COST-EFFECTIVE MICROARCHITECTURE OPTIMIZATION FOR ARM7TDMI MICROPROCESSOR

Ing-Jer Hunag and Yu-Liang Hung and Chi-Shaw Lai

Department of Computer Science and Engineering,
National Sun Yat-Sen University, 70 Lien-hai Rd. Kaohsiung, Taiwan
Email:ijhuang@cse.nsysu.edu.tw

ABSTRACT

In this paper, we present how to optimize our ARM7TDMI instruction set compatible microprocessor. The ARM7TDMI is a 32-bit microprocessor developed by ARM Ldt. It used in embedded application such as mobile phones, pager and PDAs. The ARM7 family owes its success to the combination of low power, low cost and high performance. However, as applications become more complex and integrate more and more functionality, the processor is required to provide more and more performance. We use synthesis tool to synthesize our RTL design and analyze timing to fund the critical path of our microprocessor. We will describe how to optimize the critical path to increase performance.

1. INTRODUCTION

Since the scientists invented semiconductor, the electronic technology has advanced at a rapid rate. Today, many electronic products use microprocessor based application such as PDAs and mobile phones. However, as applications become more complex and integrated more and more functionality, the processor is required to provide more and more performance.

ARM7TDMI is a 32-bits microprocessor that developed by Advanced RISC Machines Ltd. ARM7 has been successful in many portable applications. In our laboratory, we also developed an ARM7TDMI instruction set compatible microprocessor. We will optimize this microprocessor to provide higher performance and discuss the cost-effect in different optimization methodology.

We know a CPU execute a program time is:

CPU time = Instruction count ´ CPI ´ Clock cycle time

There are three factors can effect the CPU time. If we can improve one of three factors we can improve the processor performance. Instruction count is effected by compiler technology or advanced algorithm. It can use fewer instructions to complete the same thing. The CPI is Cycle Per Instruction that is an average number of an instruction execution cycles. If we use advanced computer architecture like superscalar or cache memory that can reduce memory accesses cycles, the CPI value can be reduced. The clock cycle time is the clock period time. In this time, all hardware must figure out the results and wait to save to registers.

In this paper, we focus on the cycle time optimization.

We will fund the critical path of this microprocessor and optimize this path. We will describe how to optimize and after change the microarchitecture what will be effected.

- **ARM9**

To gain the extra speed, ARM has finally extended its characteristically short three-stage pipeline to five stages. Figure 1 maps the differences the ARM7 and ARM9. In ARM9, the trivial fetch stage from ARM7 remains, but the overstuffed execute stage is now spread among no fewer than four stages, with some work being done in the decode stage.
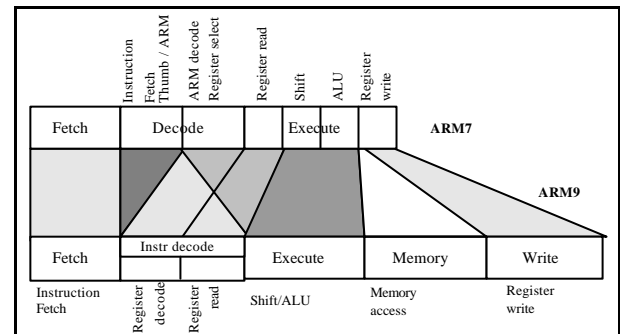

Figure 1: The differences between the ARM7 and ARM9 pipelines

The re-pipelining allows the clock rate to be increased. The ARM9TDMI may be clocked at twice the rate of ARM7TDMI. With these changes, ARM will reach 120-150 MHz in commodity 0.35-micron processes. ARM9 has an average clocks-per-instruction ratio of 1.5, according to ARM, substantially better than ARM7's 1.9 CPI. The forwarding paths in ARM9 allow instructions to execute without stall cycles. The increase in complexity required to achieve the increase in performance requires around 50% more transistors and the area has increased by almost 90%. This area increase in accounted for by an increase in number of routing channels in the datapath.

	ARM7TDMI	ARM9TDMI
Area (mm², 0.35μm)	2.2	4.15
Transistor count	74K	112K
Pipeline stages	3	5
CPI	1.9	1.5
MIPS/MHz	0.9	1.1
Typical Max Clock rate (0.35μm)	60	120
Power (mW/MHz @3.0V)	1.5	1.8

Table 1: ARM7 v.s ARM9 comparison summary

1

2. MICROARCHITECTURE

According to the ARM7TDMI instruction set, we must build a lot of functional units, include ALU, Shifter, Multiplier, Register File, Status Register and Thumb instruction Decompressor. The microprocessor we design is three-pipeline stage architecture and we will decompose this architecture to three mainly models that are fetch model, decode model and execute model.
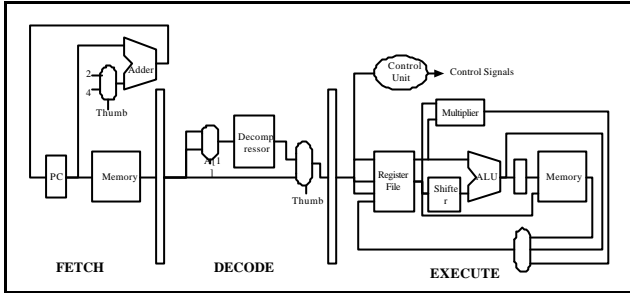

Figure 2: ARM7 Block Diagram

2.1 Fetch Stage

In this stage, we read an instruction from memory which memory address comes from the PC register. The PC address is incremented by 4 or 2 and written back into PC register for the next clock cycle. Figure 3 shows the fetch stage block diagram. T flag is used to select the distance for ARM or Thumb instructions (the distance can be 4 or 2). If the next instruction is an ARM instruction, then T flag will be zero (the distance is 4). If branch instruction occurs, then the PC value is Branch address. Otherwise, the PC value will be the adder result. If interrupt occurs, then the PC register will from interrupt address. If load instruction that will modify PC register is executing, then the PC register will from MemDataOut. The PC value will be the address of instruction in execute stage plus 8 because the pipeline. When a multi-cycle instruction is executing, the fetched instruction will be save in a register and PC will plus 4 doing prefetch.
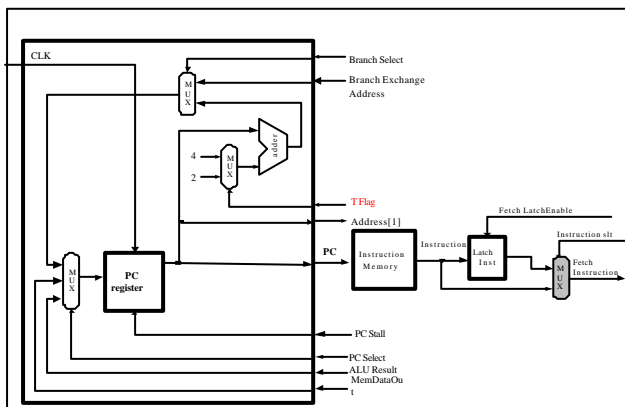

Figure 3: Fetch Stage Block Diagram

2.2 Decode Stage

In this stage, the instruction that came from the preceding stage has to be decompressed and decoded. If we read a

Thumb instruction, then we will translate Thumb instruction to ARM instruction by decompressor module. After translating, we will decode this 'ARM' instruction. When an instruction enter decode stage, the control unit will generate two control signals, index and nCPI. The signal of nCPI is used to notify the instruction is a coprocessor instruction or not. If nCPI is high, then the incoming instruction is not a coprocessor instruction. If nCPI is low, then the instruction is a coprocessor instruction. Index signals are used to notify what type for this instruction is. One instruction may be has more than one action by according to its instruction fields and the control unit has to discern what actions for the instruction really is. For example, if the instruction is "LDR" that will load byte or word from memory, then the index signal is 6. Here, index signal will be propagated to the executed stage to generate its internal control signals. Figure 4 shows the decode stage block diagram and Table 2 shows the index table of instructions. When index is 1, the incoming instruction may be BL or B instruction. Then in execute stage, it will generate the corresponding control signals with index and some instruction field. Thus, in this stage, decode unit is used to generate instruction index.
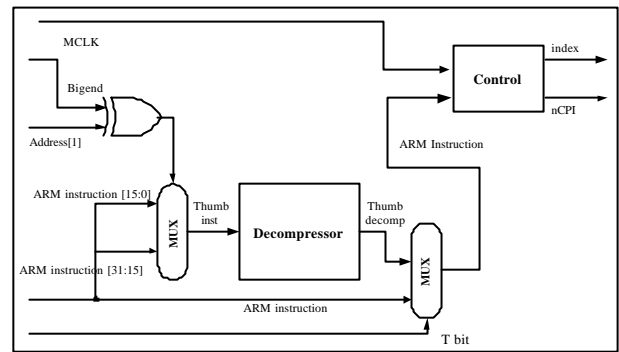

Figure 4: Decode model diagram

Index	Meaning	Corresponding ARM instruction
0	Nop instruction	
1	Branch instruction	BL,B
2	Branch instruction	BX
3	Data Processing(normal)	AND, EOR, SUB, ADD, ADC, etc.
	PSR transfer	MRS, MSR
5	Data processing, Shift(Rs)	AND, EOR, SUB, ADD, ADC, etc.
6	Data transfer : load, dest=normal	LDRH,LDRSH, LDRSB,LDR
7	Data transfer :store, dest=normal	STRH,STR
8	Multiply	MUL,MLA, MULL,MLAL
9	Block data transfer : load	LDM
10	Block data transfer : store	STM
11	Data swap : source/dest = normal	SWP
12	Software interrupt	SWI
13	Coprocessor data operation	CDP
14	Coprocessor data transfer :load	LDC
15	Coprocessor data transfer :store	STC
16	Coprocessor register transfer : load	MRC
17	Coprocessor register transfer : load	MCR
18	Undefined interrupt	Undefined

Table 2: The indexes distribute

2

2.3 Execute Stage

In this stage, functional units are executed according to the control signals that have been decoded. Figure 5 shows the execute stage block diagram. The condition unit will generate cond_bit signal that decides this instruction will be executed or not. In ARM instruction set, all instructions are conditionally executed according to the state of CPSR condition code and the instruction's condition field. The field (bit 31:28) determines the circumstances under which an instruction is to be executed. There are fifteen different conditions may be used. Table 3 listed the condition code summary. The sixteenth (1111) is reserved, and must not be used. In the absence of a suffix, the condition field of most instructions is set to "Always" (suffix AL). This means the instruction will always be executed regardless of CPSR condition codes.
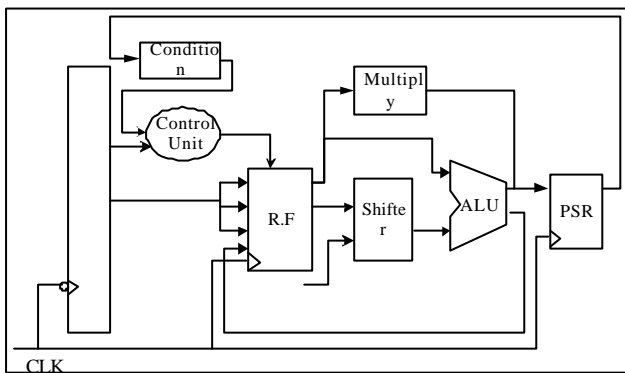

Figure 5: Execute stage block diagram

Code	Suffix	Flags	Meaning
0000	EQ	Z set	Equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	Negative
0101	PL	N clear	positive or zero
0110	VS	V set	Overflow
0111	VC	V clear	not overfolw
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear and Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	Always

Table 3: Condition code summary

The control unit will generate all control signals according to index from decode stage and instruction under executed. Shifter unit can 5 kinds shift operations, include logical shift left (LSL), logical shift right (LSR), arithmetic shift left (ASL), arithmetic shift right (LSR) and rotate right

(ROR). ALU unit is used to do 16 different arithmetic and logic operations. And the multiplier is executed for multiply instructions.

Register File includes 31 general-purpose 32-bit regis ters – but these cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer. The PSR unit is Program Status Register that includes condition code flags and control bits.

2.4 Multiplier

ARM instructions include multiply and multiply- Accumulate instructions. This multiplier must perform unsigned and signed multiplication with optional accumulate. We use an 8-bit Booth's algorithm to perform integer multiplication. The multiplier and multiplicand are extended from 32 bits to 33 bit for differentiating signed and unsigned multiplication. In signed operation, the value will be done signed extension. But in unsigned, the highest bit will be filled zero. The multiplier perform 33*8 multiplication every cycle. But, if bits [32:8] of the multiplier operand are all zero or all one, we can reduce 3 cycle for multiplication. If bits [32:16] of the multiplier operand are all zero or all one, we can reduce 2 cycle for multiplication. If bits [32:24] of the multiplier operand are all zero or all one, we can reduce 1 cycle for multiplication. Table 4 shows the modified radix-4 booth's algorithm and Figure 6 shows the block diagram of the multiplier.

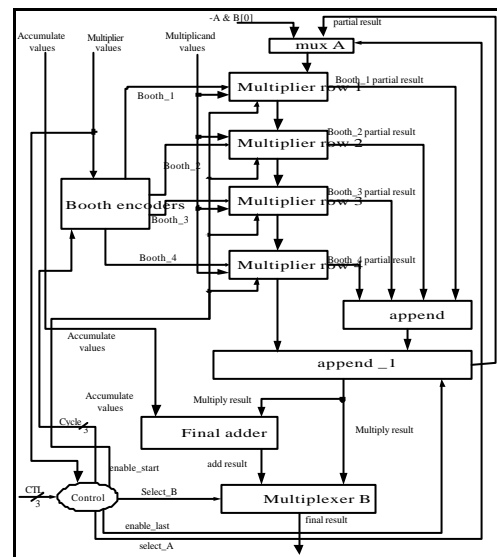| Multiplier			Operation	Remarks
B[i+l]	B[i]	B[i-l]		
0	0	0	0	String of zeros
0	0	1	+A	End of 1's
0	1	0	+A	A single 1
0	1	1	2A	End of a 1's
1	0	0	-2A	Start of 1's
1	0	1	-A	End/start of 1's
1	1	0	-A	Start of 1's
1	1	1	0	String of 1's

Table 4: Booth's algorithm

Figure 6: Multiplier block diagram

2.5 Pipeline interaction

Because the cycle counts for ARM instruction is different ranging from 1 to 17 cycles and it is more difficult for us to design. Table 5 shows the instruction cycle summary. It not only tells us the cycles in every instruction, but also tells us the interaction with other stages. We partition all instructions into 15 different types. The gray shade is the execution cycle. Besides, it also controls the interaction with other stages. The "N" shows the cycle in the next cycle that is non-sequential cycle. In another way, the PC value will not be sequential and the fetch and decode stages have to be flushed. The "S" shows the cycle in the next cycle that is sequential cycle. In another way, the PC value will be sequential. The "I" shows the cycle in the next cycle that is internal cycle. In another way, the PC value will not be changed until the instruction is completed. Here, the fetch and decode stages have to be stalled. The "C" shows the cycle in the next cycle that is coprocessor register transfer cycle. The PC value will not be changed until the coprocessor register transfer cycle is completed. When the processor enter the coprocessor register transfer cycle, then the processor wishes to use the data bus to communicate with a coprocessor, but does not require any action by the memory system.

The circle shows to flush decode and fetch stages when the PC register is modified in execute stage. The fetch and decode stages have to be stalled when an instruction executes multi-cycles. The "[" shows begin of stall and the "]" shows end of stall. From Table 5, we know some instructions spend one cycle and some instructions may spend more than one cycle. For multi-cycle instruction, the control signals of the next cycle are affected by the status of current cycle. Hence, our control unit is designed based on mealy machine. For example, Branch instruction takes three cycles that is N, S, S cycle. When Branch instruction is executed, it will update PC value from the branch address. The fetch and decode stages have to be flushed. Here, when the next PC value is not sequential address, the next cycle type is N cycle. Following the N cycle, it is S cycle.

Action Type	1	2	3	4	5	6	7	8	...	...		Cycles
1. Branch and Branch with Link	○	S	S									3
2. Branch and Exchange	○	S	S									3
3. Data Operations: normal	S											1
dest=pc	○	S	S									3
shift(Rs)	I	S										2
shift(Rs),dest=pc	I	○	S	S								4
4. Multiply and Multiply Accumulate: MUL	I	I	I	I	S							5
MLA	I	I	I	I	I	S						6
MULL	I	I	I	I	I	S						6
MLAL	I	I	I	I	I	I	S					7
5. Load Register: normal	N	I	S									3
dest=pc	N	I	○	S	S							5
6. Store Register	N	N										2
7. Load Multiple Registers: n registers	N	S	S	S	S	...	S	S	I	S		n+2
n registers,incl pc	N	S	S	S	S	...	S	S	○	S	S	n+4
8. Store Multiple Registers : n registers	N	S	S	S	S	...	S	S	N			n+1
9. Data Swap	N	N	I	S								4
10. Software Interrupt and Exception Entry	○	S	S									3
11. Coprocessor Data Operation : Ready	N											1
not Ready	I	I	I	I	I	...	I	N				n
12. Coprocessor Data Transfer : n registers , Ready	N	I	S	S	S	...	S	S	N			n+1
m registers, not Ready	I	I	I	...	N	S	S	...	S	S	N	n+m+1
13. Coprocessor Register Transfer : MRC ,Ready	C	I	S									2
not Ready	I	I	I	...	I	C	I	S				n+2
14. Coprocessor Register Transfer : MCR , Ready	I	N										2
not Ready	I	I	I	...	I	C	N					n+1
15. Undefined Instruction	I	○	S	S								4

Table 5: Instruction cycles summary

Thus, we will decompose the action of decode into two steps. First, we will find what kind of instruction that the current instruction is, then we send a set signals to next decode components. When the next decode components

get these signals, it will send control signals to control functional units according to these signals from the prior decode components. Figure 7 shows the control model design. Output function is used to generate the data path controls according to instructions, processor status, conditional flags, and control inputs from the previous stage. And next-state function is used to generate next_state according to status and processor status and control inputs from the previous stage.
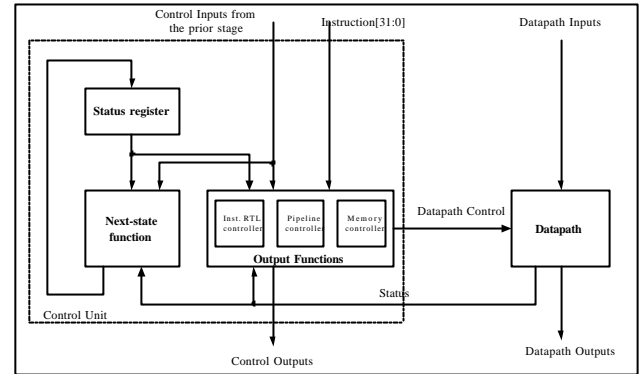

Figure 7: The Control model

3. OPTIMIZATION

3.1 Critical path

After synthesize original version, we found the critical path is in execution stage. The critical path is the condition code flags read from PSR to Condition unit and Control unit to decode control signals, then read register operands from Register File to perform the first cycle multiplication in multiplier and save the first cycle results to a register. This critical path was shown in Figure 8's path ✱. We will describe how to optimize the multiplier.
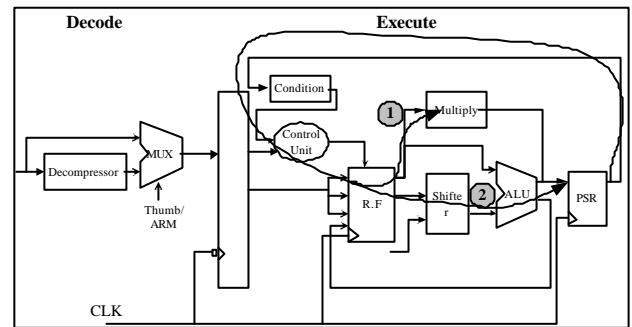

Figure 8: The critical paths

After multiplier optimization, the critical path appears on another path. The front of this path is the same as path ✱. But the tail is not through the multiplier, it is through the Shifter-ALU and write the flags to PSR. We will describe this path optimization. Table 6 is the critical path's delay. The delays both are started from negative edge triggered register to positive edge triggered register. The clock cycle is double of the delay time, if the clock duty cycle is 50%.

4

Critical Path	Delay	Clock rate
Path ✶	41.3 ns	12 MHz
Path ✶	33.5 ns	14 MHz

Table 6: The critical path's delays

3.2 Multiplier Optimization

● **Use single edge triggered**

In multiplier design, it must finish 8-bit Booth's result and the partial product must be added with next 8-bit Booth's calculation. In original design, it adopts two clock edge triggered registers. The multiply operands read from Register File perform first 8-bit multiplication and save to a positive edge triggered register. Then, the result of the Booth's multiplication perform signed extension to save in negative edge triggered register prepare for next Booth's calculation. But for using clock's duty cycle is 50%, the cycle time must be the longest delay time twice. So, we change to use negative edge triggered register design
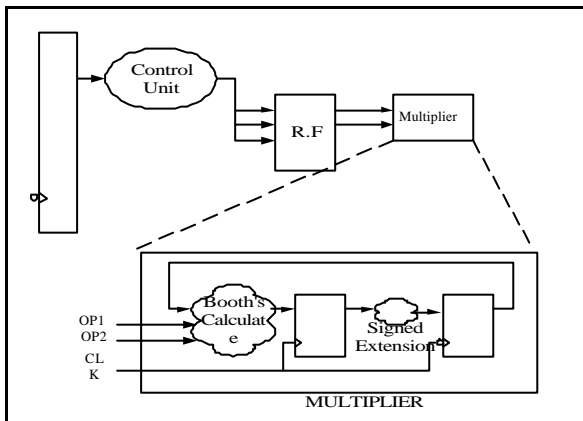

Figure 9: The multiplier use dual clock edge triggered registers

● **Use Carry Save Adder**

In multiplier, we must add the multiplier rows that are four 40-bit numbers. In original design, it uses carry ripple adder to perform addition. But, it will generate long carry chain of adders. We use Carry Save Adder instead of carry ripple adder. It avoids the long carry chain delay. The last, using a fast adder like carry look-ahead adder to add carry and sum that are generated by carry save adder. We use the dw01_csa that is carry save adder of Synopsys® DesignWare Component. Figure 10 is the addition architecture using carry save adder.
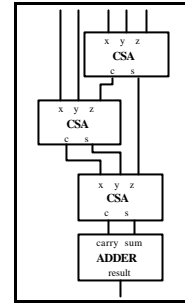
In multiplier, we must add the multiplier rows that are four 40-bit numbers. In original design, it uses carry ripple adder to perform addition. But, it will generate long carry chain of adders. We use Carry Save Adder instead of carry ripple adder. It avoids the long carry chain delay. The last, using a fast adder like carry look-ahead adder to add carry and sum that are generated by carry save adder. We use the dw01_csa that is carry save adder of Synopsys® DesignWare Component. Figure 10 is the addition architecture using carry save adder.

Figure 10: Using Carry save adder

● **Put off the final adder**

In multiplier, we use the carry save adder to calculate the partial product. The final, we use a carry look-ahead adder to add sum and carry and save to a register. It must finish the carry save adder and the final adder addition in a multiply cycle. We can put off this final adder. The carry and sum of carry save adder are not necessary added in every cycle. The final addition can be performed after multiplier cycle that shows in Figure 11. After we put off the final adder, it increment use one carry save adder and twice registers.
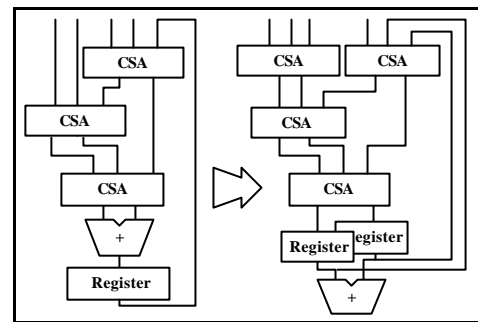

Figure 11: Put off the final adder

3.3 Execution Stage Optimize

After we optimize the multiplier, the critical path becomes path ✱ of Figure 8. It includes Condition unit, Control unit, Register file, and Shifter, ALU and PSR unit. We will describe the execution stage optimization.

● **Use single edge triggered**

In execution stage, the instruction is read from negative edge triggered pipeline register and decoded control signals. Then read registers from register file and through shifter and ALU operation. Final, save flags to PSR unit at positive edge clock. It also uses dual clock edge triggered design. We adopt PSR and register file in single clock edge triggered instead of dual edge triggered.
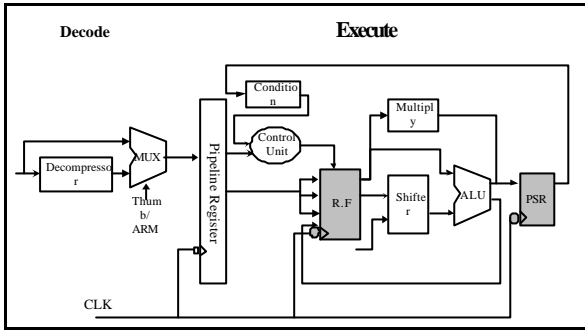
5

Figure 12: Use single edge triggered in execution stage

● **Register file use DesignWare component**

The Register File includes 30 32-bit registers, two read ports and one write port. In original design, it is Verilog RTL design using latch based. But consider design for test and synthesis, we adopt DW_ram_2r_w_s_dff of Synopsys® DesignWare Component that is synchronous write-port and asynchronous dual read-port RAM (Flip-Flop based).

● **Condition decision**

Every ARM instructions are conditionally execution according the instruction field and the CPSR condition code flags. The Condition unit is used to decide the condition. If the flags fulfil the condition of field, the instruction is executed, otherwise it is ignored. In original design, it decides the condition first and then generates control signals, see Figure 13. If instruction is ignored, the index will be mask and control unit will decode NOP control signals.
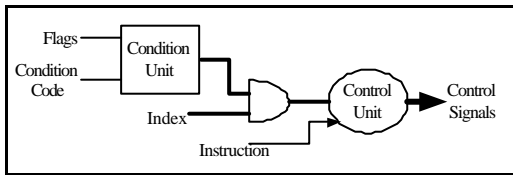

Figure 13: original condition design

The condition decision can succeed control unit decode the control signals. Figure 14 shows the modified block diagram. If condition not satisfied, the multiplexer can be selected NOP control signals by condition bit. We will increment use a multiplexer.
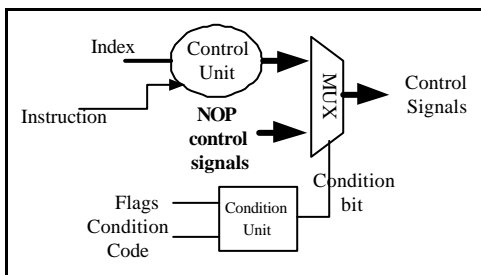

Figure 14: Modified condition design

3.4 Prefetching Register File Operands

Our microprocessor adopts 3 pipeline stages design that are fetch, decode and execute. After Execution stage optimi-

zation, the critical path stills on path ✳ of Figure 8. In decode stage, the Thumb instruction translated to ARM instruction by Decompressor and generate index number. The maximum delay of decode stage is only one three of execution's delay. If Register File and control unit move to decode stage, we can minimize the cycle time. After re-pipelining, the microarchitecture must be modified for execution accurate. We will discuss them below.
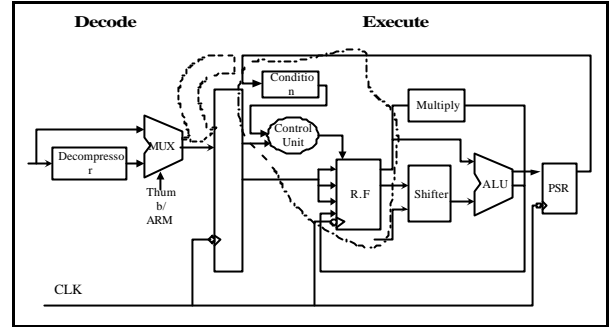

Figure 15: Register File and control unit move to decode stage

● **Data Hazard**

Register File is read in decode stage. If execution stage write register is the same as decode stage read, it will occur data hazard. The register is read in decode stage before the same register is written in the execution stage. The read register is old value. It shows in Figure 16.
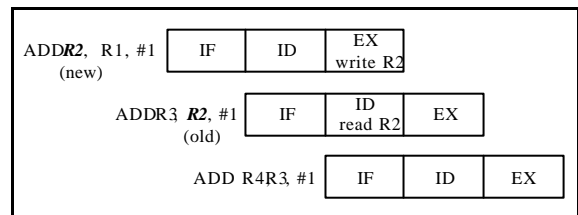

Figure 16: Data Hazard in instructions

We resolve this problem using forwarding path allow back to back data processing instructions to execute in the pipeline without stall cycles. The Forwarding unit in Figure 17 determines the read data is from register file or written data of execution stage.
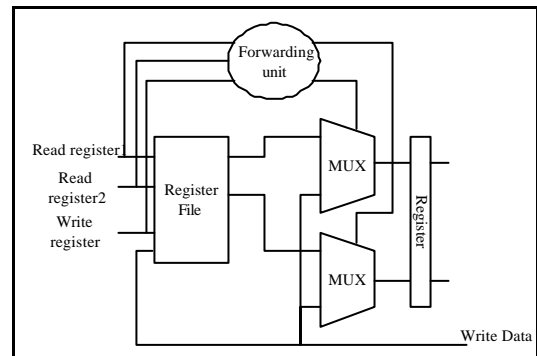

Figure 17: Using forwarding path to resolve data hazard

● **Pipeline interaction**

If the instruction is single cycle execution, the pipeline will operate normally. Because the Register File and control

unit are moved to decode stage, the control signals must be delayed one cycle to sent execution stage. But multi-cycle execution, the control unit will decode the same instruction more than one cycle. The pipeline control has a little difference. When multi-cycle execution, decode stage will still operation. So, the pipeline register between decode stage and execution stage can not be stalled. When multi-cycle execution, the control unit is decoding the instruction that is in the execution stage, not in decode stage. Until the last cycle, control unit starts to decode the decode stage's instruction.

4. RESULTS

We use synthesis tool to synthesize our design and analyze timing to fund the critical path. We use Synopsys® Design Compiler for our synthesis tool and use Galax! Inc. CB35OS142 0.35μm cell library.
Table 7 is the optimization results. We calculate speed up compare with previous version use Amdahl's Law. The path ✳ and ✳ are in Figure 8.

Version	Delay	Clock rate	Speed up	Gate count	Critical path
A	*41.27	12 MHz	-	42616	✳
B	*33.5	14 MHz	1.24	39700	✳
C	58.2	17 MHz	1.15	39152	✳
D	32.93	30 MHz	1.77	37373	✳
E	31.68	31 MHz	1.04	38371	✳
F	29.92	33 MHz	1.06	43857	✳
G	24.29	41 MHz	1.23	44121	✳
H	18.56	53 MHz	1.3	44758	decode

*: from negative edge to positive edge delay

Table 7: The optimization results

The version A is the original version. The maximum delay is 41.27ns from pipeline register to PSR's register. But, it is from negative edge to positive edge delay. The clock cycle time is twice of delay about 83ns, the clock rate about 12 MHz.
The version B adopts single edge triggered registers and reduces a lot of registers and latches in execution stage. The gate count reduces about 2900 gates but the speedup rise 1.24. The critical path from path ✳ becomes path ✳.
In version C, the execution stage also adopts single edge triggered registers. The speedup is not obvious because the critical become path ✳.
The version D, we use carry save adder instead of carry ripple adder and also reduce registers and latches to use in multiplier. The speedup is 1.77 and the gate count reduced about 1700 gates.
In version E, we put off the final adder calculation. The multiplier's speedup is 1.4 but the overall's speedup is 1.04. The reason is the critical path becomes to path ✳. The multiplier increases the carry save adder and registers to cause the gate count increase about 1000 gates.
The version F, the Register file is used dw_ram_2r_w_s_dff of Synopsys® DesignWare Component. This Designware is flip-flop-based design but the original is latch-based design. It increases about 5400 gates and the speed up is 1.06. The flip-flop based register file can easy to design for test.
The version E is condition decision succeed the control unit. That's speedup is 1.23 and only about 250 gates in-

crement. It is a good strategy.
The last version G, we prefetch the register operands in decode stage. The speedup is 1.3 and increases some registers and forwarding path. It spread the pipeline more evenly, thereby permitting a higher maximum operating frequency.

5. CONCLUSION

In this paper, we have presented how to optimize our ARM7TDMI instruction set compatible microprocessor. The clock rate is from 12 MHz to 53 MHz. The CPI value is the same as the original design. The performance is increased about 4.4 and the gate count only increased about 2K gates.

In the future, the pipeline stages can be extended from 3 to 5 stages like ARM9TDMI. It spread the pipeline more evenly. Some of the ARM instructions are multi-cycle execution in ARM7 that have to stall the pipeline. ARM9 use forwarding paths to avoid stalling the pipeline and reduce the CPI value.

6. REFERENCES

[1] http://www.arm.com
[2] Simon Segar, "The ARM9 Family – High Performance Microprocessors for Embedded Applications" IEEE International Conference on Computer Design 1999 (ICCD'99).
[3] Steve Furber "ARM System Architecture" Addison Wesley Longman Inc, 1996.
[4] Michael Keating and Pierre Bricaud "Reuse Methodology Manual for System-on-a-Chip Designs", 2nd Edition, KLUWER ACADEMIC PUBLISHERS.
[5] "DesignWare Components Databook, Vol 1, Foundation Librarys", Version 1997.08, Synopsys Inc.
[6] David A. Patterson and John L. Hennessy, "Computer Organization & Design – The Hardware/Software Interface", 2nd Edition, Margan Kaufmann Publishers, Inc.
[7] John L. Hennessy and David A. Patterson, "Computer Architecture A Quantitative Approach" 2nd Edition, Margan Kaufmann Publishers, Inc.
[8] Behrooz Parhami, "Computer Arithmetic - Algorithms and Hardware Designs", Oxford University Press, Inc.
[9] Mike Clark and Lizy Kurian John, "Performance Evaluation of Configurable Hardware Features on the AMD-K5", IEEE International Conference on Computer Design 1999 (ICCD'99).