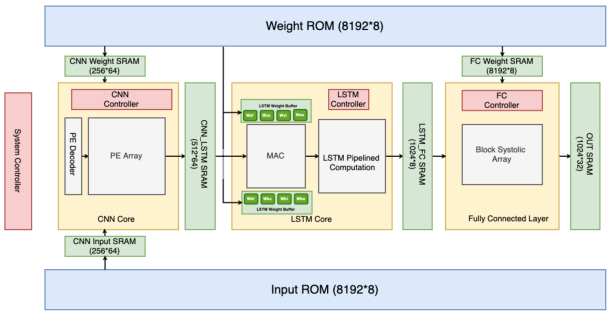

Fig. 2. Top Level Architecture

Our toplevel architecture design is shown in figure 2 Two ROMs are used to store the initial weight and input data, the data is sent into the chip at the initial stage and is stored in different SRAMS on the chip. The CNN module gets started when the loading state is completed, when the first time step data is calculated, LSTM module gets started, the module works together with the CNN module to save more calculation cycles during the entire process. When the computation in the LSTM module is completed, FC module gets started, the module loads all the data from the previous LSTM's output, when the FC module finished its computation, the output data will be stored in a 1024×32 SRAM.

IV. CNN core Design

For our design, we need to apply convolution on sequential data. Figure 3 shows how this convolution is achieved. For each data, there will be several time steps and we do convolution on timesteps.

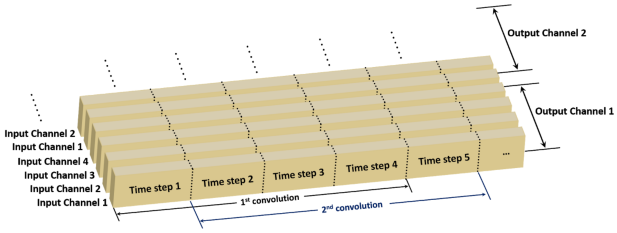

Fig. 3. Diagram of cnn in our design(take 4 input channels, 2 output channels and 4 kernels as example)

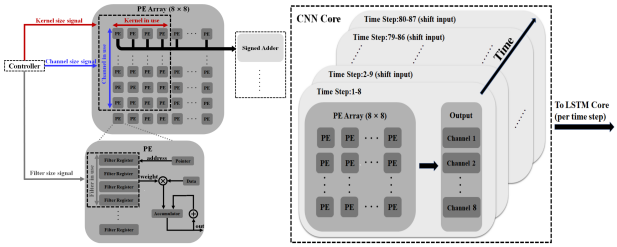

Fig. 4. Structure of hardware implementation of CNN core

Figure 4 shows how we implement this convolution in hardware. We designed a PE Array to do convolution and every PE have 16 registers to store weight and the input will be sent into each column of PE array. After the multiplication and accumulation is done in PE, output from each row will be added through a signed adder and get the final output channel result. We tried to make this CNN core more flexible so we added three control signals(kernel size signal, channel size signal and filter size signal) which will decide how many kernels and input channels we are using and how many output channels we want to get. To do convolution on time steps, new input data will be sent to the first column and every data will be shifted to the PE on the right. And then repeat doing calculation and shifting until all time steps are sent into the PE Array. For CNN core, the input data is 8 bits and we do 8 bits MAC inside the PE. The output data of each output channel is 16 bits and we intercept to 8 bits and store it in the SRAM between CNN and LSTM part. Figure 4 is the simulation result of CNN core.

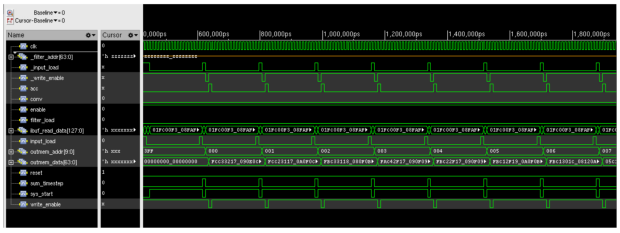

Fig. 5. CNN Simulation Results

We finished synthesis and part of APR for CNN core. In the synthesis result, the area is 14976373 μm^2 and the total power is 478.16 mW. For our testing scenario, we have 40 time steps for each data and the kernel size is 4. So in total we need to do 37 times convolution and the total time we use is 2828 ns under 250Mhz. In CNN core we have 3 blocks which are CNN Controller, PE Decoder and PE Array. By far only the PE Decoder is DRC and LVS clean. The total area of these three blocks is 3.2 mm^2. Figure 6 shows the floorplan we designed and the APR layout of PE Array.

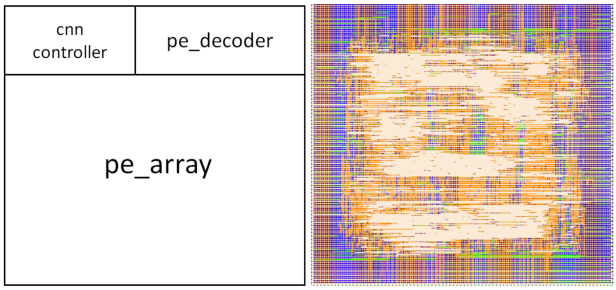

Fig. 6. CNN floor plan and layout of PE Array

V. LSTM Core Design

A. LSTM background

LSTM network has the advantage in dealing with sequential data, it has an inherent sequential logic in the calculation process, as shown in figure 7, the output of the current time step data is based on the input data of the current time step and the cell's output of the last time step data.

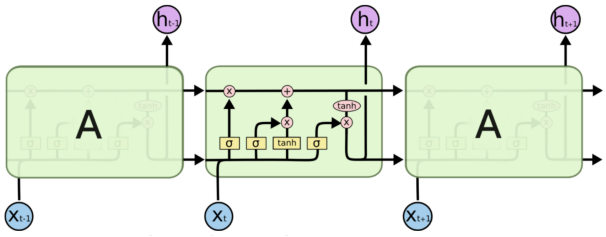

Fig. 7. LSTM unfolded model

The calculation is finished in the following steps:

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i) \quad \text{input gate} \quad (1)$$
$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f) \quad \text{forget gate} \quad (2)$$
$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b^o) \quad \text{output gate} \quad (3)$$
$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1} + b^c) \quad \text{candidate memory} \quad (4)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad \text{memory cell} \quad (5)$$
$$h_t = o_t \odot \tanh(c_t) \quad \text{hidden state.} \quad (6)$$

B. LSTM Architecture

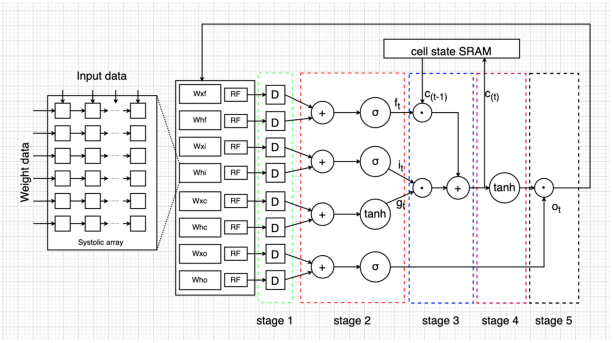

Fig. 8. Pipelined LSTM architecture

The idea of the design is from [1]. A pipelined structure and systolic arrays are used to maximize the throughput and also minimize the static power. Assuming the LSTM network requires the number of hidden nodes to be 16, the systolic array requires 16 cycles to compute all the 16 hidden data. In our design, at the i^{th} cycle of the computation, the i^{th} hidden data is computed and sent to the input of the systolic arrays, making the systolic arrays move forward by one step. The comparison between our design and traditional LSTM design is shown in table I. After 16 cycles, all the hidden data is computed and the entire system could continue the computation of the next time step. Some parameters of LSTM are shown as follows and figure 8 shows our pipelined LSTM layer architecture.

- The number of hidden node is 16.
- sigmoid and tanh functions calculation: 5 cycle.
- Total pipeline stages: 13.

TABLE I
COMPARISON OF OUR DESIGN AND TRADITIONAL DESIGN

Comparison	Clock cycles	System width(stage 1-5)
Traditional series design	28	16
Our design	29	1

Both the designs of LSTM in our project and in the reference paper have the feature of throughput optimization. Since the systolic arrays are the biggest part in the entire design, the pipeline structure can help the systolic arrays continue the computation and only be stalled in a small period of time. The design in our project also have less bubbles during the calculation stage. The states that divided in the original paper will make the systolic arrays stalled when every data is sent into the pipeline, the entire number of bubbles equals to the number of hidden nodes in the design, typically 16, 32 or 128. In our design, the total number of bubbles equals to the number of stages of the pipeline, which is always less than the number of hidden nodes utilized in a LSTM layer

C. Synthesis results:

The clock frequency of the LSTM layer is 250 MHz. From the synthesis report, the power consumption of the pipelined structure is 81.380 mW, the area is 5.061 mm^2. Combined with the systolic array and weight register, the total power is 872.941 mW and the total area is 87.221 mm^2. We are still in the process of fixing the DRC LVS error in some sub-blocks. But we've designed a floorplan for the whole layer as shown in figure 9. The total estimated area is 8mm × 8mm and figure 9 shows the layout of LSTM pipeline structure.

D. APR results:

We are still in the process of fixing the DRC and LVS error in some sub-blocks. But we've designed a floorplan for the whole layer:

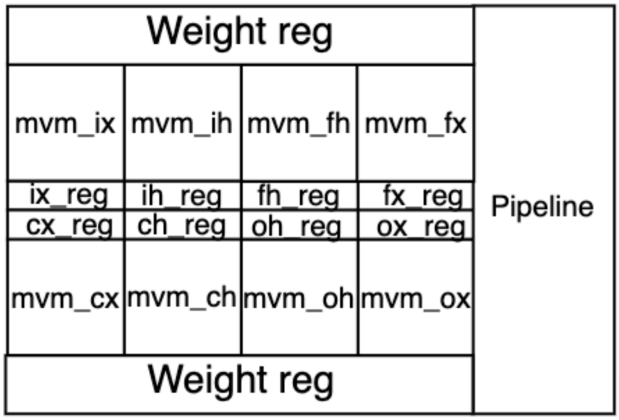

Fig. 9. LSTM Floorplan

The total estimated area is $8mm \times 8mm$.

Our block level apr result are shown in Figure 10.

Block name	DRC	LVS	Area	Density
Controller	clean	unclean	$0.12mm \times 0.12mm = 0.0144mm^2$	61.93%
Systolic array 16×16	unclean	unclean	$1.6mm \times 1.6mm = 2.56mm^2$	53.50%
Systolic array 16×8	unclean	unclean	$1.2mm \times 1.2mm = 1.44mm^2$	69.01%
Weight register files (1024)	clean	unclean	$0.8mm \times 0.4mm = 0.32mm^2$	75.15%
Pipeline structure	unclean	unclean	$1.5mm \times 1.5mm = 2.25mm^2$	29.07%

Fig. 10. LSTM Block APR Results

Here's the LSTM Sub-blocks layout:

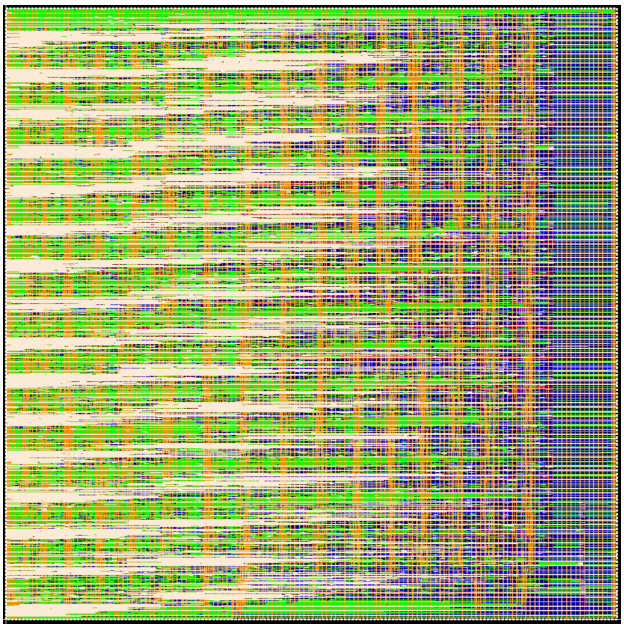

Fig. 11. 16×16 Systolic Array APR layout

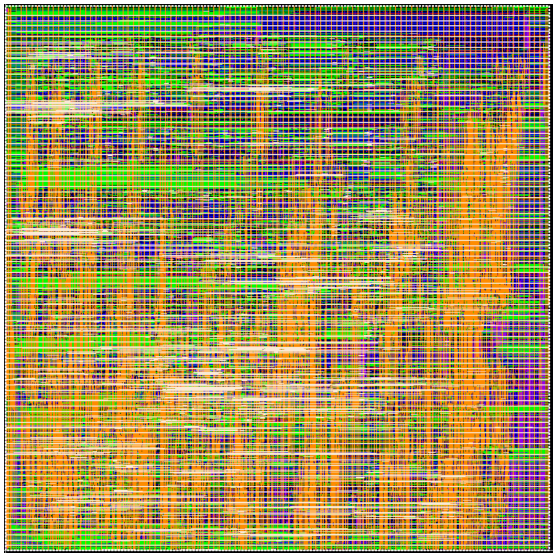

Fig. 12. 16×8 Systolic Array APR layout

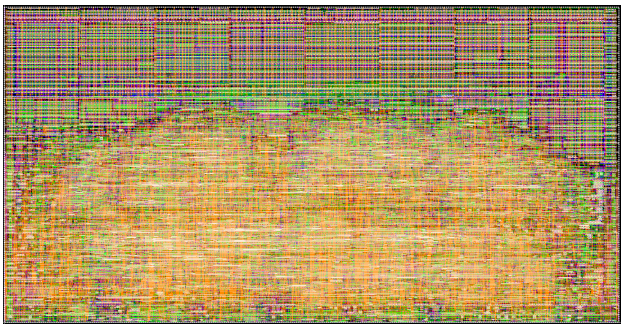

Fig. 13. LSTM Weight Register APR Results

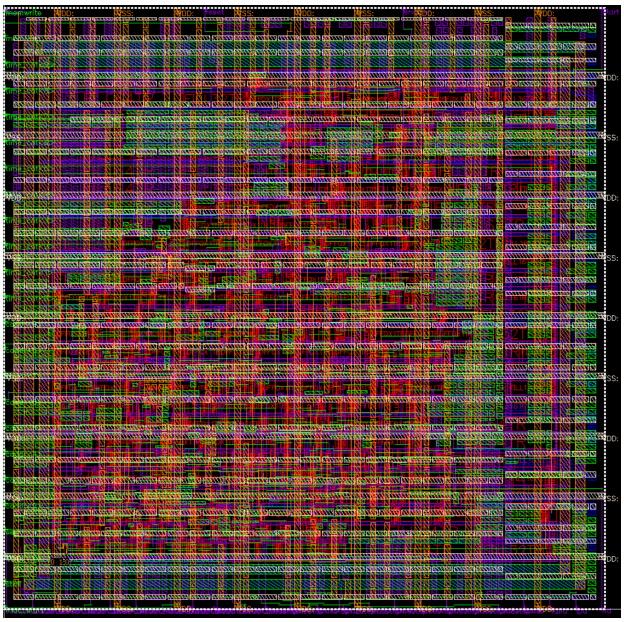

Fig. 14. LSTM Controller APR Results

E. FC layer (Block systolic array) design

Since input data dimension is 592×8 bits and weight matrix is $592 \times 10 \times 8$ bits, directly using 592×10 systolic array will be area-consuming. A trade-off needs to be made between area and speed to make FC design more balanced. Thus, we utilize block systolic array to implement our FC layer. The high-level idea to implement block mvm(matrix-vector multiplication) is shown in Figure 15:

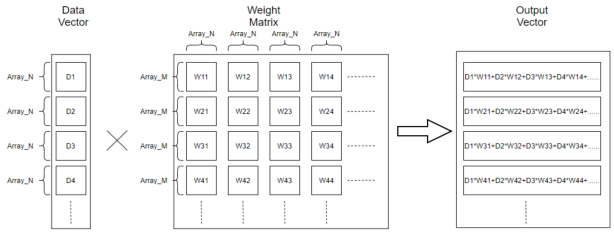

Fig. 15. Block MVM structure

We mainly use an $array_N(= 16)$ by $array_M(= 10)$ systolic array as the computation core to compute $D_i W_{ji}$, Then in accumulation registers, we add all partial results $(\sum_i D_i \times W_{ji})$ and get the final result. Fig 16 shows the state transition between idle, computation, and accumulation to control how FC layer works.

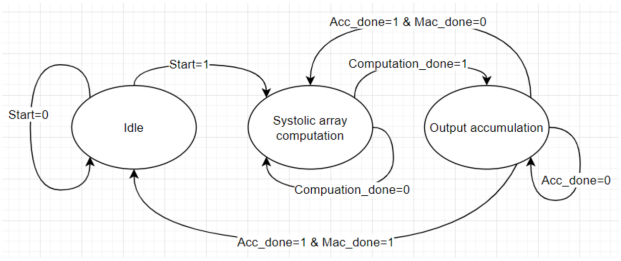

Fig. 16. FC Controller FSM Diagram

In this way, although we need more clock cycles to compute matrix vector multiplication, chip area will be significantly reduced. And Fig.17 shows the result of functional simulation.

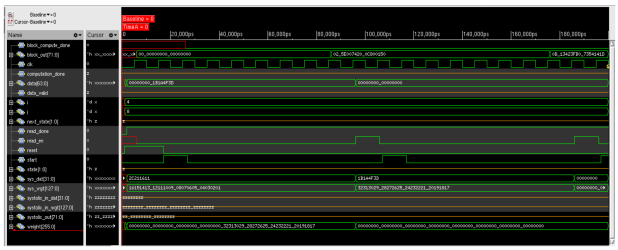

Fig. 17. Functional Simulation

After synthesis, the estimated area is 5.45mm2 and estimated power is 32.72 μW. Automated layout after APR is shown in Fig.18.

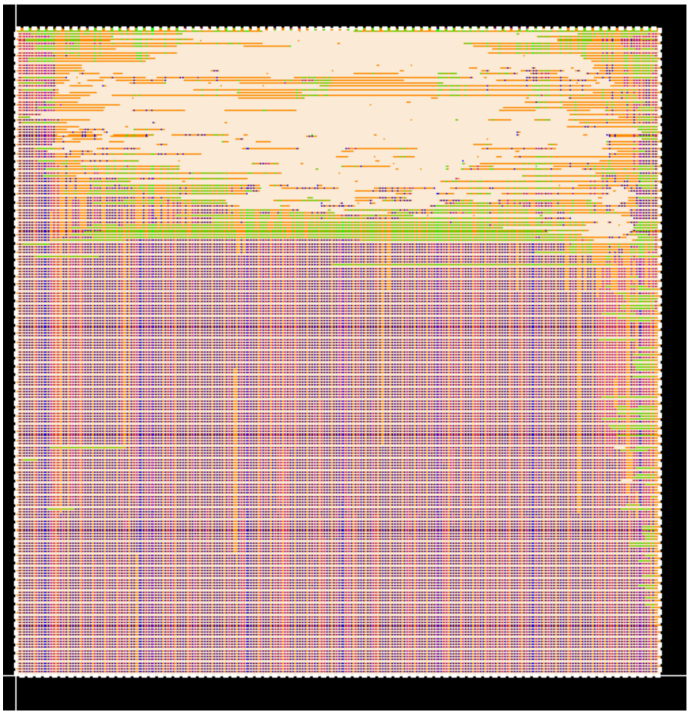

Fig. 18. FC Apr layout

VI. Level of completion

Figure VI shows our level of completion. By far we are working on block level APR and the toplevel floorplan is shown in figure 19.

	Behavior	Synthesis	Block synthesis	Post-syn verification	Block APR	Post-APR verification
CNN	done	done	done	done	In process	Not start
LSTM	done	done	done	done	In process	Not start
FC	done	done	done	done	In process	Not start
Toplevel	done	done	done	done	In process	Not start

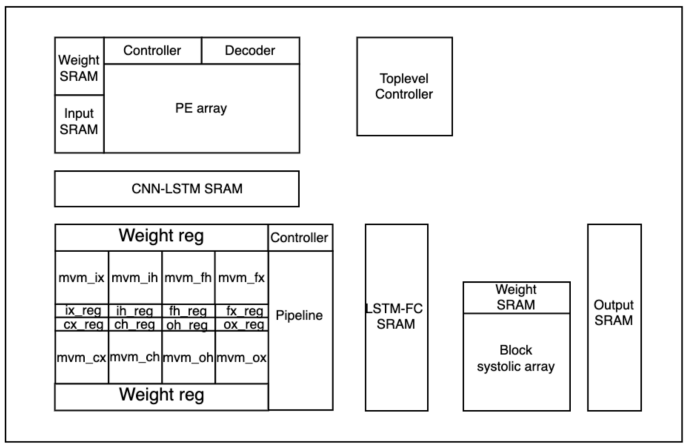

Fig. 19. Toplevel floorplan design

VII. Contribution

- Aditya Shukla: Software simulation; model quantization and accuracy simulation for analog MACs
- Guoqin Yin: LSTM verilog and syn/apr
- Jiajun Cao: FC layer design and syn/apr
- Rui Sun: CNN core verilog coding and syn/apr
- Wenbo Duan: Model, LSTM/Toplevel design and verilog, LSTM syn/apr
- Xiuneng Lu: LSTM verilog coding, weight reg syn/apr

References

[1] E. Azari and S. Vrudhula, "Elsa: A throughput-optimized design of an lstm accelerator for energy-constrained devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, pp. 1–21, 02 2020.