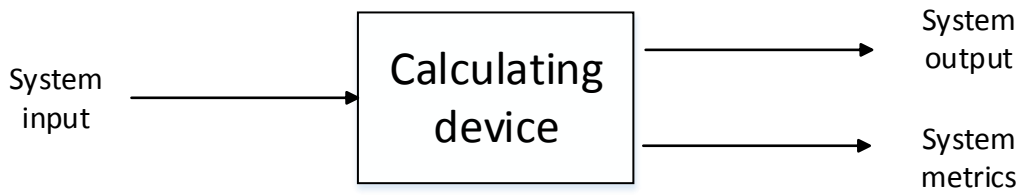
Catalog

About Pulsesim

Pulsesim Overview

Pulsesim is an architecture simulator for simulating the accelerator designed for accelerated deep learning algorithm. The architecture simulator is a tool to reproduce the behaviors of calculating device.



Pulsesim is coded by C language, and compile to an executive application through compiling by cpp compiler.

The Function and Development Intention for Pulsesim

The time of hardware is long, simulator can help to accelerate the 验证 of designed instruction set and save the time of developing. So that, we can concentrate more on designing the instruction set and hardware and explore more design space.

Setup Pulsesim

Under Windows

Create a windows application platform project by visual studio 2010 or above. And copy all the files to the project category, add the source code documents to the project. Then compile the project, the pulsesim.exe executable program can be created. You can also use other c++ compilers. Then copy all the source code documents to the new created c++ project file, then compile the project. Finally, the executable program can also be created.

Under Linux

Decompose the composed package, directly make in terminal, and then you can get the pulsesim executable program.

Pulsesim Modes Overview

Fast Running Mode

This mode is used for instruction program quick running, and get the data after the implement of the program. Only a small part of information can be got, such as the cycle of programing running, time and data and so on. The detail will be introduced in the following case.

Creat RTL Binary Code

Pulsesim can be also used to provide instruction and data in memory for the RTL design. Under this pattern, pulsesim will create the binary code according to the disorder executable instruction program and the binary code of and data which are used in the execution processing. Which means that while instruction program are Out-of-order executing and after the instruction issuing, the related binary code will be created. In the meantime, in order to verify RTL more conveniently, the data of the executive results also create.

Get Performance Data

To analyze the designed hardware more conveniently, the data for helping to analyze the performance will be created, such as leakage bandwidth of register and memory, data reuse information and so on. The detail data will be given in the following case.

Normal Mode

This mode means real work mode, and supports to verify the design of program, gets the data after the program execution, gives the data and instruction to support RTL design, and get related performance information. That’s to say, this mode includes all the functions of the upper 3 modes.

Debug Mode

This mode is used for tracking instruction execution, debugging the program, printing the changes of memory and register each cycle after the execution of instructions. Please pay attention to choose this mode, because of the high request of space to store the printed data.

Pulsesim Parameters

Running Pption Parameters

Add [-p] [inst.txt] and [-m] [mem.txt] to point out the place of instruction program and memory data. Without this parameter, you have to put the instructions and the memory data document in the root directory of the program, and named them as inst.txt and mem.txt.

Other parameters are optimal.

[-r] [pulsesim.rpt] is used for modifying the place of the instruction execution report file. It will be created in pulsesim.rpt in report file in the root directory of the program default.
[-t] is used for single-step running。
[-h] is used for showing detail information of related parameters.

Internal Configurable Parameters

Pulsesim internal configurable parameters include two part, one is in ins.conf and another is in ps.conf configuration file。ins.conf include configure information of instruction, such as:

[ins]
SLOAD = load
LOAD_CYCLE=3
loadWordConsumed=2

The above setting are used for configuring the instruction mnemonic, the cycle for instruction execution and the number of bytes in RLT coding of load instruction.

The ps.conf configuration file includes the related configuration of pipeline execution of pulsesim.

[regs]
#register entry size
RG_SIZE=16
#RV register number
RG_NUM=16
#BV register number
BV_NUM=4
#RDV register number
RDV_NUM=4

#EXTV register number
EXTV_NUM=8
#MAT structure entry size
V_SIZE=16
#MAT structure number
V_NUM=16
#NAT structure layer number
M_LAYER=8

[memory]
#initial memory size
INIT_MEM_SIZE=8192

Above is the configuration of size of memory and register.

[pipeline]
fetch_num=10
RTL_A_WORD=5

Above are used to configuring the size of pipeline multiple issue window, the size of value window of bottom RTL which decides the number of executable instruction in execution phase in one time of pulsesim.

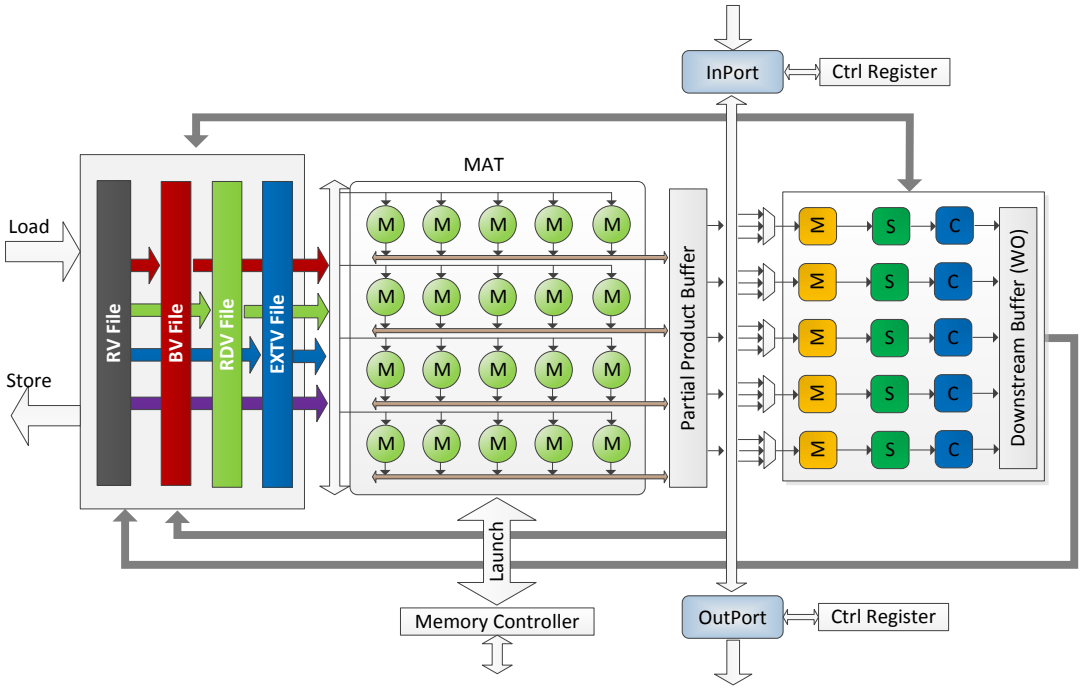
[localBuffer]
MEM_BUFF_ENTRY_SIZE=1024
MEM_BUFF_BANK_SIZE=256

[psMode]
ps_mode=5

The choice of mode of pulsesim. Optimal parameter 1,2,3,4,5 represent the 5 modes above.

Pulsesim Hardware Overview

The hardware structure is shown as the following

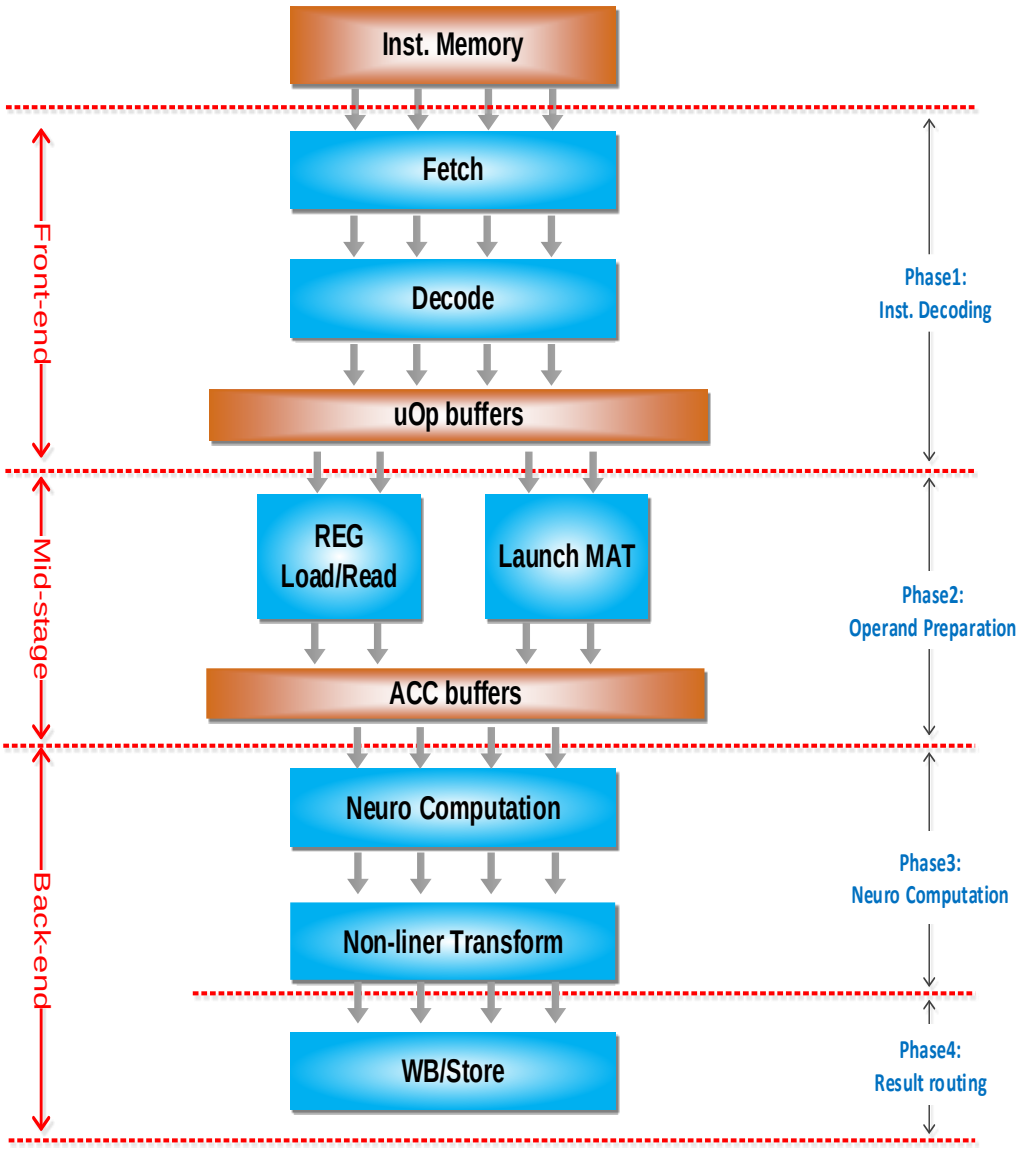


Localbuffer structure is designed for accelerating loading number from memory. By using launch and wb instructions before load and store, we can fetch the data from MEM to localbuffer in advance.

The matrix (MAT) in the hardware structure is the Pulse Unit (PU), PU contains a group of multiplier and adder, which can at least completely finish all the calculation in neural network. Owing to this design, we can achieve the vector multiplication and addition operation horizontally and vertically in MAT, and vector and vector multiplication and addition operation horizontally and vertically in enable column or row in MAT.

Pulsesim Internal Structure Overview

The pipeline structure is shown as the following.



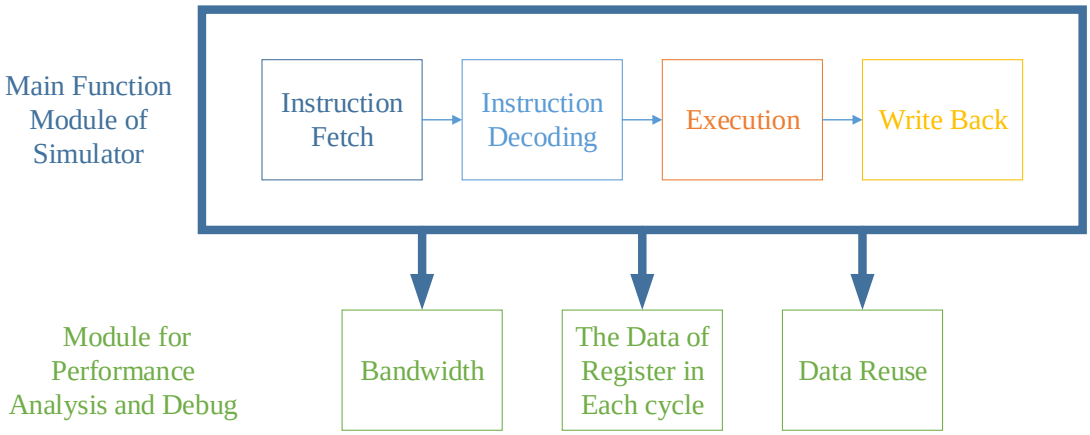
We achieve the out-of-order execution by using multiple instruction issue, forwarding and so on. When submit an instruction, it waits only if it cannot submit, otherwise it submits. This strategy speed up the running of instruction, but burden decode phase.

Simulator Structure

Simulator is mainly used to simulate the 4 pipeline in hardware and the out-of-order execution, provide binary code and data to hardware test and execution. However, in order to make it more convenient to instruction and hardware design and verification, the simulator is asked to output some useful information for verification.

Therefore, the simulator not only achieve 4 pipeline out-of-order execution, but also add print function which is helpful in performance analysis and debug in RLT design, such as bandwidth and data reuse information in register, local buffer, local memory and memory. What's more, these functions are optional, users can decide whether to use the function or not during the instruction execution time through the relative parameters.

This is the structure of pulsesim simulator.



Pipeline Structure

Pipeline is a technology to achieve instruction executing in the same time based on the parallelism between the operations, which becomes a key point in high speed CPU.

Hazard is the barrier in pipeline, which prevents the next instruction in instruction stream from executing in designed clock cycle. Which means hazard decrease the ideal performance of pipeline. There are three types of hazard.

1. Structure hazard: also called structure conflict. It happens when hardware cannot support multiple instruction executing in the meantime, and causes source conflict.

2. Data hazard: It happens when the operand of next instruction depends on the unavailable result of a previous instruction still in the pipeline in the meantime, which result has not yet been computed.

3. Control hazard: which is caused by branch instructions or other instructions which change the PC in pipeline.

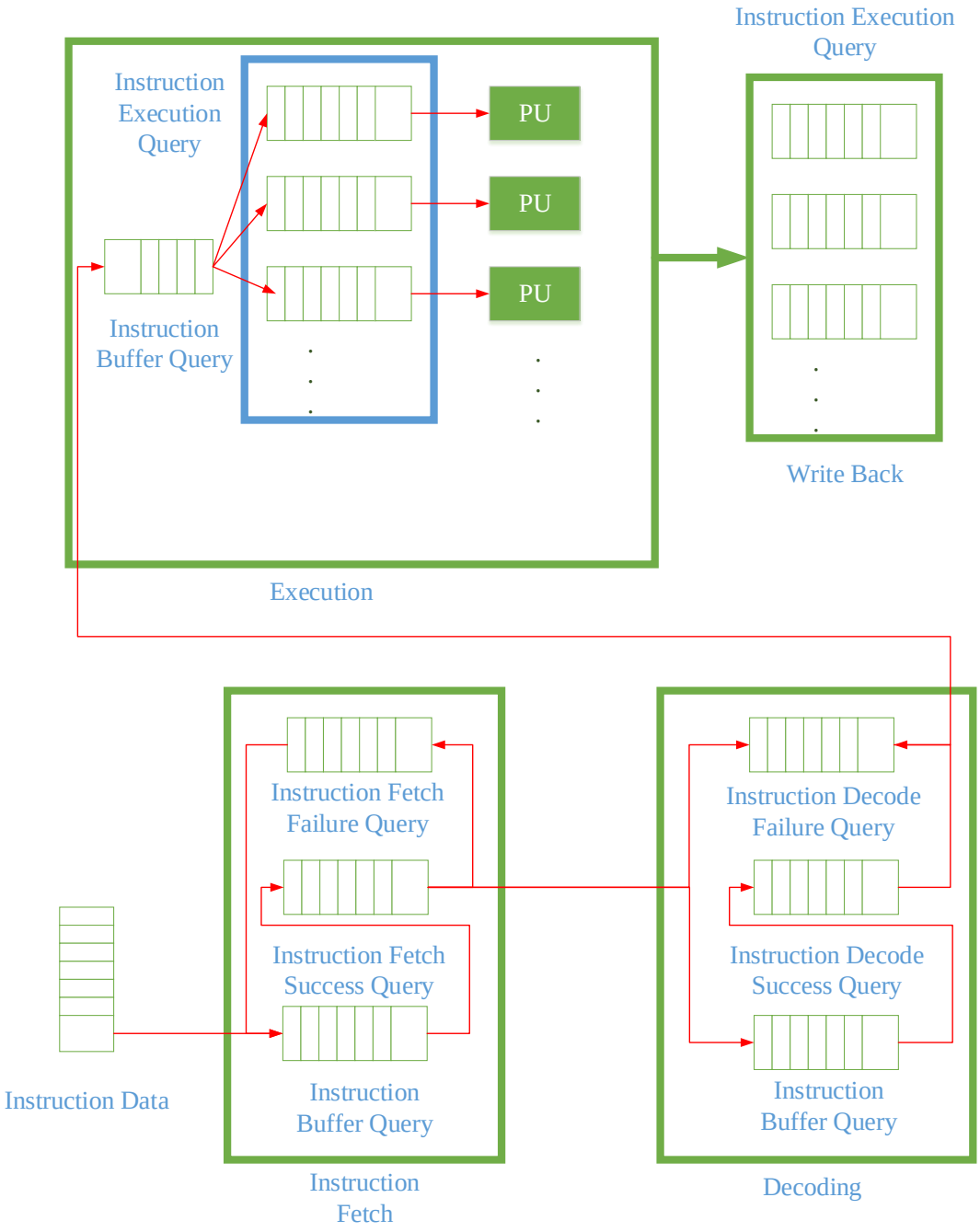
According to the previous chapter, we don't design branch instruction. Therefore, there is no control hazard in our instruction set.

Where data hazard can be divided into three kinds.

1. write after write, WRW
2. read after write. RRW
3. write after read, WRR

The two former are called fake hazard. According to the book, *Computer Architecture, a Quantitative Approach*, we can solve it by using register renaming. But we design the accelerator in order to simplify hardware design. However, register renaming need hardware support, which increases the complexity of hardware design. So we pause the pipeline to deal with data hazard.

Of course, pausing the pipeline will affect the efficiency of instruction execution, therefore, we increase the efficiency by adjusting the instruction window, that's to say the number of instruction fetch, which asks to achieve out-of-order instruction execution. We put forward the pipeline structure based on this requirement. The picture of data instruction stream is the following.



The simulator has four-stage pipeline, the buffer in each stage is used for simulating hardware execution cycle-by-cycle. Here are the four stages.

1. Instruction Fetch Cycle (IF)

   This stage consist of instruction fetch from instruction file and saving it to the memory. Firstly, judge whether the instruction fetch success query is empty or not, if it is empty, then fetch instructions from instruction buffer query and put them to the instruction fetch success query, and put the rest of instructions into instruction fetch failure query. Secondly, judge whether the instruction fetch failure query is empty or not, if it is not empty, then

fetch all the instructions from instruction fetch failure query and put them to the instruction buffer. Finally, judge whether the instruction buffer is empty or not, if it is empty, then fetch instructions from instruction file and fill up the query.

2. Instruction Decode Cycle (ID)

This stage consists of instruction decoding and out-of-order execution. Our simulator is based on special hardware structure, therefore, there are a lot of different rules from general OS solutions of pipeline hazard in out-of-order execution strategies. Generally speaking, we have the following rules supporting out-of-order execution.

a. Coding different instructions into different number of bytes. Decoding at most five bytes instruction code in one decoding cycle.

b. DMA instruction can only execute one by one.

c. Neural calculation or mathematical calculation instruction only one can be decoded successfully in one decoding cycle.

d. Memory access instruction only two can be decoded successfully in one decoding cycle and they are needed being different instruction types.

e. Not only satisfying the upper conditions, but also no data hazard can instructions decoding be successfully.

f. Release the occupied register after reading operand and acc buffer in ALU finishing in the first execution cycle. Release the register after writing operation finishing executing.

Out-of-order execution and hardware design are supported by the upper rules, so that we can simulate the real hardware structure.

The main process of instruction decoding is shown as the following. Firstly, judge whether the instruction decoding failure query is empty or not, if it is not empty, then fetch instructions to the instruction buffer query. Secondly, fetch instructions from instruction buffer query and judge on the rules above whether they can be decoded successfully or not. If it is successful, then put the instruction into instruction decoding success query, if not, put it into instruction decoding failure query.

3. Execute Cycle (EX)

This stage is used for instruction execution and modifying related status. Firstly, fetch instructions in instruction buffer query, and put them into corresponding instruction execution query according to the operation code. The length of instruction execution query is determined by the different execution cycle of different instruction. In the meantime,

judge whether the rear of instruction execution query is empty or not, if it is not empty, then execute the operation using related ALU, then release the destination operand of the instruction. If the front of instruction execution query is not empty, then read the operand of instruction and release the related status bite.

To cycle-by-cycle simulate the hardware, the instructions in query will be post shifted one bite per cycle.

4. Write Back Cycle (WB)

This stage is used for simulating the destination operand written back to physical memory. The main operation is to judge whether the rear of instruction execution query is empty or not, if it is not empty, then fetch the instructions, finish the destination operand writing back according to the character of instruction, and release the occupied destination operand. If not, no execution operation will be done.

5. Synchronization Cycle

We need to save the data in every stages temporarily in every cycle in order to simulate the hardware pipeline. And push the data in buffer to the next stage and change the status. All of these operations are collected in one cycle named synchronization Cycle, which ensures the connection between every stages in the pipeline.

The Function of Pulsesim Source Code

The main.cpp is the entrance of this program. Finish the initialization of configuration parameters and data, then call the function sim() which is the simulation mode. Write back the data after finishing it.

The option.cpp is used for configuring the program parameters.

The configuration.cpp is used for configuring the project parameters.

The bandwith_monitor.cpp, the data_reuse.cpp, the print_bandwith.cpp, the operation_type.cpp are used for statisitics and outputting the performance data.

The memModeling.h, the dataTyoeModeling.h, the insModeling.h, the localBufferModeling.h, the regsModeling.h are used for modeling hardware structure, data and operation.

The convb.cpp, the print_ins_to_binary.cpp are used for outputting binary code.

The print_mat.cpp, the print.cpp, the printR.cpp, the printINAddr.cpp are used for printing debug information.

The pl_kernel.cpp, the ps_pipeline.cpp are the core of pulsesim and pipeline。

The reportall.cpp, the reportString.h are used for printing the comprehensive performance data of simulator.

The pu.cpp is used for simulating instruction operation.

The wintolinux.cpp, the wintolinux.h are used for realizing compatibility between windows and linux.

Pulsesim Core Module

Data fetch Module

Firstly, this stage need to fetch the data of decoding failure instructions in the last cycle. Secondly, fetch the data from memory according to the size of instruction window and fill it up. Furthermore, set flags such as data fetch buffer query ready and the number of transfer instructions.

Decode Module

Decoding stage is the most complex module in the pipeline. To pause the pipeline, we have to set related flags. What's more, we have to solve data hazard and structure hazard to support out-of-order execution. To simulate the hardware pipeline (Five bytes data of every instructions will be fetched in one cycle.), we have to count the number of bytes of instructions in decoding stage to support special instruction out-of-order execution.

Execute Module

Linear function and nonlinear function in the instruction set are accomplished by language related function and LUT.

Submit Module

We set an instruction buffer for every instruction types in our simulator. The occupied resources will be released but not written back in execution stage.

Write-back Module

The occupied resources will be released completely and written back after the execution in the case of no hazard.

Limitation of Pulsesim

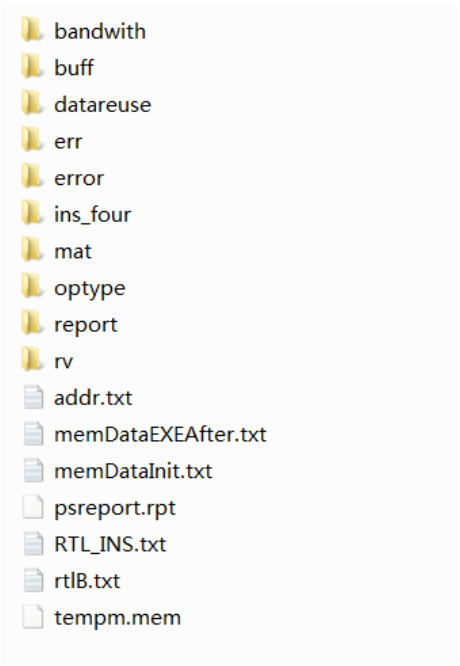
Although we can verify the hardware function, the error maybe come from your study, and the accuracy is controlled by yourself.

Case Analysis

Simulated cycle-by-cycle hardware by our simulator, users can get more detailed information.

The inst.txt document included in the code file is the CNN program code which we can achieve recognizing the handwriting number. Then we analyze the data based on this program.

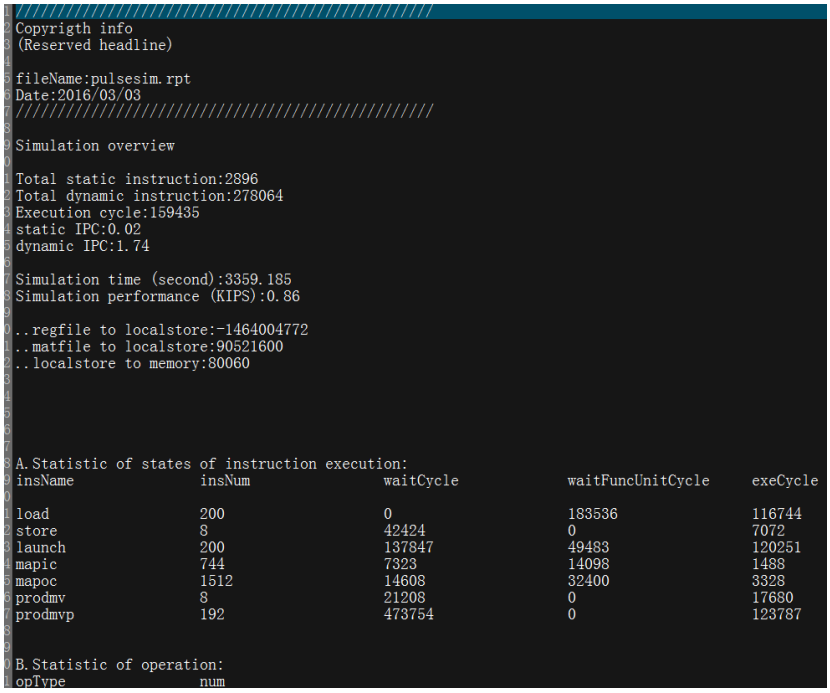
Under the normal mode, we get the file structure is shown in the following.



The psreport.rpt is used to print the information of helping to debug the simulator.

The RTL_INS.txt and rtlB.txt print the order of instruction after disorder executing and the binary code.

Report file include two document. The pulsesim.rpt is shown as following:



```
//////////////////////////////////////////////
Copyrigth info
(Reserved headline)

fileName:pulsesim.rpt
Date:2016/03/03
//////////////////////////////////////////////

Simulation overview

Total static instruction:2896
Total dynamic instruction:278064
Execution cycle:159435
static IPC:0.02
dynamic IPC:1.74

Simulation time (second):3359.185
Simulation performance (KIPS):0.86

..regfile to localstore:-1464004772
..matfile to localstore:90521600
..localstore to memory:80060




A.Statistic of states of instruction execution:
insName              insNum               waitCycle            waitFuncUnitCycle    exeCycle

load                 200                  0                    183536               116744
store                8                    42424                0                    7072
launch               200                  137847               49483                120251
mapic                744                  7323                 14098                1488
mapoc                1512                 14608                32400                3328
prodmv               8                    21208                0                    17680
prodmvp              192                  473754               0                    123787


B.Statistic of operation:
opType               num
```

The pulsesim.rpt is used to statistic the performance data of instruction code and simulator. We can find out the main point effecting the execution of progaram by this document, and optimize the instruction design. As shown in the picture, we can notice that the penalty of calculation instrucuton prodmvp is longer, we can analyze the penalty of memory of prodmvp instruction and the execution of instructions such as prodmv and so on. So that we can give a excisional explain and optimal the penalty of prodmvp.

Bandwidth of every part of hardware and the data of memory accession of instruction program are also included in this document.