International Journal of Applied Mathematics and Computing Science. Volume 3, Issue 4, (2014) 44-51
http://www.cayley-nielson.com/ijamcs

Controller Design of 4G Communications Processor Architecture

Cheng Gang
State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan 430072, China
E-mail: *chengang88@whu.edu.cn*

Abstract

With the widespread popularity of wireless mobile devices, the demand of emerging applications has been proposed, such as video telephony and HD video play. The new applications require the wireless mobile devices equipped with processors have better performance and lower power consumption. The current generation of devices employs a combination of general-purpose processors, digital signal processors, and hardwired accelerators to provide giga-operations-per-second performance on milliWatt power budgets [1]. Such heterogeneous organizations are inefficient to build and maintain, meanwhile, it will course waste of silicon chip area and power. The next generation of wireless mobile computing devices needs to have higher data transfer rate, more complex algorithms, as well as low power consumption. At this stage, mobile devices are unable to meet demand in the computing performance and power consumption. It necessarily designs the next-generation processors for mobile devices to meet the application requirements. This paper gives out a variable-width SIMD processor architecture's controller and data transfer module designing method.

Keywords: SIMD; Controller; Data Parallel; Processor Architecture; AnySP; Data Transfer Module.

1. Introduction

For 4G wireless communication, a lot of work has been done in the research of designing high-performance and low-power processors, such as SODA [2], Ardbeg [3], AnySP [1], etc. The controller design mainly focuses on improving the computing performance and versatility of processor, and balancing power consumption. The major way to improve computing performance is to effectively ensure the data-level parallelism. With the emergence of more and more applications, more sophisticated algorithms will be implemented in wireless mobile devices. The natural lengths of these algorithms are different, so it is necessarily provide a variable-width of the SIMD processor architecture to adapt the natural length of a variety of algorithms. On the observation of many embedded applications, 25% of the registers occupy 75-92% of the entire register file access [4]. Therefore, to select the most frequently used registers and put those in a relatively smaller power register file will reduce the power consumption of the entire register file.

On the basis of the pioneering research, the proposed architecture is shown in Fig. 1. Not only does the controller control the program flow, but also transfer PE (process element) instructions for PE decoders and initialize the data transfer module. Firstly, global memory's data is transferred by data transfer module into the interface module, and then stored into the local memory of each PE. At the same time, the PE instructions which are passed by controller are buffered into the buffer register file. When the transferring work has been accomplished, PE instructions are issued from the register file to the decoder of each PE.

ISSN 2333-2425
December, 2014

As described above, the controller plays an important role in the architecture. In this paper, take three-stage pipeline to achieve the controller design. Although pipelining can improve the throughput of the controller, it needs to address the problems of branch and data hazard. In summary, the structure of three-stage pipeline can effectively reduce the complexity of the controller design and can meet the design requirements better.

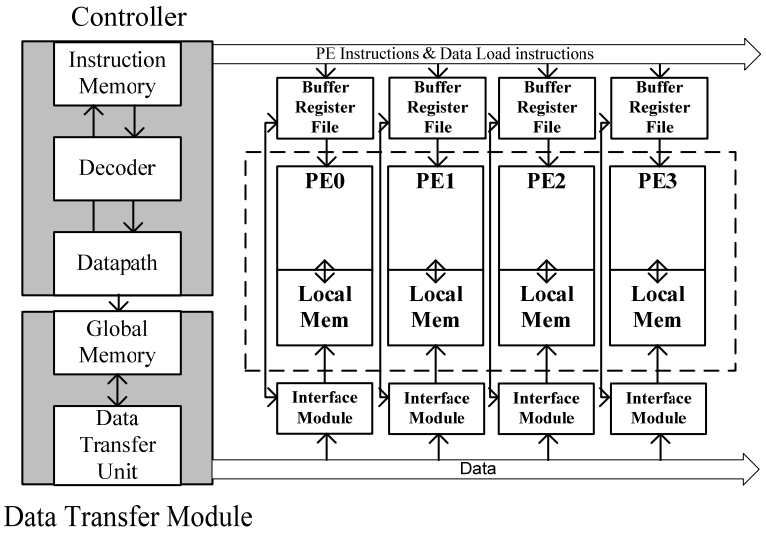

Figure 1. The architecture diagram

Each PE consists of 8 local memory banks, 8 register files, 64 groups of ALU + Multiplier + Adder + Loader (AMAL), an adder tree and a decoder. The work of decoding PE instructions is not completed in controller but in the decoder of each PE. Data transfer module passes the data into the 8 local memory banks. Load instructions stores local memory's data into the specified entry of 8 register files or partial of them. The PE decoder decodes the PE instructions, and then generates control signals for AMAL, adder tree, etc.

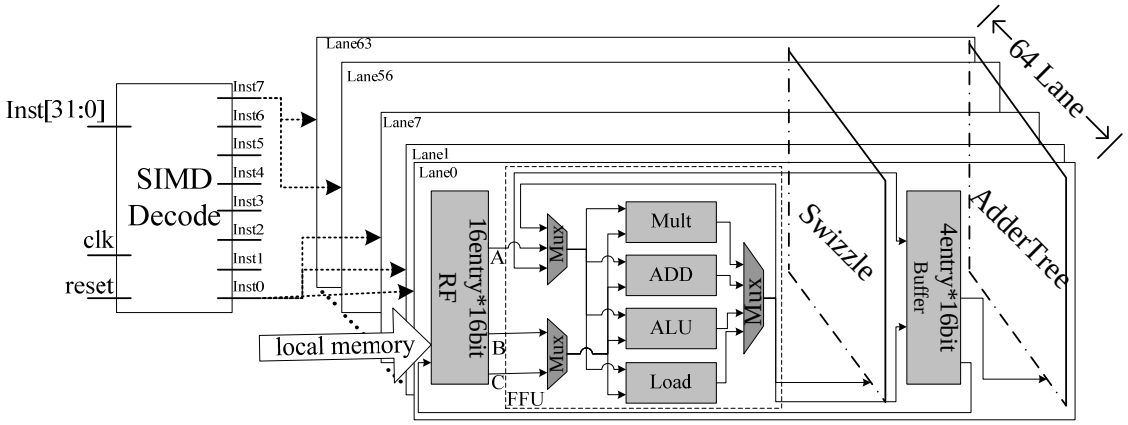

Figure 2. The PE architecture diagram

The paper is organized as follows. Section Ⅱ describes the instruction sets of the scalar and data transfer. Section Ⅲ explains the controller design. Section Ⅳ introduces the module and interface that assist the controller to complete the data transfer. Finally, a conclusion is given in Section V.

2. Scalar and Data Transmission Instruction Set

In the proposed architecture, the instruction set consists of three parts: scalar, PE, and data transfer instruction sets. This article discusses the controller design, and the PE instructions are not introduced. As follows is the scalar and data transfer instruction sets:

Table 1. Scalar and data transfer instruction sets

Table 1：	
Table 1：	
Table 1：	
Table 1：	
Table 1：	
Table 1：	Data Transfer Instruction
Table 1：	Single-entry data loaded instruction
Table 1：	2-entry data loaded instruction
Table 1：	4-entry data loaded instruction
Table 1：	8-entry data loaded instruction
Table 1：	

The proposed instruction set uses 32-bit instructions. The instruction format of arithmetic and logic instruction is similar to the MIPS instruction set's R-type and I-type. Those instructions complete the arithmetic and logic calculations for two registers, or an immediate and a register. To sequentially execute instructions, the program counters increments by 4 after each instruction. The Control instructions modify the program counter to skip over sections of code or to go back to repeat previous code. In the data transfer/load instructions, the MOV instruction is decoded in the controller's decoder. The produced signals are used to initialize the data transfer module. The four load instructions are decode in the interface module's load decoder. They focus on how to load data from PE's local memory to PE's register file.

3. Controller Design

The major work of the controller is controlling the program flow, initializing the data transfer module, identifying the PE & Load instruction and passing them to the interface module's buffer register file. It consists of datapath, decoder, and instruction memory. In order to improve the throughput of the overall structure, design the dedicated data transfer module to assist the controller to complete data transfer from the global memory to the local memory of each PE.

(1) Decoder

The decoder primarily decodes the scalar and data transfer instructions. All kinds of instructions are stored in the instruction memory. In the Fetch stage, an instruction is fetched from the instruction memory. It is passed to the controller's decoder in the Decode stage. If it is a scalar or data transfer instruction, controller's decoder gives the corresponding control signal for datapath or data transfer module. But, when controller's decoder encounters a PE instruction or a Load instruction, it does not execute any operations

and only passes the instruction to the interface between the data transfer module and PEs. The interface is responsible for issuing the PE instruction to PE decoder at a particular time and reading/storing data for each PE's local memory banks.

(2) Datapath

The datapath is mainly composed by an arithmetic logic unit, a comparison unit, and a register file. The wireless communication and high-definition video algorithms are implemented in the PE array. Therefore, the arithmetic logic unit of controller completes simple arithmetic operations, logic operations and shift operations. Comparing operation is completed in the comparison unit. The register file is composed of 30 general purpose registers r1 to r30, a constant zero register r0, and a specially used to store the return address register r31.

(3) PC Logic

PC Logic completes the selection of the next instruction by two 2-way multiplexers, a register, and an added. As shown in Fig. 3, two multiplexers are distinguished as A and B from left to right. The A is used to select whether to perform branch. The B is used to determine whether to execute jump. Address of the next instruction is selected through the two multiplexers A and B. On clock rising edge, the address is passed to the instruction memory to fetch the instruction. At the same time, pc is word aligned, so the sum of pc and 4 is passed to the multiplexer A for the next instruction selection. But when it encounters a jump instruction, the jump destination address pcjD and control signal jumpD are generated in the Decode stage and later passed into the Fetch stage. The jump instruction does not need to determine condition, so it's work can be achieved in the Decode stage; if it is a branch instruction, the branch destination address pcbE has been generated in the Decode stage, but the control signal bE is generated in the Execute stage. Because the branch instruction needs to determine condition in the Execute stage.

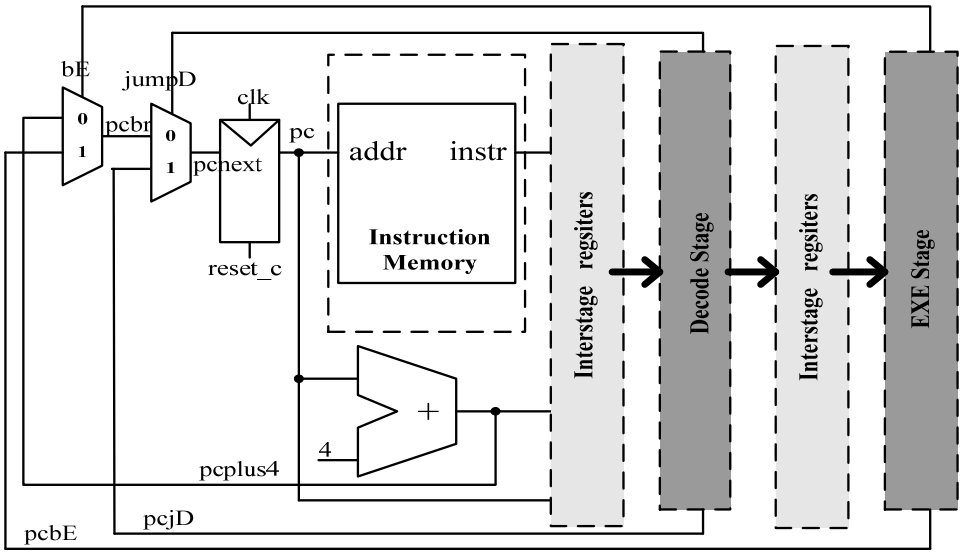

Figure 3. PC Logic

In the Decode stage, the control signal jumpD is generated by the decoder. JumpD is not only passed to the PC logic, but also passed to the inter-stage registers between the Fetch stage and Decode stage as a refresh signal. When jumpD is 1, it refreshes the registers to ensure that the instruction following the jump instruction does not enter the Decode stage to ensure the correct execution of the program. The control

signal bE is different to jumpD. Shown in Fig. 4, bE = (~zeroE & S15) | (S9 & cpresultE [0]), which selects the next instruction address from the pc + 4 and branch address. The S15 and S9 control two kind of branches. The S11 and S12 tell the arithmetic logic unit and comparison unit to do what. When a branch instruction is in the Execute stage, the Decode stage and Fetch stage have separately entered an instruction. As similar to the jump instruction, the inter-stage registers are refreshed by the bE.

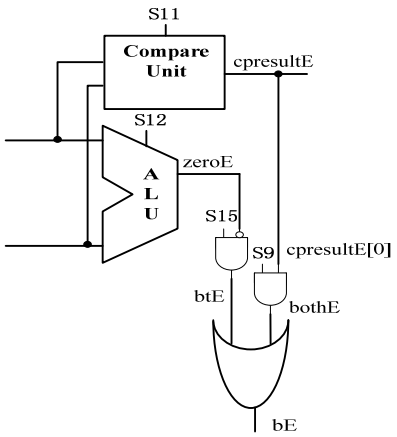

Figure 4. Branch Control Signal Generating Logic

The major work of the address generation unit is to produce the next instruction address for the branch and jump instruction. The jump destination address pcjD and the branch destination address pcbD are produced by the address generation unit in the Decode stage. The pcjD is sent to the Fetch stage in Decode stage, but the pcbD is passed in Execute stage.

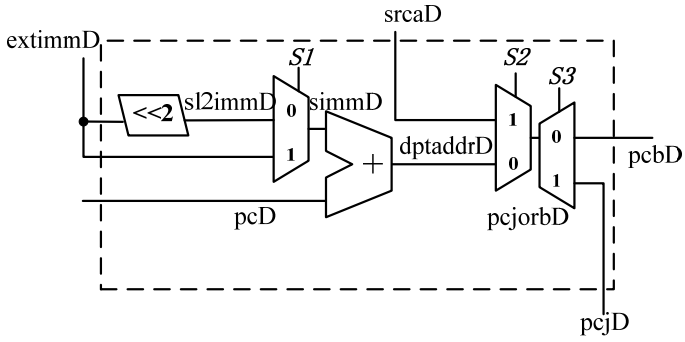

Figure 5. Address Generation Unit

Shown in Fig. 4, the address generation unit consists of a shifter, three 2-way multiplexers, and an adder. The control signals S1, S2, S3 are generated by the decoder in the Decode stage. The shifter's operation is that the extimmD is left-shifted by 2 to multiply it by 4. After the S1 controls 2-way multiplexer to choose the simmD from extimmD and extimmD × 4, then simmD is passed to the adder to calculate the sum of pcD and simmD, which is the destination address of the branch or jump. The S2 controls the multiplexer to support the JLINK instruction. The S3 controls anti-2-way multiplexer for jump and branch instruction to pass the address to pcjD or pcbD.

(4) Hazards

When one instruction is dependent on the results of another that has not yet completed a hazard occurs [5]. For solving this problem，When the instruction will write back result into the register file in the Execute stage, it compares the wrote register and the read register of following instruction. When the hazard has been occurred, the result is forwarded from the execute stage of the previous instruction to the decode stage of the dependent next instruction. In the Execute stage, there is a special hazard unit to process this matter.

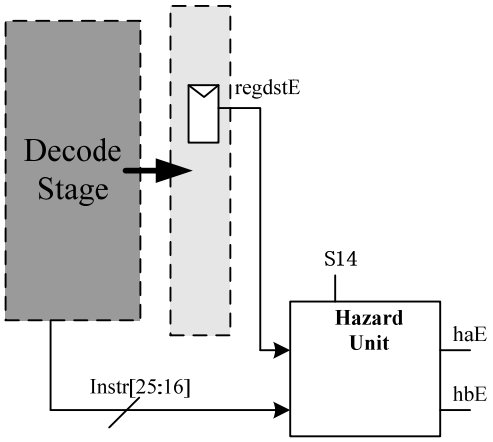

Figure 6. Data Hazard Unit

Shown in Fig. 5, when an instruction writes back the result into the register file, the control signal S14 is 1, the data hazard unit solves data hazard with forwarding. Otherwise the unit will be bypassed. The two operands of hazard unit are from the decode stage and execute stage. The Decode stage passes the identity of read register and the Execute stage passes the identity of the register wrote. The signal haE and hbE choose the forwarded value as the operand for the next instruction.

4. Data Transfer

To effectively support the data-level parallelism, it must timely pass data for the SIMD array. Design data transfer module and interface module to support this work.

(1) Data Transfer Module

Without the assistance of controller, the data transfer module independently transfers bulk data to the local memory banks of each PE. When the controller fetches the data transfer instruction in the Fetch stage, it will pass the initialization information to the data transfer module in the Decode stage. Then the controller continues to perform the program, and the module starts transferring data for each PE with the initialization information. The controller and the data transfer module execute in parallel. With assisting of the data transfer module, the workload of controller has been reduced, and it also can timely transfer data for each PE.

The Mov instruction format is as follow:

Table 2. Mov instruction format

31 26	25 22	21 16	15 0
opcode	bank_index	count	initial_address
010010			

The operation is determined solely by the opcode, and the Mov instruction's opcode field is 0100102. bank_index is the index of each PE's local memory bank which will receive data. The number of transferring data is the count field. The initial_address field indicates the first reading data address in global memory. For example, bank_index, count, and initial_address fields are 00012, 0000112, and 0(00000000000000002), respectively. The data transfer module will transfer three 128bit numbers to each PE's local memory bank1. In every cycle, the data transfer module only transfers one 128bit number and 1bit write-enable signal to each PE's special bank. So when once transfer operation has been finished, determine the result of count – 1 is greater than 0. If the result is greater, data transfer work continues to do and vice versa.

b) Interface Module

Sometimes the data transfer module and controller execute in parallel. When the data transfer module is working, the PE or Load instructions passed from the controller are buffered into the buffer register file of interface module. After data transfer work has finished, the buffered instructions are fetched from the buffer register file and passed to the load decoder of interface module. Then the load decoder identifies the past instruction. If it is a load instruction, the load decoder executes decoding operation; if it is a PE instruction, the load decoder only passes it into the PE decoder. But if there is not any data transfer, PE instructions does not need to be buffered into the register file and simply passed into the PE decoder.

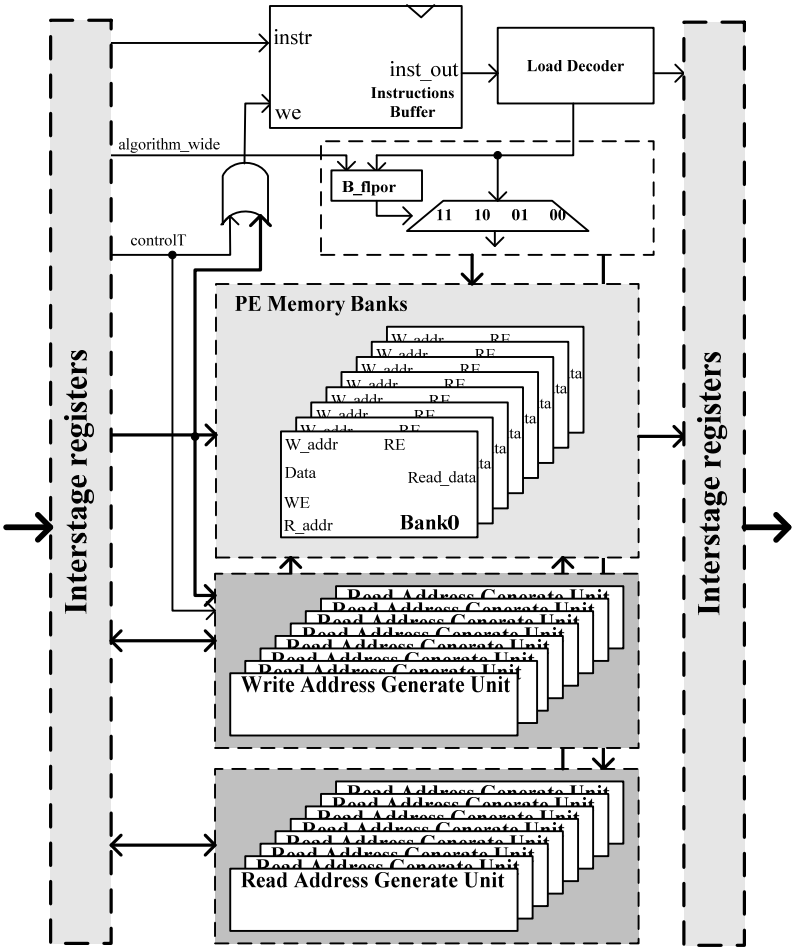

Figure 7. PE0 Interface Module

As the name suggests the load decoder focuses on the operation of decoding load instructions. When the load decoder identifies the instruction which is fetched from the buffer register file is a load instruction, it products a read-enable signal. With the assistance of the algorithm width signal, it controls the reading operation of PE local memory Bank0~Bank7. Because most of the wireless communications and high-definition video algorithms just need streaming data, reading data is sequential. The reading and writing address of each bank are produced sequentially. The load instruction based on the algorithm width signal can complete the loading operation by the following modes: single-entry data loading, 2-entry data loading, 4-entry data loading, and 8-entry data loading.

In addition, when the instruction is not a load instruction, the load decoder does not perform any operation and simply passes it into the decoder of each PE.

5. Conclusion

In this paper, a design of the controller and the data transfer module is presented. The controller designing focused on controlling the program flow and initializing the data transfer module. The work of decoding PE instructions is completed in the PE decoder. The data transfer module executed the work of data transfer without the assist of controller. By simulating the architecture on Xilinx ISE Design Suite, the fundamental modules have been designed and tested. The results of simulation verify the correctness of the controller and data transfer module design.

References

[1] Woh, Mark, et al. "AnySP: anytime anywhere anyway signal processing." IEEE micro 30.1 (2010): 81-91.

[2] Lin, Yuan, et al. "Soda: A low-power architecture for software radio." ACM SIGARCH Computer Architecture News. Vol. 34. No. 2. IEEE Computer Society, 2006.

[3] Woh, Mark, et al. "From SODA to scotch: The evolution of a wireless baseband processor." Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on. IEEE, 2008.

[4] Kozyrakis, Christoforos E., and David A. Patterson. "Scalable, vector processors for embedded systems." Micro, IEEE 23.6 (2003): 36-45.

[5] Harris, David Money, and Sarah L. Harris. Digital design and computer architecture. Elsevier, 2013.