A Superscalar Dual-Core Architecture for ARM ISA

Jih-Ching Chiu, Yu-Liang Chou and Po-Kai Chen
Department of Electrical Engineering,
National Sun Yat-Sen University, 70 Lien-hai Rd, Kaohsiung 804, Taiwan.
d943010010@student.nsysu.edu.tw

ABSTRACT

*Recently, embedded systems increase the complexities of the applications and system designers must choose more powerful embedded processors to meet the complex applications. For keeping the system life time and saving the software reengineering cost, it is important for the designers of computer architectures to increase the performances of the instruction set architecture family. Consequently, this paper proposes a **S**uperscalar **D**ual-**C**ore (SDC) architecture consisting of two five-stage pipeline ARM processor cores and an instruction dispatched unit (which can fetch two instructions at once and dispatch the two instructions to both processor cores by proposed dispatched rules) to make the high instruction set compatibility and to support higher performance requirements.*

This paper presents the philosophy of the SDC architecture, and the structure of SDC programs. The paper also discusses performance issues in the SDC architecture, and compares it with the five-stage pipeline architecture. According to experimental results, the SDC architecture can obtain average 41% performance speedup comparing to the five-stage pipeline architecture.

1: INTRUDUCTIONS

Since embedded systems increase the complexity of the software and system designers must have to choose more powerful embedded processors to meet the complex applications. At the same time, low power is still a key feature for embedded systems. In such a situation, chip-multiprocessor (CMP) becomes a key method to be adopted to deliver high throughput without simply increasing the clock frequency. On the other hand, how to keep the system life time and save the software reengineering costs are also important features for embedded system designs. Current CMP designs are almost based on symmetric multiprocessing (SMP). Some of current CMPs like IBM POWER5 [1], Athlon64 X2, and Pentium D include two superscalar processor cores on one physical chip. OS or applications that has been designed or optimized for the CMPs will receive a noticeable performance boost.

However, these CMPs are not available for embedded systems because of their large hardware cost and high clock frequency lead to high power consumption. Some of other CMPs [2, 3] are designed for embedded systems. They employ simple embedded processors in consideration of low power and low hardware costs. Nevertheless, while executing single-thread applications, they have poor performance because of the simple processor core.

This paper proposes a **S**uperscalar **D**ual-**C**ore (**SDC**) architecture that uses the statically scheduled superscalar ideal on the CMP architecture to achieve high performance, low hardware costs and less software reengineering time for the embedded systems. Section 2 presents the philosophy of the SDC architecture. The design challenges of the CMP architecture performing superscalar are explored in Section3. Section 4 shows the hardware designs of the SDC architecture. In Section 5, this paper provides and explains the code examples of the SDC architecture. A performance evaluation of the SDC architecture is given in Section 6. Section 7 summarizes this work and offers concluding remarks.

2: PHILOSOPHY OF THE SDC

Different from current CMPs, the SDC architecture is not only capable of running single-thread applications without modifications but also improve the single-thread applications executed performance. This is a key feature of the SDC architecture. Figure 1 shows the block diagram of the SDC architecture. CORE_A and CORE_B are five-stage pipeline ARM CPU cores connected with an **I**nstruction **D**ispatched **U**nit (**IDU**). The IDU handles the instruction fetched and dispatched works, program counter (PC) and status register update, and register files write back control. Two processor cores share the data memory. With a memory access arbiter, two processor cores can execute two data processing instructions simultaneously. Each processor core has its own register files. According to different operation mode, two processor cores share the CORE_A register files or use their individual register files.

To make the SDC architecture be more suitable for embedded systems and reserve the advantages of the CMP architecture, new extended instructions and three operation modes (single mode, superscalar mode, and multithreading mode) are proposed.

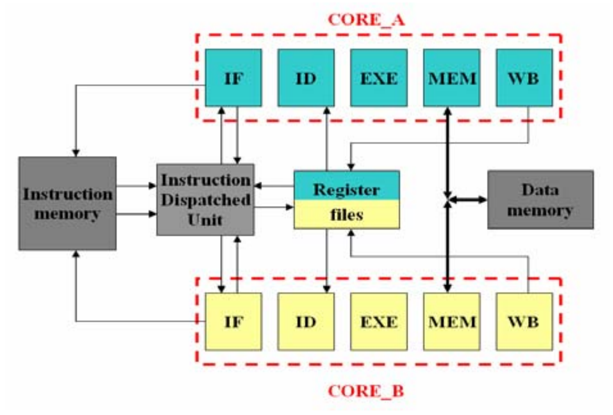


Figure 1: An Overview of the SDC Architecture.

When the SDC architecture is in superscalar mode, the IDU fetches two instructions at a time and dispatches the instructions to both cores by the dispatched rules. Two cores share the register files of CORE_A and can read or write the CORE_A registers at the same time. As two processor cores tend to write the same register, register files will accord to the IDU control singles to update the latest data of the register. Because the SDC architecture in superscalar mode acts like a Statically Scheduled Superscalar processor, designers needn't have to reengineer the original system and can easily obtain high performance.

On the other hand, low power dissipation is a requirement of embedded processors. To address the power issue, the designer can use the extended instructions to dynamically switch the SDC architecture into single mode with proposed new extended instructions to suppress the standby power of the embedded system. When the SDC architecture is in single mode, CORE_B will be stalled and now it acts like the five-stage pipeline ARM processor.

Furthermore, the real-time multi-task kernel is an important part of embedded systems. The SDC architecture provides the multithreading mode for the real-time multi-tasking OS to enhance the ability of handling the real-time task. As the real-time events occur, the SDC architecture is switched into multithreading mode. So one processor core can be used to handle the real-time task, and the other processor core can be used to handle the multi-tasking operation system.

According to different applications of embedded system, the designers can dynamically use the new extended instructions to change the operation mode of the SDC architecture and obtain the balance between the utility of the hardware and the consumption of the power.

3: DESIGN COGITATIONS OF THE SDC

When the SDC architecture is working like a superscalar machine, there will cause some problems let two instructions can't be executed simultaneously.

The first problem is the read after write (RAW) hazards between two fetched instructions. The SDC architecture connects two ARM five-stage pipelines with the IDU and redesigns the register files but doesn't change the original ARM five-stage pipeline design and each ARM five-stage pipeline still has it own forwarding unit. When two sequential instructions that have RAW hazards are executed in the SDC architecture simultaneously, some errors may occur because the SDC architecture doesn't have the forwarding paths between two processor cores. In this situation, only one instruction can be dispatched and the two instructions will be dispatched to the same processor core so the RAW hazards will be resolved by the forwarding unit of the processor core.

The second problem is the RAW hazards between the fetched instructions and the instructions being executed in two pipelines. In Figure 2, I2 and I1 have RAW hazard with R1. I2 and I0 also have RAW hazard with R2.But I0 and I1 don't have any RAW hazard. In this situation, the correct data of I0 and I1 can be forwarded to I2 in the five-stage pipeline architecture, but in the SDC architecture it will lead to some errors as show in Figure 2. The data of I1 can't be forward to I2 in the SDC architecture. In order to avoid this hazard, the IDU will check the RAW hazards between the new fetched instruction and the executing instruction in two processor cores. If the IDU find that the new fetched instruction both have dependencies with the instructions in CORE_A and the instructions in CORE_B, the IDU will first stop to fetch the new instruction. Second, dispatch NOP instruction to two processor cores until the hazards don't exit.

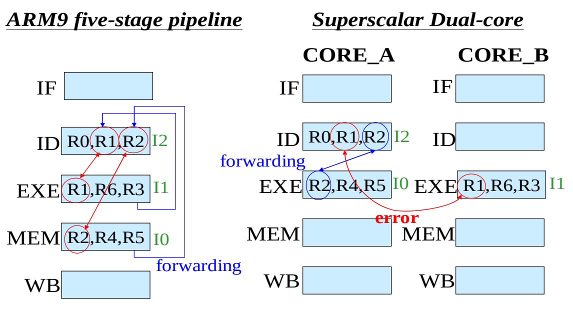


Figure 2: RAW Hazards of the SDC Architecture.

The third problem is the control hazard causing the pipeline flushed. In the five-stage pipeline architecture, a common improvement over branch stalling is the predict-not-taken stratagem. In the SDC architecture, if the predict-not-taken stratagem is also adopted, the instructions that are being fetched and decoded in the other core will can't be flushed. So when the IDU fetches a control flow instruction, the IDU will dispatch the instruction to CORE_A and stop fetch new instruction until the Control flow instruction is dry.

The forth problem is the conditionally executed instructions. An unusual feature of the ARM instruction set is that every instruction is conditionally executed. The latch between ID stage and EXE stage will checks the status register to decide whether the instruction be executed or not. In the SDC architecture, if two instructions, I0 will change the status register and I1 is a conditionally executed instruction, are executed at the same time, the status register values may be old and invalid for I1 to check. In the SDC architecture, the two fetched instructions, one will change the status register and the other is a conditionally executed instruction (its condition code is not AL), can't be executed in parallel.

The final problem is the out of order execution problem. If one processor core is stalled by data memory accessing and the IDU continuously dispatches instructions to another core, it may cause WAW and WAR hazards. So before the IDU dispatches instructions to the non-stalled core, it will check the WAW and WAR hazards between the fetched instructions and the executing instructions be stalled in another processor core.

4: HARDWARE DESIGNS OF THE SDC

To reduce the processor redesign costs, the SDC architecture doesn't change the already designed five-stage pipeline architecture and all of its control signals. It simply connects the existing control signals of the two cores. The main design of the SDC architecture is the **I**nstruction **D**ispatched **U**nit (**IDU**). The IDU handles the instructions fetched jobs and connects the fetch stage of both cores with the instruction memory. The control signals between fetch stage and instruction memory now are all connected to the IDU.

4.1: DESIGNS OF THE IDU

The IDU handles three main works. The first work is the three operation modes, single and superscalar and multithreading mode, switch controls. The second work is to pre-decode the new extended instructions. The third work is to dispatch the fetched instructions to both processor cores by the dispatched rules.

New extended instruction	Use	Action
suprs	System instruction	Let superscalar dual-core architecture enter superscalar mode。
single	User instruction	Let superscalar dual-core architecture enter single mode。
mthd	User instruction	Let superscalar dual-core architecture enter multithreading mode。
joint	User instruction	The instructions, joint and wait, are a pair of rendezvous instructions. Designer can use the two instructions to synchronize two threads.
wait	User instruction	
move Rd ,Rn	User Instruction	Store the register Rn of CORE_A to the register Rd of CORE_B

Table 1: New Extend Instructions

Table.1 shows the purposed new instructions. In the remains of this paper, the word in italic and bold type is used to represent the new extend instruction. The ***suprs*** is a system instruction and it can be used only when processor core is in the system mode. The ***joint*** and ***wait*** are pair of rendezvous instructions. When the SDC architecture is in the multithreading mode, the designer can use these two instructions to synchronize two being executed threads. The instruction, ***move Rd ,Rn*** , is an communicating instruction. It can move the register values of CORE_A to the registers of CORE_B.

The using examples of these new extended instructions will show in Section 5. Continuously, how the IDU handles the operation mode change and pre-decodes new extended instructions will be illustrated. Figure 4 shows the relations between new extended instructions and operation modes.

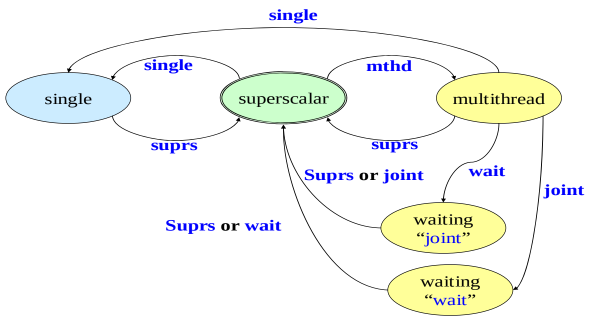


Figure 4: Relations between New Extended Instructions and Operation Modes

When the SDC architecture is reset, it enters the superscalar mode and only ***single ,mthd*** and ***move Rd,Rn*** are valid. The signal connection is showed in Figure 5. There are four steps to fetch instructions. First, the IDU will read PC from CORE_A register files. Second, bypass the PC value to the fetch stage of CORE_A and bypass the PC+4 value to CORE_B fetch stage. Third, the corresponding fetched instructions form instruction memory will be send to the IDU, and the IDU will dispatch these two instructions to both cores according to the dispatched rule. Finally, the IDU will update the correct PC value return to CORE_A register files.

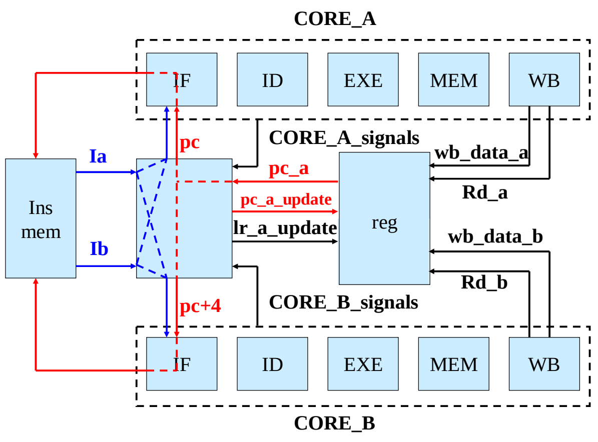


Figure 5: Signal Connections of the SDC in Superscalar Mode

When the IDU fetches the ***single*** instruction or the ***mthd***, it will stall fetch stage and wait until the instructions in both processor cores be finished and then change the SDC architecture into single mode or multithreading mode. When the ***move Rd , Rn*** is fetched, the IDU will re-encode the instruction into mov Rd, Rn (the ARM instruction) and dispatch it to CORE_B. Because the SDC architecture now is in superscalar mode, two processor cores share the register files of CORE_A.

When the mov Rd, Rn is decoded, the value of Rn is read form CORE_A register files. When the instruction enter write back stage, the IDU will control the write back enable single let the value be wrote back to CORE_B register files.

In single mode, only the ***suprs*** is valid. When the IDU fetches the ***suprs***, it will wait the instructions in CORE_A be finished and than change the mode into superscalar mode.

In multithreading mode, only the ***wait, joint, single*** and ***suprs*** instructions are valid and the SDC architecture acts like as two independent five-stage pipeline architectures

4.2: THE RE-ENCODE DESIGN OF THE NEW EXTENDED INSTRUCTIONS

Although six new extended instructions are created, but the decode stage of each processor core don't have to be re-designed. The new extended instructions will be decoded previously in the IDU. When the IDU gets the information form the new extended instructions, it will re-encode the new extended instructions into the ARM instructions.

4.3: INSTRUCTION DISPATCHED RULES

In superscalar mode, there are four conditions for the IDU to handle the instruction fetched works.

First condition is CORE_A pipeline stalled: In this situation, the IDU will fetch one instruction. If the fetched instruction doesn't have RAW, WAW, WAR hazards with the instructions in CORE_A pipeline, the fetch instruction will be dispatched to CORE_B pipeline. Second, CORE_B pipeline is stalled: In this situation, the IDU will fetch one instruction. If the fetched instruction doesn't have RAW, WAW, WAR hazards with the instructions in CORE_B pipeline, the fetch instruction will be dispatched to CORE_A pipeline. Third, both pipelines are stalled: In this situation, the IDU doesn't fetch any instructions form instruction memory. Fourth, both pipelines are not stalled: In this situation, the IDU will fetch two instructions. According to the dispatched rules, the IDU will dispatch two, one or zero instruction to both cores.

Before showing the dispatched rules, the instructions are classified into more detail categories for the dispatched rules.

Type0: Data processing instructions.
Type1: Single register load and store instructions.
Type2: Multiple register load and store instructions.
Type3: Control flow instructions, undefined instructions , and the instruction that its condition code is not AL.
Type4: SWP, MRS, MSR, and other ARM instructions.

To issue two instructions simultaneously the fetched instructions must satisfy the following dispatched rules: (I0 is the instruction fetched from PC and I1 is the instruction fetched from PC+4)

1. I0 and I1 must be type0 or type1 instruction.
2. There must be no RAW or WAW hazards between them
3. If I0 will change status register then condition code of I1 must be AL.
4. RAW hazards satisfy one the three conditions
   a. If there are no RAW hazards between I0 and instructions in both CORE_A and CORE_B then there must be no RAW hazards between I1 and instructions in both CORE_A and CORE_B.
   b. If there are RAW hazards only between I0 and instructions in CORE_A then there must be no RAW hazards between I1 and instructions in CORE_A. (In this situation, I0 will be dispatched to CORE_A and I1 will be dispatch to CORE_B)
   c. If there are RAW hazards only between I0 and instructions in CORE_B then there must be no RAW hazards between I1 and instructions in CORE_B. (In this situation, I0 will be dispatched to CORE_B and I1 will be dispatch to CORE_A)

5: PROGRAMS OF THE SDC

The SDC architecture provides new extended instructions for programmers to perform tow different type multithreading, intra-program multithreading and inter- program multithreading.

5.1: INTRA-PROGRAM MULTITHREADING

When executing a single thread task, the designer can divide this task into two parallel sub-threads. The designer can use ***wait, joint, mthd*** and ***move Rd,Rn*** to construct an inter–program multithreading program. For example, if the designer wants to sort 20 non-sequential records then he can create thread_a to sort first ten non-sequential records and thread_b to sort the other non-sequential records. Figure 6 shows an intra-program multithreading program example of the sort scheme.

```
1.Initialize          ldr    r0, =array_1      sort:
  thread_a            ldr    r1, =num_1           stmfd     sp!, {r0-r14, lr}
                      ldr    sp,=stack_a          …
                      ldr    r2, =array_2         …
2.Initialize          ldr    r3, =num_2           ldmfd     sp!, {r0-r14, pc}
  thread_b            ldr    r4, =stack_b
                      ldr    r5, =tread_end    merge_sort:
                      ldr    r6, =sort            stmfd     sp!, {r0-r14, lr}
3.Move the            move   r0, r2               …
initialized values    move   r1, r3               …
to register files     move   sp, r4               ldmfd     sp!, {r0-r14, pc}
of CORE_B then        move   lr, r5
enter Multi-          move   pc, r6              thread_end:
threading  mode       mthd                                joint
                      bl     sort
4.Put the wait at     wait                              5.Put the joint at the
  the return                                              end address of
  address of          …                                   the thread_b
 the thread_a         bl      merge_sort
```

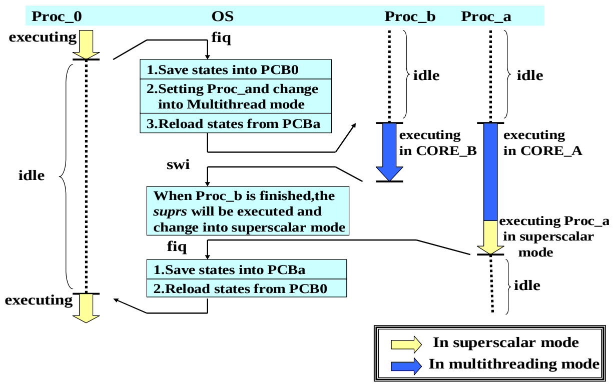
Figure 6: An Intra-program Multithreading Program Example

The intra-program multithreading programs are constructed by the following steps. First, initialize the arguments of thread_a. Second, initial the arguments of thread_b. Notice that r5 is stored the return address of thread_b and r6 is stored the start address of thread_b. Third, use the ***move Rd,Rn*** to move the initialized values to the register file of CORE_B and use the ***mthd*** to enter multithread mode. Notice that the r5 will be store to the lr of CORE_B and r6 will be store to the pc of CORE_B. When the ***mthd*** is executed, the SDC architecture will enter multithreading mode and now each core fetches its own instructions according to its own pc. Because the pc of CORE_B is stored the start address of sort scheme, CORE_B now will start to sort the corresponding records. At the same time, CORE_A fetch the "bl sort" instruction then branch to the sorting function to sort its corresponding records and the address of ***wait*** (return address after function call) will be stored to the lr of CORE_A. When CORE_A finished its work, the lr will be re-stored to pc and the ***wait*** will be fetch by CORE_A. When CORE_B finishes its work, CORE_B will branch to the thread_end tag and fetch the ***joint***, because the address of thread_end tag is stored to the lr of CORE_B in previously settings. When ***wait*** and ***joint*** are both fetched, this means that both tread of both core are finished and the SDC architecture will enter superscalar mode to perform merge_sort scheme.

5.2: INTER-PROGRAM MULTITHREADING

In the time sharing operation system, task switching is executed when interrupt or system call occur. OS will save the old process state into its PCB (process control block) and load the new process state form new process 's PCB and then start to execute the new process.

If there are two independent processes in OS, OS may execute these two processes simultaneously on the SDC architecture. Figure 7 shows the process switching of the SDC architecture. Initially, Proc_0 is executed in superscalar mode. When the fiq interrupt occurs, OS will set the environments Proc_b and reload the state of Proc_a from PCBa than enter multi threading mode and execute two processes simultaneously. If Proc_b is finished, it will generate a system call to OS and OS will save the execution results of Proc_b and change the SDC architecture into superscalar mode to execute Proc_a continuously. When Proc_a is finished, OS will save state into PCBa and reload state form PCB0 then execute Proc_0.



Proc_0
OS
Proc_b
Proc_a
executing
fiq
1.Save states into PCB0
2.Setting Proc_and change into Multithread mode
3.Reload states from PCBa
idle
idle
executing in CORE_B
executing in CORE_A
idle
swi
When Proc_b is finished,the suprs will be executed and change into superscalar mode
executing Proc_a in superscalar mode
fiq
1.Save states into PCBa
2.Reload states from PCB0
idle
executing
In superscalar mode
In multithreading mode

Figure 7:.The Process Switching of the SDC

Figure 8 shows the part of Inter-program multithreading program. In the fiq_handler, first stores state into PCB0 and second setting the environments of Proc_b then change into multithreading mode. Finally, restore the state form PCBa and now CORE_A is executing Proc_a, CORE_B is executing Proc_b. Notice that the lr of CORE_B is stored the address of thread_end tag. When Proc_b is finished, it will branch to thread_end tag and execute the SWI instruction to generate a system call for OS. The swi_handler will store the computation result of Proc_b and terminate Proc_b then finally execute the ***suprs*** to change the SDC architecture into superscalar mode.

```
                     fiq_handler:                       swi_handler:
                     1.Store state into PCB0                  suprs
                     2.Setting the environment
                       of Proc_b                        proc_a:
                                                           stmfd  sp!, {r0-r14, lr}
                     ldr    r1, =parameter1                …
Setting the          ldr    r2, =parameter2                ldmfd  sp!, {r0-r14, pc}
 Proc_b              ldr    r3, =stack_b                proc_b:
 arguments           ldr    r4, =tread_end                 stmfd  sp!, {r0-r14, lr}
                     ldr    r5, =proc_b                    …
Move the value       move   r0, r1                         ldmfd  sp!, {r0-r14, pc}
into CORE_B          move   r1, r2
register files       move   sp, r3                      thread_end:
and switch into      move   lr, r4                              SWI
multithreading       move   pc, r5
mode                 mthd
                     3.Restore the state form PCBa
```

Figure 8: Example of Inter- program Multithreading Program

6: VERIFICATION AND EVALUATION

This paper evaluates the performance of the SDC architecture in superscalar mode through trace-driven simulation methodology. The benchmark programs are compiled and linked by the ARMulator's armcc and armlink. The ARMulator(ARM emulator) is a suite of programs that models the behavior of various ARM processor cores in software on a host system. The traces of the benchmark programs are extracted by using ARMSD Tracer (armsd). The five-stage pipeline simulator and the SDC architecture in superscalar mode simulator are written for simulations.
Then the trace files are fed into the two simulators to generate the results.

The benchmark programs this paper use are from MediaBench suite [4]. The MediaBench suite is developed to accurately represent the workload of emerging multimedia and communication system. Because multimedia and communications application workloads are becoming the main part of the workload of the embedded systems, this paper adopts the MediaBench suite for benchmark programs.

Figure 9 is the simulation results of both simulators. The IPC (instructions per cycle) is used to evaluate the performance. According the simulation results of the five-stage pipeline simulator, the average IPC of the benchmark programs is 0.6 because not all of the instructions will be finished in one clock cycle. In the SDC simulator simulation results, the SDC architecture can enhance the IPC up to 0.9. Figure 10 shows the performance speedup of the SDC architecture compare to the five-stage pipeline architecture. When the SDC architecture performs the MPEG2 decoder trace, it will obtain 51% performance speedup. On average, the SDC architecture will obtain 41% performance speedup.

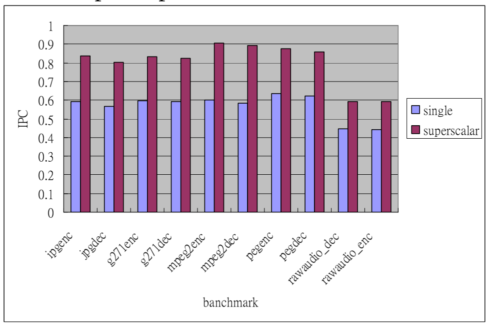


Figure 9: Simulation Results

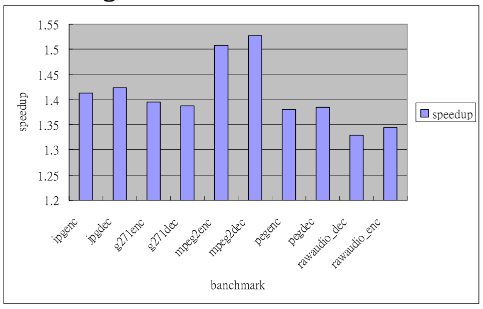


Figure 10: Performance Speedup

The SDC architecture was implemented on a 0.25um 2.5V technology and its clock is 279MHz, its core area is 4091287 um^2. The ARM9TDMI core area is 2.1mm^2 and its clock is 200MHz [5]. The SDC architecture core area increases 95% comparing to ARM9TDMI core area and will obtain 41% performance speedup.

7: CONCLUSIONS

The proposed architecture, Superscalar Dual-Core (SDC) architecture, consists of two five-stage pipeline ARM processor cores coupled with the instruction dispatched unit (IDU). The IDU can fetch two instructions at once and dispatch the two instructions to both processor cores by the dispatched rule. According to the simulation result, the SDC architecture in the superscalar mode can obtain average 41% performance speedup comparing to the five-stage pipeline ARM processor core.

The SDC architecture can provides the fully software compatibility on original ARM based embedded systems to meet the time to market. The designers can use the SDC architecture without modifying the original software of the embedded system to shorter the reengineering time and improve the performance.

In order to improve the utility of the dual-processor, this paper provides new extended instructions and three operation modes of the SDC architecture. According to different applications of embedded systems, the designers can dynamically change the operation mode of the SDC architecture to obtain the balance between the utility of the hardware and the consumption of the power.

In the feature, the data flow scheme between two processor cores will be constructed to increase the ILP of the programs. The optimized compiler for SDC architecture will be designed to get higher performance.

REFERENCES

[1] Kalla, R.; Balaram Sinharoy; Tendler, J.M.; " IBM Power5 chip: a dual-core multithreaded processor," Micro, IEEE Volume 24, Issue 2, Mar-Apr 2004 Page(s):40 – 47

[2] Kaneko, S.; Kondo, H.; Masui, N.;"A 600-MHz single-chip multiprocessor with 4.8-GB/s internal shared pipelined bus and 512-kB internal memory", Solid-State Circuits, IEEE Journal of Volume 39, Issue 1, Jan. 2004 Page(s):184 - 193 Digital Object Identifier 10.1109/JSSC.2003.820866

[3] T. Koyama, K. Inoue, H. Hanaki, M. Yasue, and E. Iwata, "A 250-MHz single-chip multiprocessor for audio and video signal processing," IEEE J. Solid-State Circuits, vol. 36, pp. 1768–1774, Nov. 2001.

[4] Bishop, B., Kelliher, T.P. et al., "A detailed analysis of MediaBench," Signal Processing Systems, IEEE Workshop on, pp. 448-455, 1999

[5] Steve furber, "ARM system-on-chip architecture second edition ", Addison-Wesley, pp. 262, 2000