國 立 交 通 大 學

電子工程學系　電子研究所

碩 士 論 文

使用 32 位元低功率嵌入式處理器
之高效能 MP3 解碼系統

Efficient MP3 Decoder System using a
32-bit Low-Power Embedded Processor Core

研 究 生：辛威虢

指導教授：黃俊達　博士

中 華 民 國 九 十 八 年 一 月

使用 32 位元低功率嵌入式處理器之高效能 MP3 解碼系統

Efficient MP3 Decoder System using a 32-bit Low-Power Embedded Processor Core

研 究 生：辛威虢　　Student：Wei-Kuo Hsin

指導教授：黃俊達　博士　　Advisor：Dr. Juinn-Dar Huang

國 立 交 通 大 學

電子工程學系　電子研究所

碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical & Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Electronics Engineering & Institute of Electronics

January 2009
Hsinchu, Taiwan, Republic of China

中華民國九十八年一月

使用 32 位元低功率嵌入式處理器
之高效能 MP3 解碼系統

研 究 生：辛威虢　　　　　　指導教授：黃俊達　博士

國立交通大學

電子工程學系　電子研究所碩士班

摘　　要

本論文提出一個高效能的 MP3 音訊解碼系統使用低功率之 32 位元嵌入式處理器 ACARM9(Academic ARM9)。其一：此處理器有較佳的性能。此處理器以 ARM V5E 指令集實作，並且在乘法部份有更好的效能，此指令集能使用 ADS(ARM Developer Suite) 的編譯器將高階程式語言(C,C++)編譯成組合語言，再將其組譯為可供 ACARM9 使用的機器語言，以顯示出此處理器的高使用性。其二：本論文提出一個完整的系統發展平台，此處理器之設計被燒錄在 FPGA 之上，再使用 ARM926EJ-S Versatile 發展板系統，以本處理器解碼 MP3 音訊資料，達到高效能的解碼速率及音訊同步化播放，完成 MP3 播放系統。

Efficient MP3 Decoder System using a 32-bit Low-Power Embedded Processor Core

Student: Wei-Kuo Hsin Advisor: Dr. Juinn-Dar Huang

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

ABSTRACT

This thesis presents the research result of an efficient MP3 decoder system using a 32-bit low-power embedded processor core, named ACARM9 (ACademic ARM9). The ISA (Instruction Set Architecture) of ACARM9 adopts the ARM V5E architecture but owns more efficient multiplication instructions. Hence the ADS (ARM Develop Suite) can be directly used. ADS can first be used to compile the code of high level programming language (C, C++) written by users to the assembly code, and then can assemble the assembly code to the low level machine code for ACARM9 use. It indicates the high usability of ACARM9. Moreover, this thesis presents a methodology for platform-based design. The proposed processor is mapped onto the FPGA and integrated within the ARM926EJ-S Versatile Development Board. Then an MP3 decoder system, which is capable of providing high decoding rate and playing audio synchronously, is successfully implemented using the proposed platform.

Acknowledgement

致　　謝

完成這篇論文，感謝指導老師-黃俊達博士的細心教導，在研究方向的指點迷津，以及實驗室環境和儀器的完善提供，給了我最好的研究環境，並且培養出獨立自主研究的精神。

感謝 ACAR 實驗室學長、同學及學弟妹們，之暉、哲霖、南興、建德、瀚蔚、亞謙及于翔，感謝你們從旁給予建議與協助。

感謝清華工科 06 級的大學同學們，分享你們的工作或研究經驗，讓我在未來旅途有了更明確的方向。

感謝在背後默默支持的父母弟妹，感謝父母從小不辭勞苦的栽培我，和弟妹的默默支持。最後感謝感謝各位口試委員，在百忙之中抽空來給予指導，在完成這篇論文後，給我建議和經驗。

還有其他未提及的朋友們，感謝大家。

Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Motivation

Nowadays, a number of digital consumer electronic products, which include Personal Digital Assistant (PDA), cell-phone, Nintendo Dual Screen (NDS), Apple iPod, iPhone and so on, have grown up drastically. Either embedded processors or Digital Signal Processors (DSP), which are powered by the batteries, are included in all these portable electronic products. Therefore, how to save more power of these portable electronic devices is one of the most important subjects in the competitive market. Moreover, portable electronic products have more and more integrated multimedia, due to the design complexity and time-to-market, the processor-based solution becomes a promising and desirable solution.

1.2 Contribution

This thesis proposes an efficient MP3 decoder system using a 32-bit low-power embedded processor core. Our proposed way is to design an in-house processor core ACARM9 in register transfer level (RTL) and to port an efficient MP3 decoder software using a RealView Platform Baseboard for ARM926EJ-S. The experimental results show the performance comparisons among different audio bitstream rates and the maximum performance on the platform.

1.3 Thesis Organization

Chapter 1 introduces the motivation that the number of digital consumer electronic products increases so dramatically nowadays. The new products are rapidly released and they are all powered by the batteries. Therefore, there are two challenges. How to save more power of these portable electronic devices and how to design a product with short time-to-market cycle are the most important subjects in the competitive market. A low-power embedded processor with MP3 decoder software is proposed in this thesis.

Chapter 2 describes the previous works related to ARM9E-S which is a member of general purpose 32-bit embedded processor of the Advanced RISC Machines (ARM) family. MPEG-1 Audio Layer-3, more commonly referred to as MP3, is a digital audio encoding format using a form of lossy data compression. It is a common audio format for consumer audio storage.

Chapter 3 describes the proposed core design ACARM9 which improves the operating performance and power consumption. The architecture of the proposed design has been discussed in this chapter. The verification strategy and synthesis result are described in the following sections.

Chapter 4 presents an MP3 Platform-based System implemented by ACARM9 which is configured in an FPGA as a specific purpose processor and all system behaviors are controlled by an ARM926EJ-S processor on the development board. The system can be examined in three different aspects, which include system level, hardware level and software level.

Chapter 5 describes the MP3 decoder software design which is analyzed in two phase. First is before optimization. After all the improvements the analysis shows that the minimum clock rate required in FPGA. The porting technique is also presented in this chapter.

Chapter 6 presents the experiment result of the decoder software and MP3 decoder system. The discussions of improvement are described in this chapter. The experiment result shows the minimum clock rate to real-time decode.

Chapter 7 describes the conclusion and future work of this thesis.

Chapter 2
Preliminaries

This chapter describes the preliminaries of this thesis. Section 2.1 introduces the ARM9E-S, which is a member of the Advanced RISC Machines (ARM) family of general purpose 32-bit embedded processors. Section 2.2 introduces the MPEG 1 Layer-3 (MP3) decoder. Section 2.3 describes the target system platform, RealView Platform Baseboard for ARM926EJ-S.

2.1 Overview of ARM9E-S

This section describes some prerequisites related to ARM9E-S, which is a member of the Advanced RISC Machines (ARM) family of general purpose 32-bit embedded processors. Section 2.1.1 depicts basic architecture of the processor core. Section 2.1.2 mentions the programmer's model. Section 2.1.3 describes the instruction set architecture (ISA) of ARM9E-S. More related works about ARM9E-S will be discussed in Section 3.1 later.

2.1.1 Core Architecture of ARM9E-S

The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles. The reduced instruction set and related decode mechanism are much simpler than those of Complex Instruction Set Computer (CISC) designs. This simplicity gives a high instruction throughput, an excellent real-time interrupt response and a small, cost-effective, processor macrocell.

The ARM9E-S core uses a pipeline to increase the speed of the flow of instructions to the processor. This enables several operations to take place simultaneously, and the processing and memory systems to operate continuously. A five-stage pipeline is used, consisting of Fetch, Decode, Execute, Memory, and Write-back stages. This is shown in Fig. 2.1[18]. The program counter (PC) points to the instruction being fetched rather than to the instruction being executed.

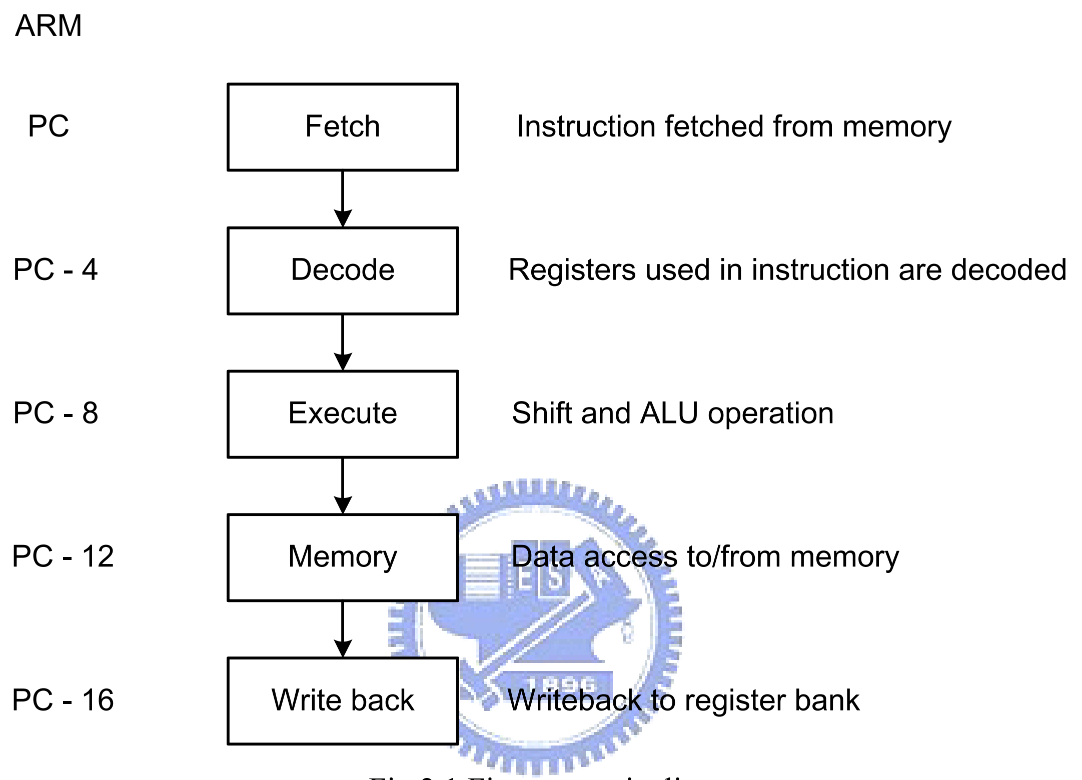

ARM
PC
Fetch
Instruction fetched from memory
PC - 4
Decode
Registers used in instruction are decoded
PC - 8
Execute
Shift and ALU operation
PC - 12
Memory
Data access to/from memory
PC - 16
Write back
Writeback to register bank

Fig 2.1 Five-stage pipeline

During normal operation, one instruction is being fetched from memory, the previous instruction is being decoded, the instruction before that is being executed, the instruction before that is performing data accesses (if applicable), the instruction before that is writing its data back to the register bank.

Fig 2.2[18] illustrates the ARM9E-S core diagram. The major components are:

- The address register and incrementer, which select and hold al memory address and generate sequential address when required.
- The register file, which contains 31 32-bit general purpose registers and six status registers.
- The 32x16 multiplier, which can perform multi-cycle multiply operations.

- The barrel shifter, which can shift or rotate one operand by an immediate offset or a register offset.
- The ALU, which can perform arithmetic and logic operations.
- The read and write data registers, which hold data storing to or lording from memory.
- The instruction decoder and control logic, which can decode instructions and generate associated control signals.
- The count leading zero (CLZ) and saturation detection (SAT) logic, which are partially useful for certain DSP applications.

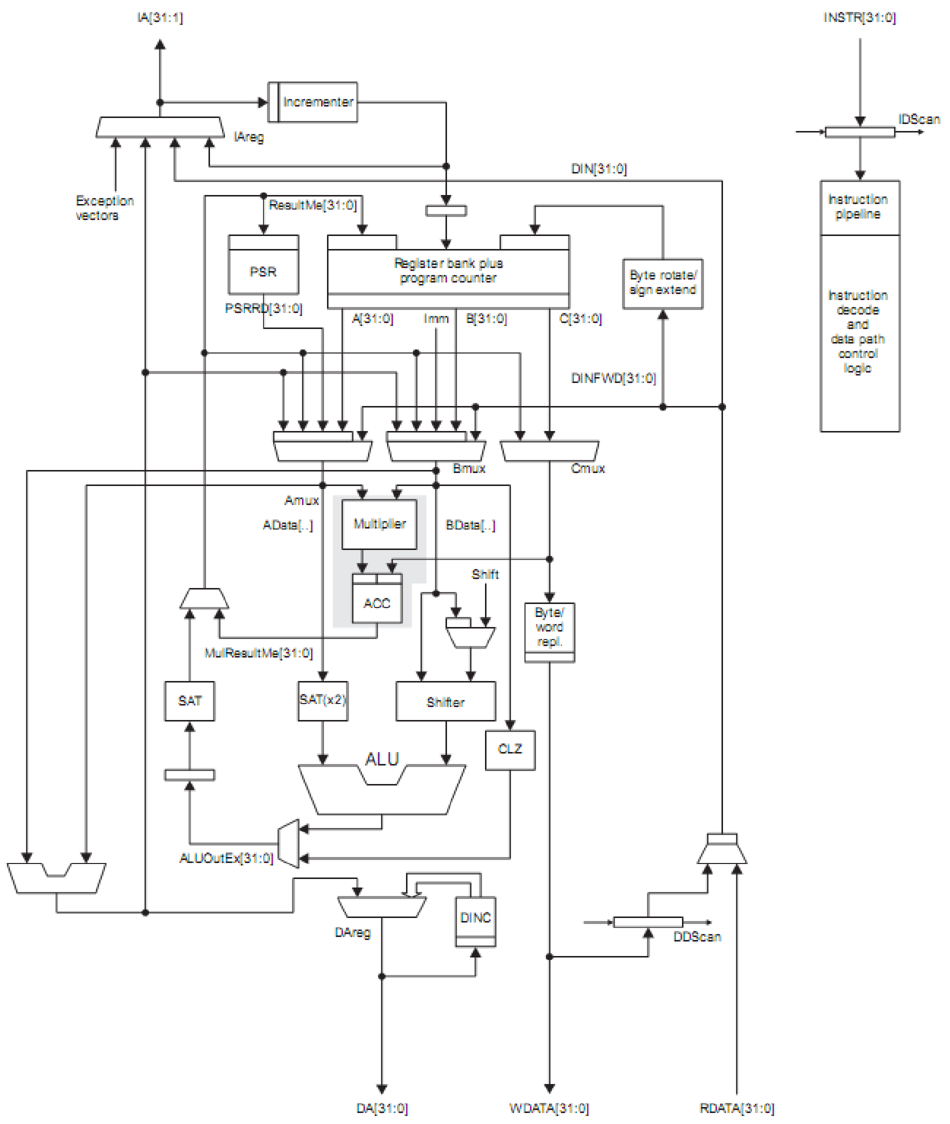
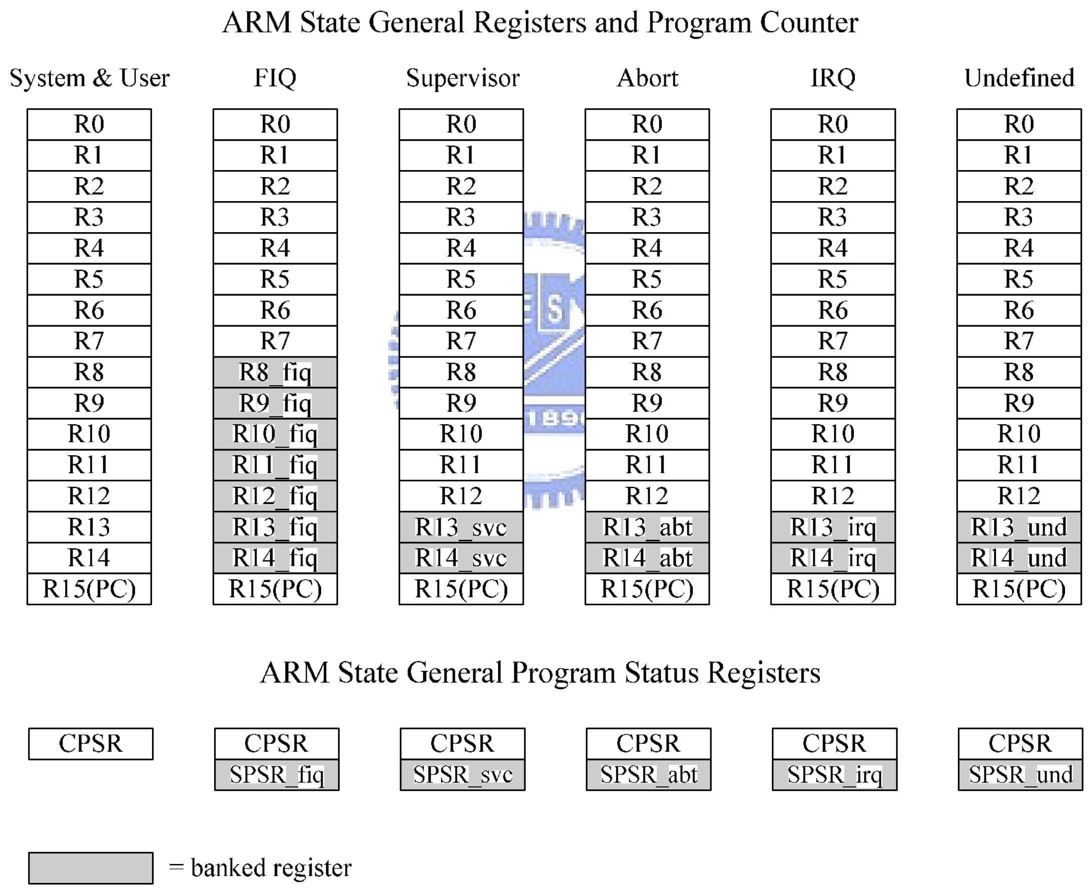


Fig 2.2 ARM9E-S core diagram

2.1.2 Programmer's Model

ARM9E-S has 37 registers including 31 general-purpose registers and six status registers as shown in Fig. 2.3[18]. The processor state and operating modes determine which registers are available to the programmer. ARM9E-S supports seven operating modes including user mode, system mode, fiq mode, supervisor mode, abort mode, irq mode, and undefined mode. The non-user modes, a.k.a. privileged modes, are used for system level programming and typically for handling interrupts or exceptions.



ARM State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)

ARM State General Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

▨ = banked register

Fig. 2.3 ARM9E-S registers organization in ARM state

ARM9E-S has six program status registers (PSRs) including a current program status register (CPSR) and five saved program status registers (SPSRs). These registers are used to hold information about the most recently executed ALU operation, control the enabling and disabling of FIQ/IRQ interrupts, and set the processor operating mode. Fig. 2.4[18] shows the PSR format. The top five bits (N, Z, C, V, Q), a.k.a. the condition code flags, may be affected by a result of an arithmetic/logical instruction, and may be used to determine whether an instruction should be executed. The bottom eight bits are the control bits. When I bit or F bit set, the processor disables IRQ and FIQ interrupts, respectively. The T bit describes the state of ARM9E-S. When T bit set, the processor is in THUMB state, otherwise the processor is in ARM state. The M4, M3, M2, M1, M0 bits are the mode bits. The mode bits determine the current operating mode of the processor. Table 2.1 shows the mode bit values and the corresponding operating modes.

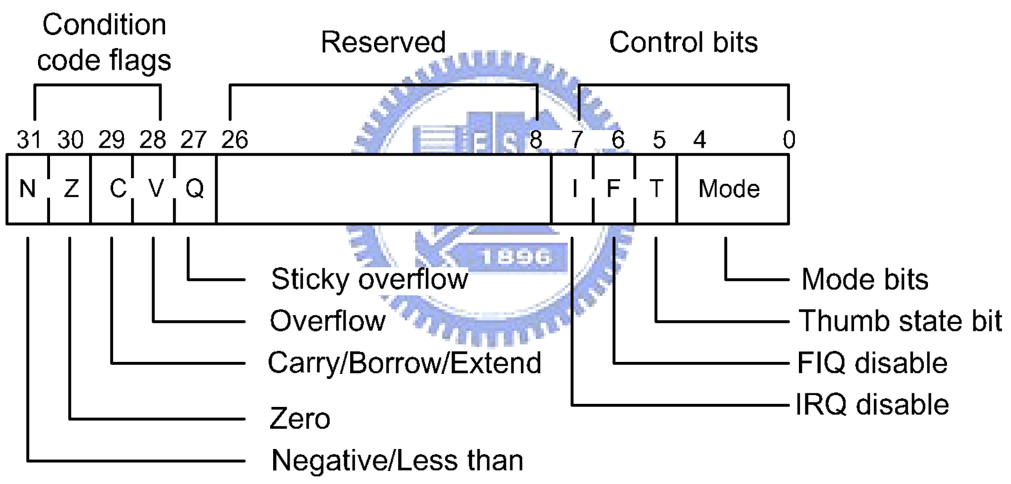


Fig. 2.4 Program status register format

Table 2.1 PSR mode bit values

M[4:0]	Mode
10000	User
10001	FIQ
10010	IRQ
10011	Supervisor
10111	Abort
11011	Undefined
11111	System

ARM exceptions can be classified into three groups:

- Exceptions generated as the direct effect of executing an instruction, such as software interrupt (SWI), undefined instruction, and prefetch abort (instruction fetch memory fault).
- Exceptions generated as the side-effect of an instruction, such as data abort (data access memory fault).
- Exceptions generated externally, unrelated to the instruction flow, such as reset, IRQ (normal interrupt request), FIQ (fast interrupt request).

When two or more exceptions arise at the same time, the exception priorities determine the order in which the exceptions are handled. Table 2.2 shows the exception priorities from high to low.

Table 2.2 Exception priorities

Priority	Exception
1	Reset
2	Data abort
3	FIQ
4	IRQ
5	Prefetch abort
6	Undefined instruction Software interrupt

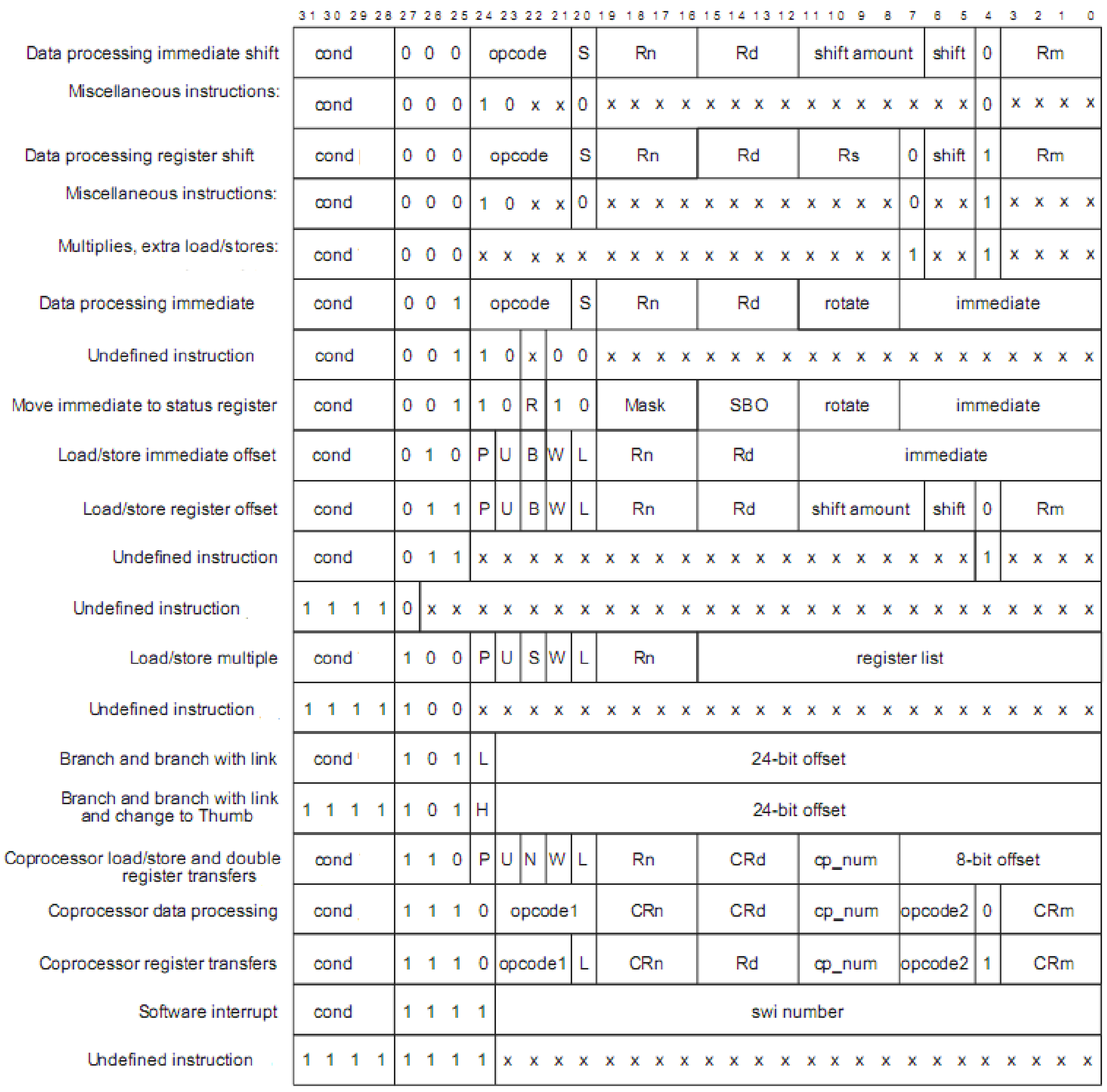
2.1.3 ARM Instruction Set Architecture Version 5

The ARM9E-S adopts the ARM ISA version 5. It can be classified into eight instruction types:

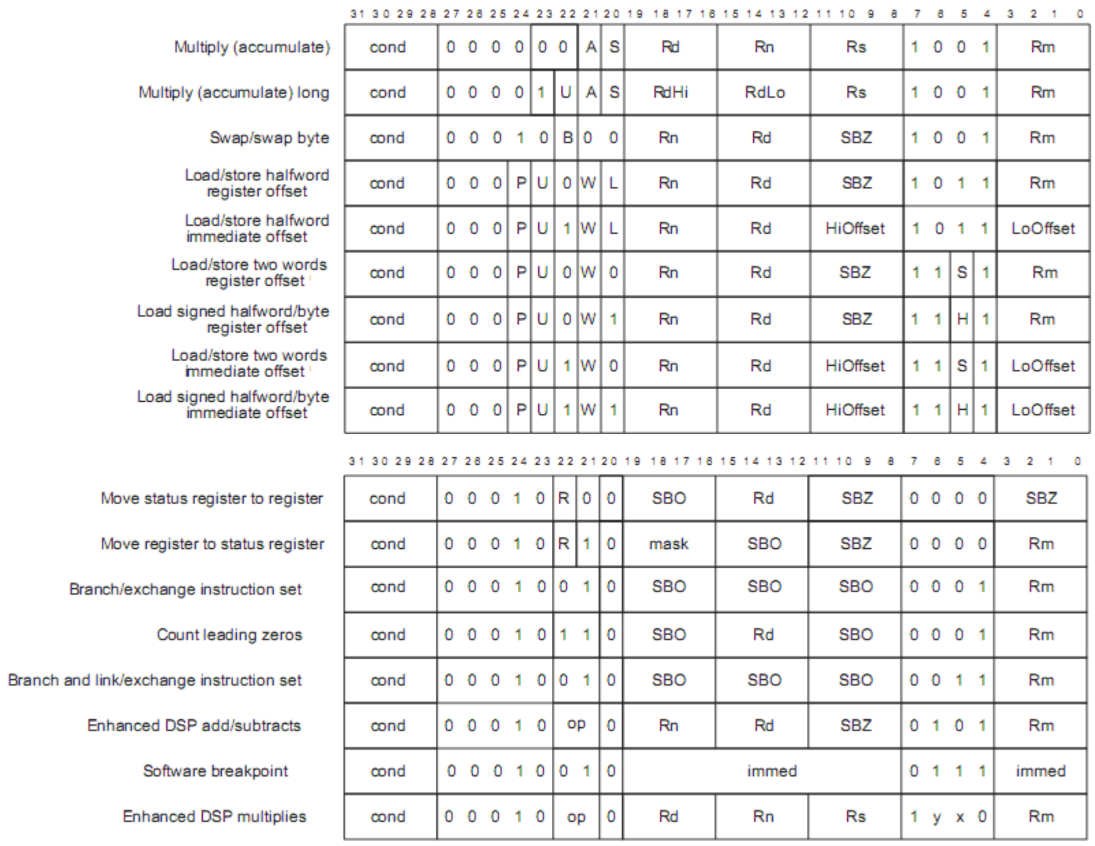
- Data processing instructions. These instructions perform arithmetic and logical operations on data values in registers and typically require two operands and produce one single result. These instructions allow the second register operand performing a shift or rotate operation before it is operated with the first operand.
- DSP enhanced instructions. There instructions include saturated integer arithmetic, half-word integer multiply and multiply-accumulate, two word load and store, cache preload and two word coprocessor register transfer.
- Program status registers transfer instructions. The MRS instruction moves PSR contents to a register. The MSR instruction moves a register contents or a 32-bit immediate value to the relevant PSR. Note that in user mode, the control bits of the CPSR are protected from change, so only the condition code flags of CPSR can be changed.
- Multiply instructions. These instructions perform multiply and multiply-accumulate operations.
- Data transfer instructions. These instructions copy memory values into registers (lord instructions) or copy register values into memory (store instructions).
- Swap instruction. This instruction exchanges values between a memory location and a register.
- Branch instructions. These instructions execute to switch to a different address, either permanently (B) or saving a return address (BL).

- Coprocessor instructions. This class of instructions is used to tell a coprocessor to perform some data operations or data transfers.
- Software interrupt instruction. The SWI instruction is used to enter supervisor mode in a controlled manner.

Fig. 2.5[6] shows the encoding formats. In the Fig. 2.5, the most significant four-bit segment of each instruction is called condition field. In ARM state, all instructions are conditionally executed according to the CPSR condition codes and the condition field of the instruction. The instruction is only executed when the condition is true, otherwise it is ignored. Table 2.3[18] lists the conditions.



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data processing immediate shift	cond				0	0	0	opcode				S	Rn				Rd				shift amount					shift		0	Rm			
Miscellaneous instructions:	cond				0	0	0	1	0	x	x	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	x	x	x	x
Data processing register shift	cond				0	0	0	opcode				S	Rn				Rd				Rs				0	shift		1	Rm			
Miscellaneous instructions:	cond				0	0	0	1	0	x	x	0	x	x	x	x	x	x	x	x	x	x	x	x	0	x	x	1	x	x	x	x
Multiplies, extra load/stores:	cond				0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	x	x	1	x	x	x	x
Data processing immediate	cond				0	0	1	opcode				S	Rn				Rd				rotate				immediate							
Undefined instruction	cond				0	0	1	1	0	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Move immediate to status register	cond				0	0	1	1	0	R	1	0	Mask				SBO				rotate				immediate							
Load/store immediate offset	cond				0	1	0	P	U	B	W	L	Rn				Rd				immediate											
Load/store register offset	cond				0	1	1	P	U	B	W	L	Rn				Rd				shift amount					shift		0	Rm			
Undefined instruction	cond				0	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	x	x	x	x
Undefined instruction	1	1	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Load/store multiple	cond				1	0	0	P	U	S	W	L	Rn				register list															
Undefined instruction	1	1	1	1	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Branch and branch with link	cond				1	0	1	L	24-bit offset																							
Branch and branch with link and change to Thumb	1	1	1	1	1	0	1	H	24-bit offset																							
Coprocessor load/store and double register transfers	cond				1	1	0	P	U	N	W	L	Rn				CRd				cp_num				8-bit offset							
Coprocessor data processing	cond				1	1	1	0	opcode1				CRn				CRd				cp_num				opcode2			0	CRm			
Coprocessor register transfers	cond				1	1	1	0	opcode1			L	CRn				Rd				cp_num				opcode2			1	CRm			
Software interrupt	cond				1	1	1	1	swi number																							
Undefined instruction	1	1	1	1	1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x



	31 30 29 28	27	26	25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6	5	4	3 2 1 0
Multiply (accumulate)	cond	0	0	0	0	0	0	A	S	Rd	Rn	Rs	1	0	0	1	Rm
Multiply (accumulate) long	cond	0	0	0	0	1	U	A	S	RdHi	RdLo	Rs	1	0	0	1	Rm
Swap/swap byte	cond	0	0	0	1	0	B	0	0	Rn	Rd	SBZ	1	0	0	1	Rm
Load/store halfword register offset	cond	0	0	0	P	U	0	W	L	Rn	Rd	SBZ	1	0	1	1	Rm
Load/store halfword immediate offset	cond	0	0	0	P	U	1	W	L	Rn	Rd	HiOffset	1	0	1	1	LoOffset
Load/store two words register offset	cond	0	0	0	P	U	0	W	0	Rn	Rd	SBZ	1	1	S	1	Rm
Load signed halfword/byte register offset	cond	0	0	0	P	U	0	W	1	Rn	Rd	SBZ	1	1	H	1	Rm
Load/store two words immediate offset	cond	0	0	0	P	U	1	W	0	Rn	Rd	HiOffset	1	1	S	1	LoOffset
Load signed halfword/byte immediate offset	cond	0	0	0	P	U	1	W	1	Rn	Rd	HiOffset	1	1	H	1	LoOffset

	31 30 29 28	27	26	25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6	5	4	3 2 1 0
Move status register to register	cond	0	0	0	1	0	R	0	0	SBO	Rd	SBZ	0	0	0	0	SBZ
Move register to status register	cond	0	0	0	1	0	R	1	0	mask	SBO	SBZ	0	0	0	0	Rm
Branch/exchange instruction set	cond	0	0	0	1	0	0	1	0	SBO	SBO	SBO	0	0	0	1	Rm
Count leading zeros	cond	0	0	0	1	0	1	1	0	SBO	Rd	SBO	0	0	0	1	Rm
Branch and link/exchange instruction set	cond	0	0	0	1	0	0	1	0	SBO	SBO	SBO	0	0	1	1	Rm
Enhanced DSP add/subtracts	cond	0	0	0	1	0	op		0	Rn	Rd	SBZ	0	1	0	1	Rm
Software breakpoint	cond	0	0	0	1	0	0	1	0	immed			0	1	1	1	immed
Enhanced DSP multiplies	cond	0	0	0	1	0	op		0	Rd	Rn	Rs	1	y	x	0	Rm

Fig. 2.5 ARM instruction set formats

Table 2.3 The condition code summary

Code	Suffix	Flags	Meanings
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	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or 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

2.2 Overview of MPEG 1 Layer-3 Decoder

This section describes some prerequisites related to MPEG 1 Layer-3, Section 2.2.1 describes the background of MPEG audio. Section 2.2.2 introduces the bitstream format of MPEG 1 Layer-3 and Section 2.2.3 describes the decoding procedure of MPEG audio.

2.2.1 Introduction

MPEG (The Motion Picture Experts Group) Layer-3, otherwise known as MP3, has generated a phenomenal interest among internet users, or at least among those who want to download highly-compressed digital audio files at near-CD quality. MPEG-1 is the name for the first phase of MPEG work; MPEG-1 audio consists of three operating modes called “Layers”, with increasing complexity and performance, named Layer-1, Layer-2 and Layer-3. Layer-3, with highest complexity, was designed to provide the highest sound quality at low bit-rates (around 128-kbit/s for a typical stereo signal). The MPEG-1 audio features are shown in Table2.4.

Table2.4 MPEG-1 audio features

	MPEG-1 audio
Channel	One or two
Sample rate	32, 44.1 or 48kHz
Bit rate	from 32kbps up to 448kbps
Samples per frame	384, 576 or 1152 samples

2.2.2 Bitstream Format

MP3 is based on frames; each frame consists of five parts: header, CRC, side information, main audio data and ancillary data. Fig 2.6[42] shows the data describing the structural factors of that frame; this data is called the frame's "header", and Fig. 2.7[42] shows the MP3 frame header represented visually, in the Table 2.5[42] shows the thirteen header files' characteristics. The header is 32 bit and first starts of with a 12-bit synchronization word to determine the beginning of each frame. Other bits are used to specify the parameters of the file, such as layer number, bit-rate and sampling frequency. Information such as copyright and original bits is not used by the decoder. The CRC is an optional feature added in during encoding process. The side information consists of information that is needed to decode the main data. The main audio data is shown in Fig. 2.8 which is decoded into audio samples. The ancillary data stores the user-defined information and not used by the decoder.

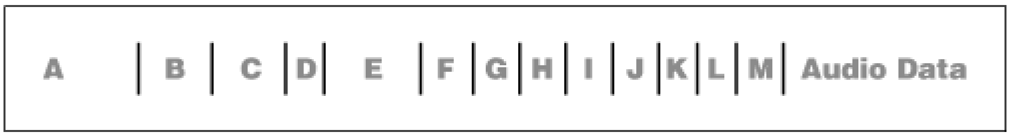


Fig. 2.6 MP3 frame header

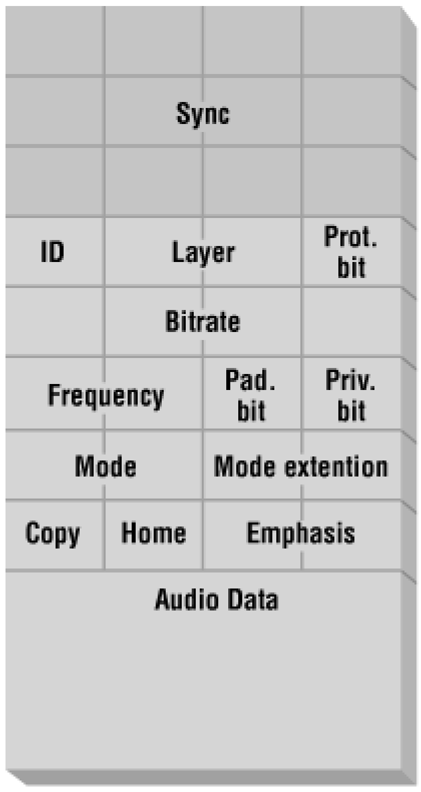


Fig. 2.7 MP3 frame header represented visually

Table 2.5 The thirteen header files' characteristics

Position	Purpose	Length (in Bits)
A	Frame sync	12
B	MPEG audio version (MPEG-1, 2, etc.)	2
C	MPEG layer (Layer I, II, III, etc.)	2
D	Protection (if on, then checksum follows header)	1
E	Bitrate index (lookup table used to specify bitrate for this MPEG version and layer)	4

F	Sampling rate frequency (44.1kHz, etc., determined by lookup table)	2
G	Padding bit (on or off, compensates for unfilled frames)	1
H	Private bit (on or off, allows for application-specific triggers)	1
I	Channel mode (stereo, joint stereo, dual channel, single channel)	2
J	Mode extension (used only with joint stereo, to conjoin channel data)	2
K	Copyright (on or off)	1
L	Original (off if copy of original, on if original)	1
M	Emphasis (respects emphasis bit in the original recording; now largely obsolete)	2

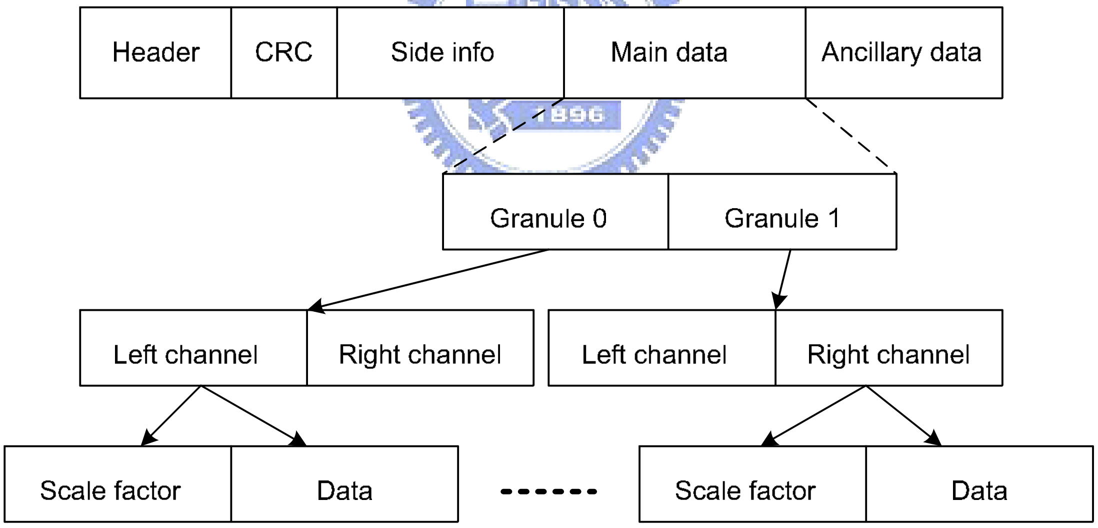


Fig. 2.8 Main data structure

2.2.3 Decoding Procedure

The decoding process is shown in Fig. 2.9[31]. We briefly describes the MP3 decoding flow.

- Synchronization: The purpose is to find out start of a frame, it is done by searching for the 12-bit synchronization word.
- Huffman decoding: Huffman encoding is a loss-less compression technique which presses according to the frequency of the input symbols occurring. The symbol that occurs most frequent is assigned a shorter code and vice versa.
- Requantization: The requantization process rescales the Huffman coded data back to the spectral values.
- Reordering: The purpose of reordering is to reorder the frequency lines in the granule to compensate for the reordering done in the encoding process.
- Joint Stereo Decoding: The purpose of joint stereo decoding is to convert the encoded stereo signal into left/right stereo signals and the information on which mode is used can be found in the header of each frame.
- Alias Reduction: The alias reduction tries to reduce the alias effects by merging the frequencies using the butterfly calculations.
- Inverse Modified Discrete Cosine Transform (IMDCT): The IMDCT transforms the frequency lines into time-domain samples.
- Frequency Inversion: This is done to compensate for the frequency inversion in the synthesis polyphase filterbank process.
- Synthesis Polyphase Filterbank: It transforms the 32 subbands of 18 time domain samples in each granule to 18 blocks of 32 PCM samples, which is the final decoding result.

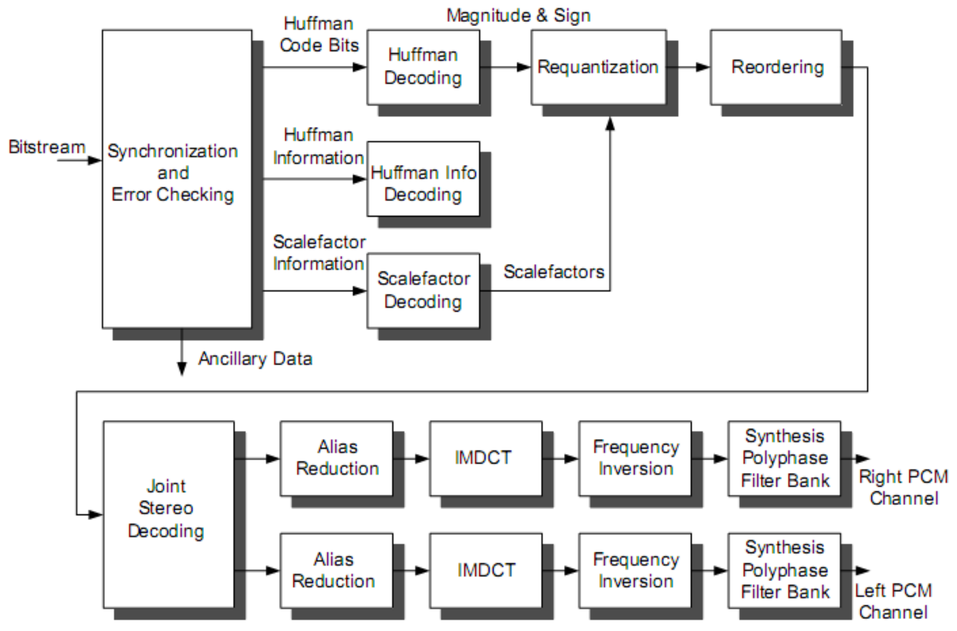


Fig. 2.9 Decoding process

2.3 Overview of Target Platform

This section describes some prerequisites related to RealView Platform Baseboard for ARM926EJ-S; Section 2.3.1 describes the major components on the platform system.

2.3.1 Major Platform Components

The proposed system adopts the components on platform are:

- ARM926EJ-S PXP Development Chip
  - — ARM926EJ-S processor core
  - — Multi-layer bus matrix that gives highly efficient simultaneous transfers
  - — Two external AHB master bridges and one external AHB slave bridge
  - — DMA controller
  - — Vectored Interrupt Controller (VIC)

- Field Programmable Gate-Array (FPGA)
- 128MB of 32-bit wide SDRAM
- 2MB of 32-bit wide static RAM
- 64MB of 32-bit wide NOR flash
- Programmable clock generators
- Connectors for keyboard, audio
- RealView Logic Tile
    - — Xilinx Virtex II FPGA
    - — configuration Programmable Logic Device (PLD) and flash memory for storing FPGA configurations
    - — Two 2MB ZBT SSRAM chips (these can be used, for example, to model IRAM and DRAM for an ARM core)
    - — clock generators and reset sources

The ARM926EJ-S PXP Development Chip is illustrated in Fig. 2.10[5].

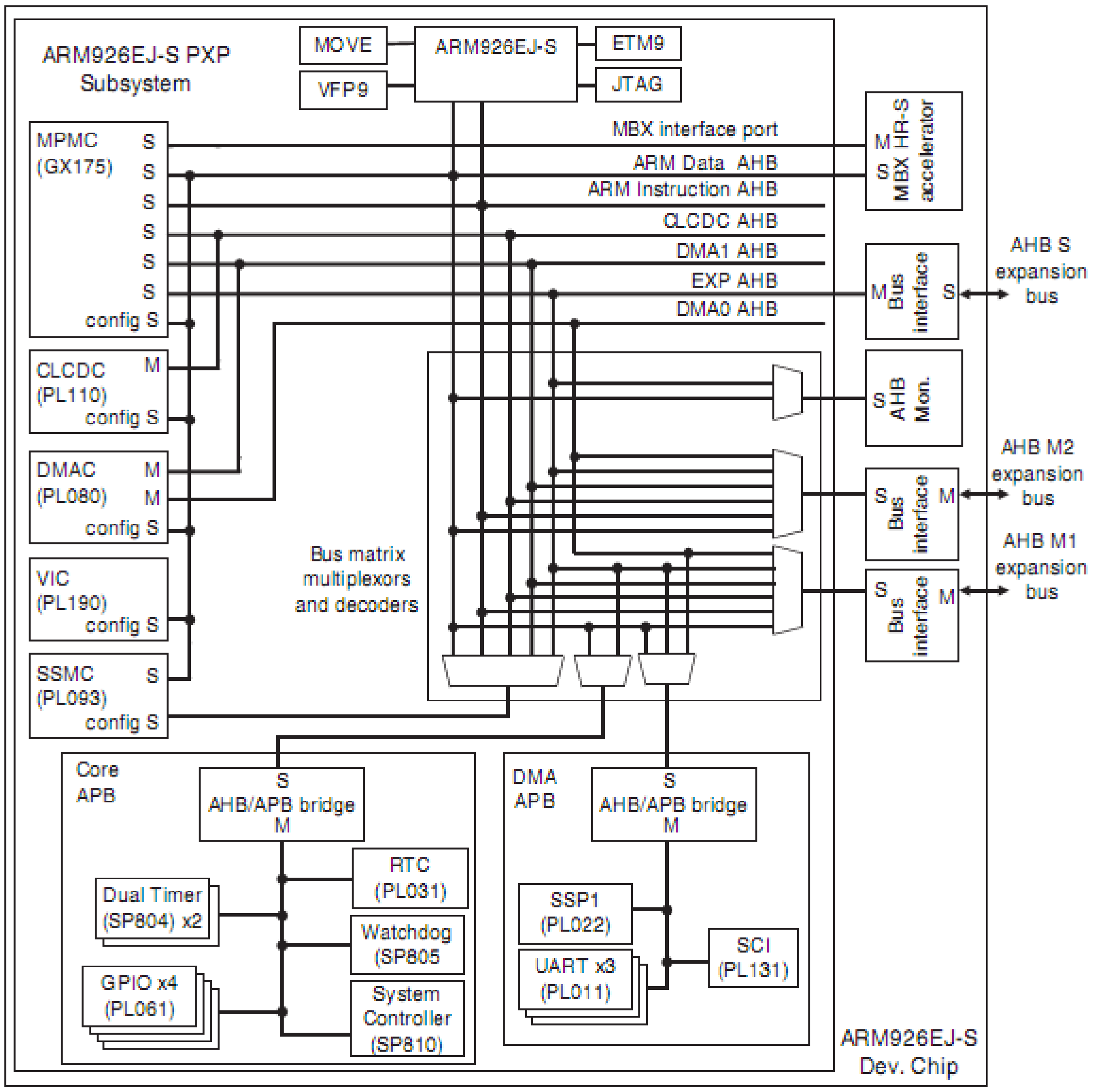


Fig. 2.10 ARM926EJ-S Development Chip block diagram

Chapter 3
ACARM9 Processor Core Design

This chapter describes the proposed processor core design. An ARM9E-like, 32-bit RISC embedded processor core is implemented in the register-transfer level with verilog language. Section 3.1 depicts the organization of the proposed processor core. Section 3.2 describes the implementation of multi-cycle multiplication. Section 3.3 describes the verification strategy, and Section 3.4 describes the synthesis result.

3.1 Organization of the Proposed Processor Core

The proposed Verilog RTL model consists of 13 major functional blocks including the decoder, register file (RF), barrel shifter (BS), arithmetic/logical unit (ALU), 32x32 multiplier (MUL), count leading zero (CLZ), forwarding unit, program counter (PC) selector, operand fetcher (OF), load/store data address generator (DAG), read/write data operation and control logic. The proposed Verilog RTL model is shown in Fig. 3.1.

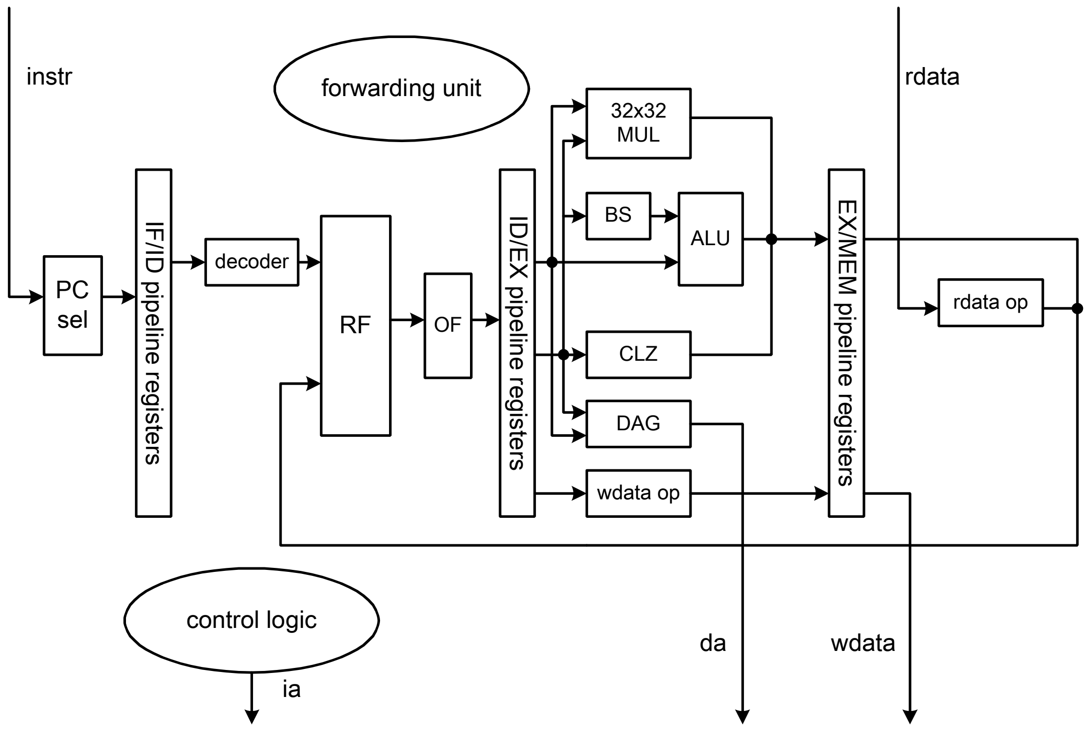


Fig. 3.1 Block diagram of the proposed Verilog RTL model

The decoder unit obtains the instruction from IF phase and decodes it to generate all the information needed for the other functional units.

The register file is composed of total 37 registers – 31 general-purpose 32-bit registers and six status registers.

The execution stage consists of a BS, an ALU, a CLZ, and a 32x32 multiplier. The arithmetic and logical operations are implemented with the ALU module whose second operand is received from the BS to perform shift operations if needed. The 32x32 multiplier produces a 32-bit product. More details of multi-cycle multiply operation will be described in Section 3.2.

The forwarding unit forwards the output data of EX stage to make sure that the next instruction can get these data as soon as possible.

The program counter selector chooses one valid address from five sources including the PC incrementer, the ALU output, LDM (load multiple)/STM (store multiple) output, read data and the interrupt as shown in Fig. 3.2.

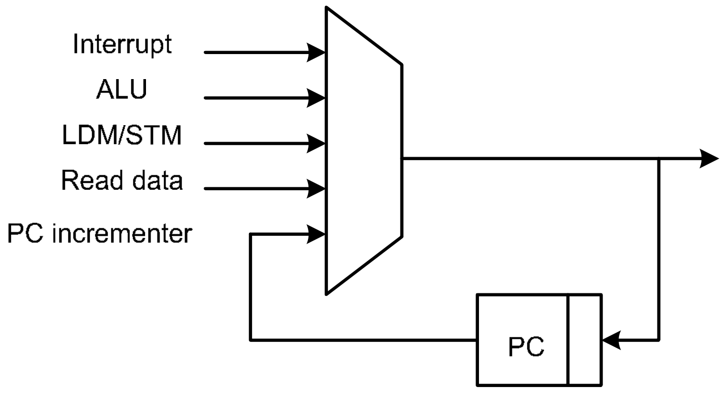


Fig. 3.2 Source selection of the address registers

The operand fetcher select data from register file, read data, ALU output, data address generator, immediate value or forwarding value as shown in Fig. 3.3.

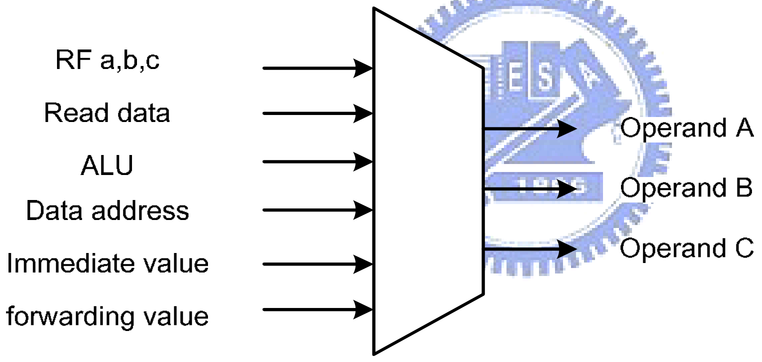


Fig. 3.3 Operand selection

The data address generator calculates the read/write data address on data bus includes the multiple load-store, and branch address on instruction bus.

The control logic controls all the data flows of combinational logic and all the state transitions of sequential logic.

The read/write data operation performs data alignment. As shown in Fig. 3.4, the read operation shifts byte data and halfword data to the bottom of a 32-bit register with zero-extended or sign-extended when the processor reads byte data or halfword data from memory. For writing halfword data to memory, the write operation copies the low halfword part to the high halfword part to fill the 32-bit data width. For writing byte data to memory, the write operation copies the least significant byte to the other three more significant bytes.

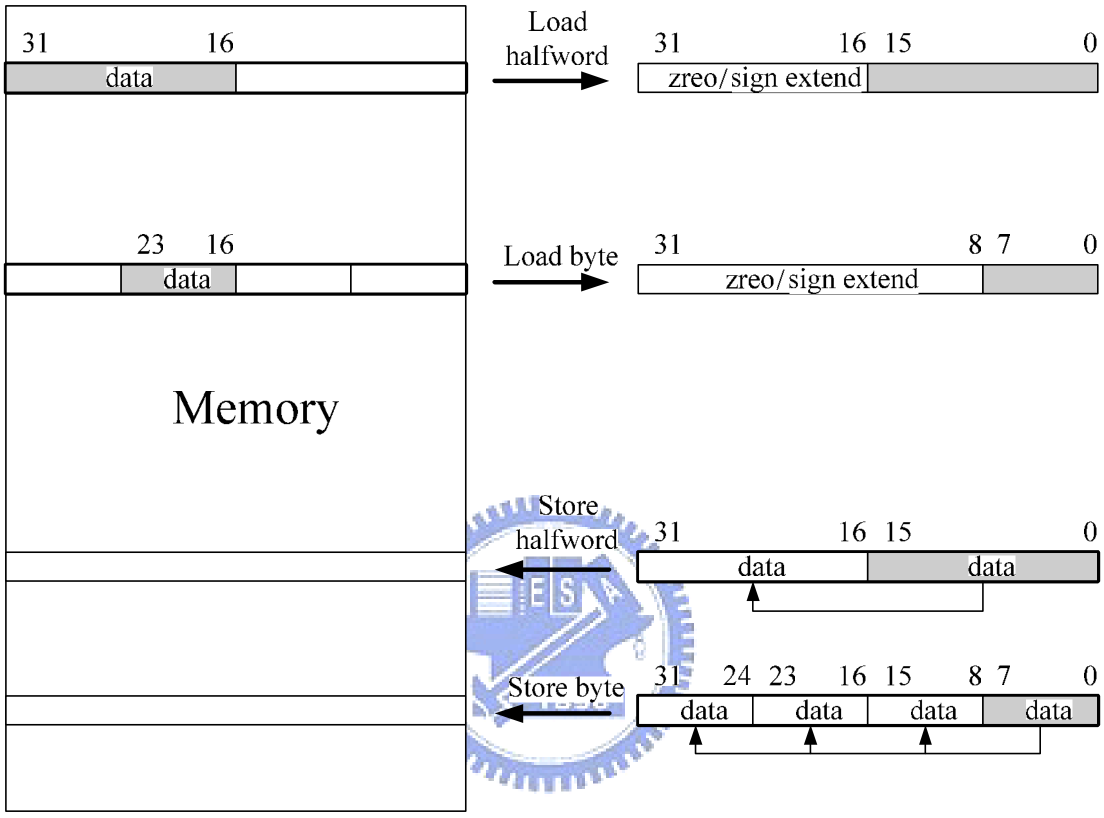


Fig. 3.4 Read/write operation

3.2 Multi-cycle Multiplication

For a multiply instruction, source operand A and source operand B are fed to the multiplier directly to perform multiply operation. Fig. 3.5 shows the multi-stage multiplication FSM.

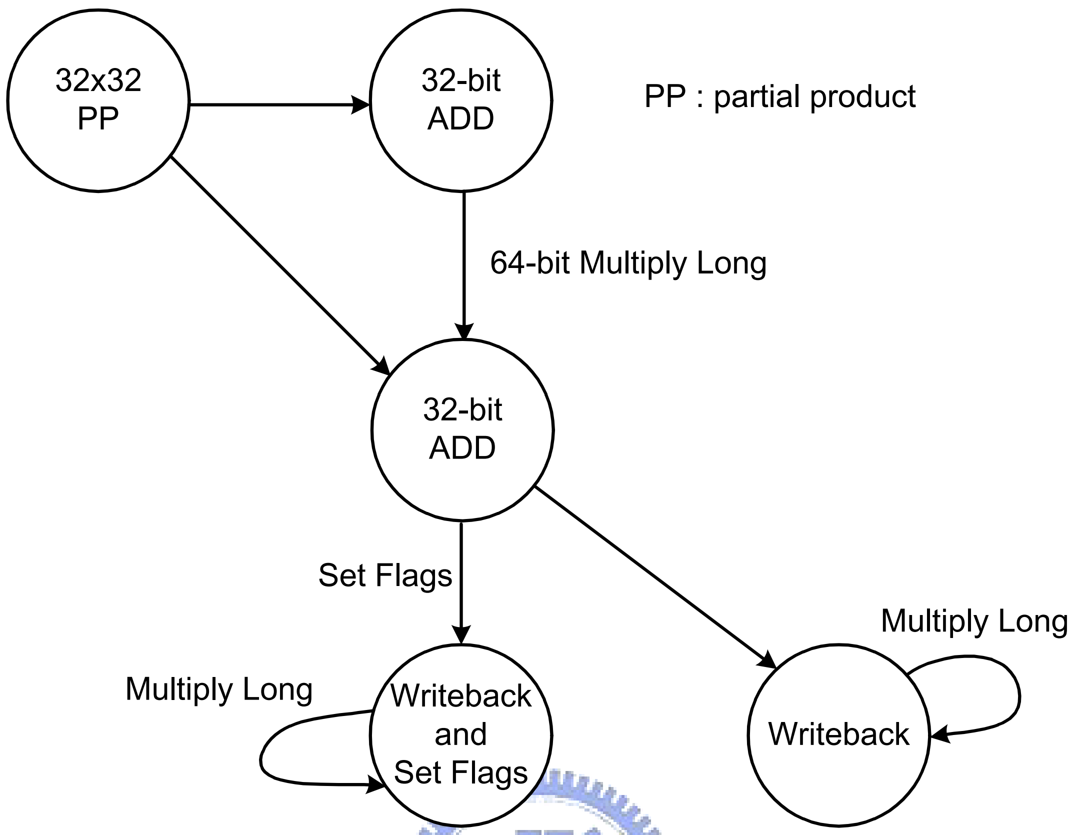


Fig. 3.5 Multiplication FSM of ARM

A normal multiply instruction with/without accumulation needs two cycles, since two 64-bit partial products are obtained in the first one cycle. The two low half 32-bit partial products and the value that is accumulated to the product are added by a carry save adder to generate two values, and then add the two values by a 32-bit adder to get final 32-bit result. While all 32-bit of product is valid, two cycles are needed. A long multiply instruction with/without accumulation needs three cycles, after the first cycle to obtain two 64-bit partial products, the 64-bit values need to take two cycles to perform 32-bit addition twice.

While the multi-cycle multiply operation is finished, a finish signal is sent from multiplication FSM to main FSM to tell that the multi-cycle operation is finished and the next instruction may get executed to continue the program flow.

3.3 Low-power Techniques

The proposed ACARM9 core design is convinced of ultra low-power characteristic and is implemented with some significant methods as follows. Section 3.3.1 presents gated unused registers, Section 3.3.2 describes gated unexecuted functional units, Section 3.3.3 describes the low-power MUX tree, Section 3.3.4 presents the low-power IP Fong-adder and Section 3.3.5 presents early flag checking for conditional instructions.

3.3.1 Gated Unused Registers

The data stored in registers are updated in every clock cycle without any clock gating operation; as a result, huge power is consumed and some manipulations must be done to avoid so much power wasting of the proposed design. One manipulation that makes new data to transmit into D port of the FLIP-FLOP only when the enable signal is high can be implemented in RTL level, as can be seen in the Fig.3.8. The disadvantage is that the stored data of registers is still re-written for holding old data every clock; as a matter of fact, it is still power wasting and other manipulations should be done. As can be seen in the Fig.3.8, a lower part with a CG cell composed of an enable signal controlled latch will make clock of the unused registers gated and guarantee that no data updated any more for holding old value. This method can absolutely shut down updating operations of unused registers and save unnecessary power consumption as much as it can.

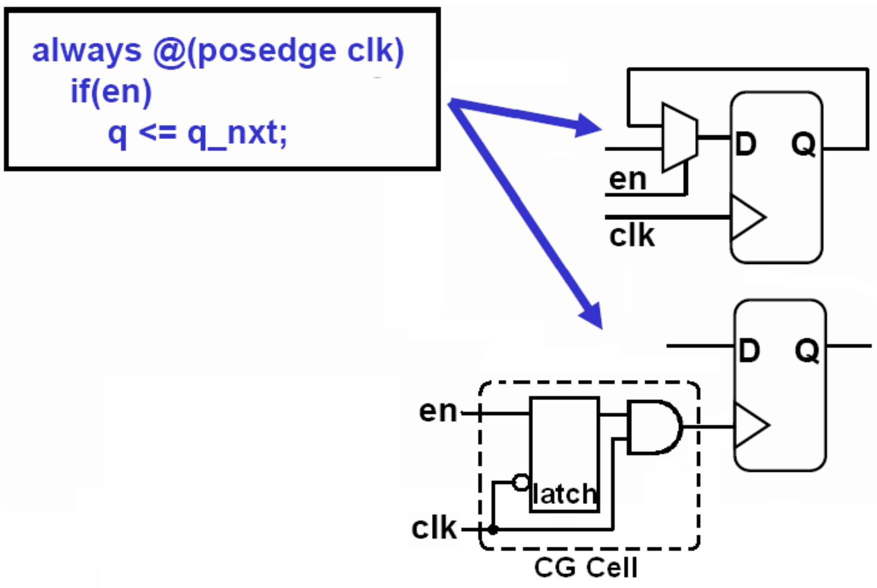


Fig. 3.6 Registers with clock gated by RTL and synthesized CG cell

3.3.2 Gated Unexecuted Functional Units

All the functional units in EX stage discussed in Section 3.1 and 3.2 are gated by registers. The operands of each functional unit are isolated and gated by registers. As shown in Fig. 3.7. When one functional unit is active, all the other functional units are gated by operand registers.

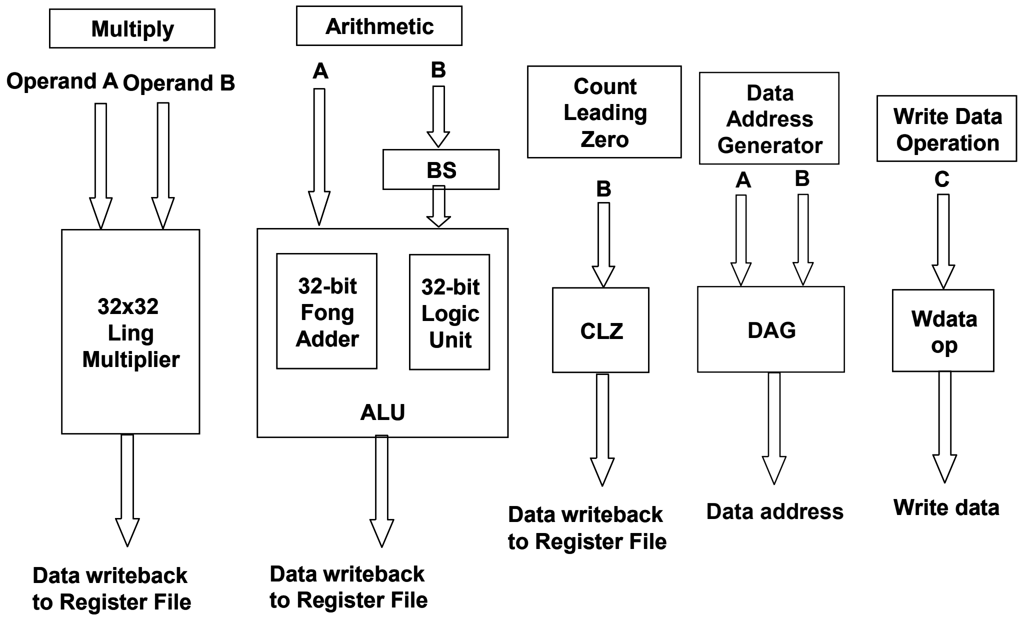


Fig. 3.7 Functional units in EX stage

3.3.3 Low-power MUX Tree

In this section, we introduce a method to reduce transitions of MUX tree of register files, the register files have 37 32-bit registers, and one or two register data will be selected for each instruction. The selection MUX tree consumes a great deal of power. An independent selection signal method is proposed in Fig. 3.8. The selection of MUX tree reduces the transitions of MUX tree.

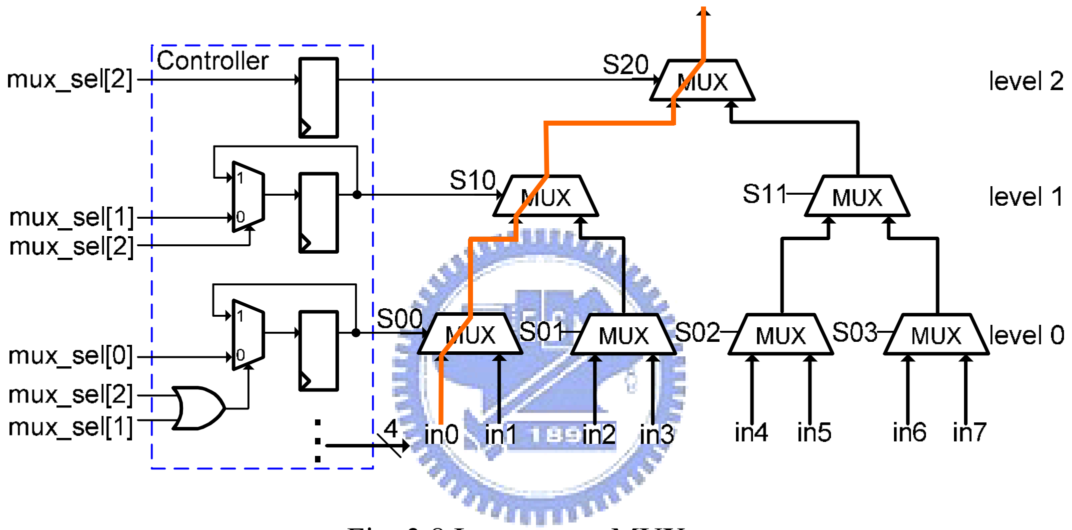


Fig. 3.8 Low-power MUX tree

3.3.4 Low-power IP Fong-adder

Fong-adder [40] is one of the most significant function units in the proposed core design due to its low-power consumption, and smaller area using with no increasing performance overhead.

As can be seen in the Table3.1, Fong-adder compares to Hybrid K-S Ling-adder [43]. The former saves 32.40% power as data width is 32-bit and 12.05% as data width is 64-bit. As can be seen in the Table3.2, the timing analysis of Fong-adder compares to Hybrid K-S Ling-adder is %2 better as data width is 32-bit and %6.56 better as data width is 64-bit; in addition, the area analysis of Fong-adder compares to

Hybrid K-S Ling-adder is also %21.40 better as data width is 32-bit and 17.72% better as data width is 64-bit. The proposed design implemented with Fong-adder has a characteristic of ultra-low power consumption and small area usage but still remains high-performance. In fact, this is one of the most outstanding advantages of the proposed core design.

Table 3.1 PDP analysis

Power Delay Product (PDP) Analysis (UMC 0.18um / TT corner)					
Data Width	Hybrid K-S Ling adder (power)	Fong-adder (power)	Hybrid K-S Ling adder (PDP)	Fong-adder (PDP)	PDP saving
32	18.98	12.83	18.98*1.02	12.83*1.00	**33.73%**
64	32.77	28.82	32.77*1.22	28.82*1.14	**17.82%**
Note : Power (mW)/PDP (pJ)					

Table 3.2 Timing and area analysis

Timing & Area Analysis (UMC 0.18um / TT corner)						
Data Width	Hybrid K-S Ling adder (Delay)	Fong-adder (Delay)	Timing Saving	Hybrid K-S Ling adder (Area)	Fong-adder (Area)	Area Saving
32	1.02	1.00	**2.00%**	14173.790	11140.114	**21.40%**
64	1.22	1.14	**6.56%**	28839.888	23730.583	**17.72%**
Note : Delay (ns) / Area (um2)						

3.3.5 Early Flag Checking

In Section 3.3.2 talks about gated unexecuted functional units. The method can be more improved by early flag check. When an instruction with setting flag is executed, the flag is also calculated and checked at EX stage. The flag checking at EX stage can save power if the next instruction is not executed. The control signals for gated operand registers are prepared before instruction enters EX stage.

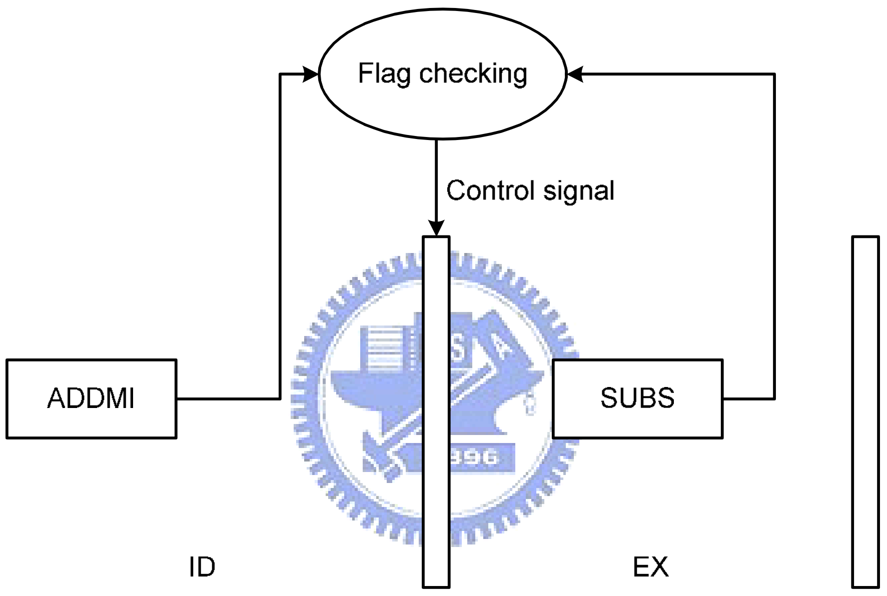


Fig. 3.9 Conditional check

3.4 Verification Strategy

A functional verification flow is proposed in this section. Section 3.4.1 proposes the overall functional verification flow. Section 3.4.2 proposes coding style checking by Linting; section 3.4.3 proposes a deterministic verification approach; Section 3.4.4 proposes an input-constrained random verification flow.

3.4.1 Functional Verification

The functional verification flow diagram is listed in Fig.3.10 and in each of steps the RTL model is verified with a systemC behavior model which is designed for matching all the cycle behaviors of ADS. All mismatches during the comparison will cause the flow back to RTL revision step for modification.

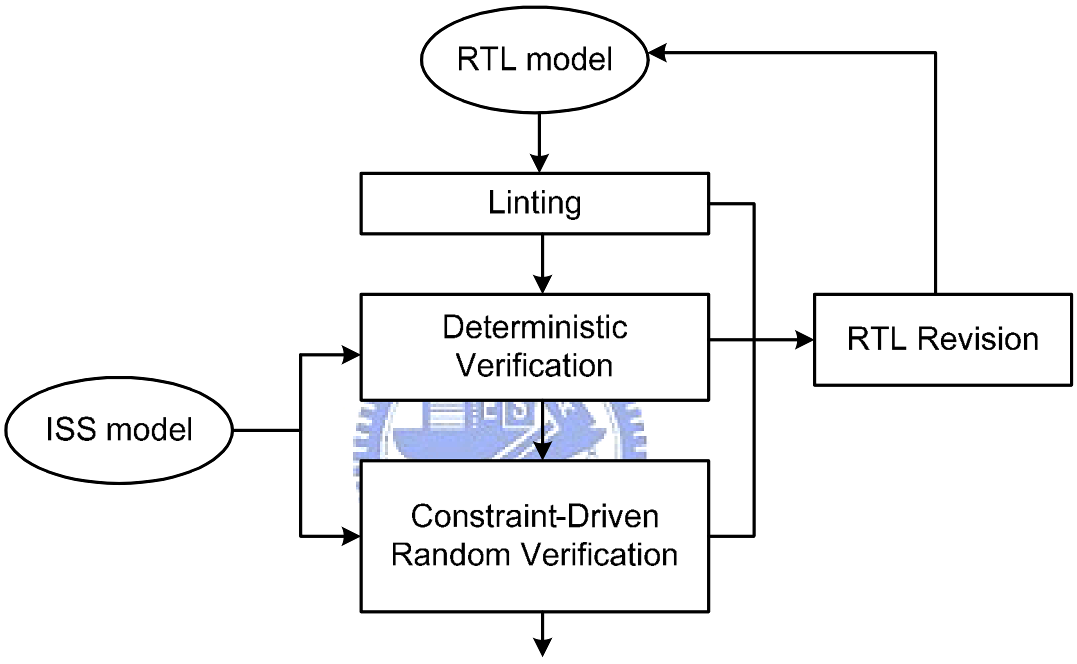

RTL model
Linting
Deterministic Verification
RTL Revision
ISS model
Constraint-Driven Random Verification

Fig. 3.10 Functional verification flow

3.4.2 Coding Style Checking by Linting Free

This section describes a static coding style check which improves the quality of the design in reuse and verification perspectives of RTL code. Checking the coding-style of the design by Novas nLint tool with 328 lint rules adopted from Frescale Semiconductor Reuse Standard (SRS) can avoid all kinds of warnings and errors including naming, synthesis, simulation, common syntax, undeclared objects, unexpected latches, DFT issue, and so on.

3.4.3 Deterministic Verification

This section describes deterministic verification which is made to check all regular cases and special corner case should be confronted with in the simulation phase. Deterministic verification is composed of three parts including specialized handcrafted pattern, real application pattern.

The handcrafted pattern is written in all cases of instructions implemented in the proposed design. It checks all results of instructions to be right in the first step in deterministic verification.

The real application patterns are implemented by some benchmarks like Dhrystone, Whetstone, and DSPstone. Moreover, a JPEG encoder and an MP3 decoder program are also provided to verify the correctness of the design. This kind of pattern checks real cases met in applications of real world and is essential in deterministic verification phase.

After the code coverage verification, it suggests that the test vectors applied to the design under verification are sufficient and the next verification step input-constrained random verification can be entered.

3.4.4 Input-constrained Random Verification

Input-constrained random verification is implemented by a constraint-driven random pattern generator which generates random pattern to both ISS model and RTL model and values of both models are compared cycle by cycle, as shown in Fig.3.11. This kind of verification is made to detect all of unexpected cases and the random pattern is constrained to the meaningful range to avoid undefined instructions generation. More simulation cycles are verified, more vector space is spanned by random patterns; therefore, fewer bugs exist in the design and more robust design can be declared.

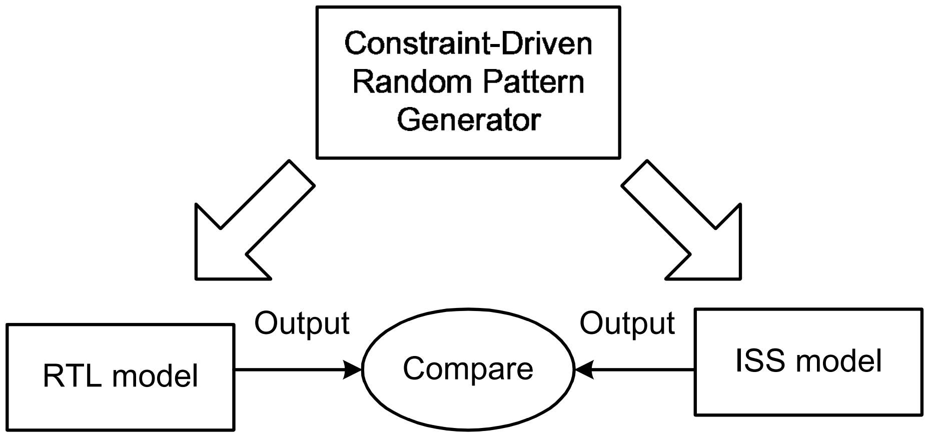


Fig. 3.11 Input-constrained random verification

3.5 Synthesis Result

Synthesis results are proposed in this section. Section 3.5.1 proposes comprehensive comparison with in-house ACARM7 and ARM. Section 3.5.2 proposes summaries of FPGA synthesis results.

3.5.1 Comprehensive Comparison

This section provides all experimental results with a comprehensive comparison compared among the proposed core ACARM9, in-house ACARM7, ARM946E-S without cache and ARM7TDMI.

All the information like timing, area, and power will be compared among the four different cores in Table 3.3. The ARM7TDMI-S and ARM946E-S are intellectual property (IP) of ARM. The ACARM7 and ACARM9 are in-house ARM-like IP. The processes of four cores are 0.18 um. The note of * is that all experiment results are measured in worst case condition. The * worst case condition is 0.18μm process - 1.62V, 125℃, slow silicon. The note of ** is that all experiment results are measured in typical case condition. The ** typical case condition is 0.18μm process - 1.8V, 25℃, typical silicon. The area calculation is in gate-count (TSMC 0.18um NAND2 equivalent).

Table 3.3 Comprehensive comparison between four different cores

Comprehensive comparisons between 4 cores				
	ARM7TDMI-S	ACARM7	ACARM9	ARM946E-S
Process (um)	0.18	0.18	0.18	0.18
Gate-count (k)	N/A	35.8	48.7	N/A
Area* (mm^2)	0.62	0.36	0.49	2.00
Performance* (MHz)	100	110	114	166
Power** (mW)	0.28	0.17	0.12	0.86
Pipeline stage	3	3	5	5
Other characteristics	Synthesizable; ARM v4T ISA	DFT support; ARM v4 ISA	DFT support; ARM v5E ISA	Synthesizable; ARM v5TE ISA
Note	All experiment results are measured in worse case except power estimation is in typical case			

3.5.2 Summaries of Synthesis Results

This section has analyzed the FPGA synthesis results of ACARM9. The device utilization summary and timing summary are shown in Table 3.4. The maximum frequency on FPGA is 29.914MHz. The frequency of ACARM9 which can run MP3 decoder real-time will be calculated in Chapter 5 and Chapter 6.

Table 3.4 ACARM9 FPGA synthesis results

ACARM9 FPGA synthesis results		
TOOL: Xilinx ISE 6.2i Device: Versatile LT-XC2V6000		
Device utilization summary:		
Number of Slices:	5633 out of 33792	16%
Number of Slice Flip Flops:	2560 out of 67584	3%
Number of 4 input LUTs:	10291 out of 67584	15%
Number of bonded IOBs:	401 out of 1104	36%
Number of TBUFs:	1 out of 16896	0%
Number of GCLKs:	2 out of 16	12%
Timing summary:		
Minimum period:	33.429ns	
Maximum Frequency:	29.914MHz	

Chapter 4
MP3 Platform-based System

This chapter describes the proposed MP3 Decoder System implemented by the ACARM9 processor core design which is implemented in the FPGA as a specific purpose processor and all system behaviors are controlled by an ARM926EJ-S processor on the develop board. Section 4.1 depicts the implementation of the system. Section 4.2 describes organization in system level. Section 4.3 describes organization from hardware perspective. Section 4.4 describes organization from software perspective.

4.1 Implementation

The development board of MP3 decoder system is implemented in a Versatile RealView Platform Baseboard for ARM926EJ-S; the proposed core ACARM9 is configured in an FPGA of Versatile LT-XC2V6000 (Xilinx VirtexII); The system software is programmed by a PC with an ARM® RealView MultiICE; All development environment is based on ARM Developer Suite (ADS) Version 1.2; The proposed design is then transformed to BIT-file to be configured in the FPGA by Xilinx Integrated Software Environment (ISE) 6.2i; the device drivers are ARM PrimeCell Vector Interrupt Controller (PL190); ARM PrimeCell Keyboard Mouse Controller (PL050); ARM PrimeCell Advanced Audio CODEC Interface (PL041); ARM PrimeCell DMA (PL080) and Character LCD display. All environment setups can be found in Table 4.1.

Table 4.1 The platform environment setup

MP3 Decoder System	
Development Board	Versatile RealView Platform Baseboard for ARM926EJ-S®
Logic Tile	Versatile LT-XC2V6000 (Xilinx VirtexII)
Multi-ICE	ARM® RealView MultiICE®
ADS	ARM® Developer Suite (ADS) Version 1.2
Xilinx ISE	Xilinx® Integrated Software Environment (ISE) 6.2i
Device Driver	ARM PrimeCell Vector Interrupt Controller (PL190)
	ARM PrimeCell Keyboard Mouse Controller (PL050)
	ARM PrimeCell Advanced Audio CODEC Interface (PL041)
	ARM PrimeCell DMA (PL080)
	Character LCD display

4.2 Organization in System Level

This section introduces the overview of the proposed system in system level view. Section 4.2.1 describes major components of the proposed system and Section 4.2.2 describes the entire control flow of the proposed system.

4.2.1 Major Components

This section describes major components of the proposed MP3 decoder system, which is composed of an ARM926EJ-S core, a 128 SDRAM, a 64MB NOR flash, a 2MB SSRAM, device drivers, FPGA on development board and a logic tile. The ARM926EJ-S core controls all behaviors and components of the entire system; the data in the SSRAM can be loaded or stored by the core; the 64MB NOR flash is a nonvolatile storage and bitstream or programs can be stored in the flash prepared to be loaded into SDRAM automatically right after power-up. A logic tile includes an FPGA, two 2MB ZBTSRAMs (Zero Bus Turnaround SRAM), a keyboard interface which is used for user interface controller of the system, DMAC (Direct Memory Access Controller) and VIC (Vectored Interrupt Controller) which are used for audio play, and many other devices and components. All components discussed above are shown in Fig 4.1

This section has described major components of the proposed system and more details of control flow in system level of proposed system will be discussed in the next section.

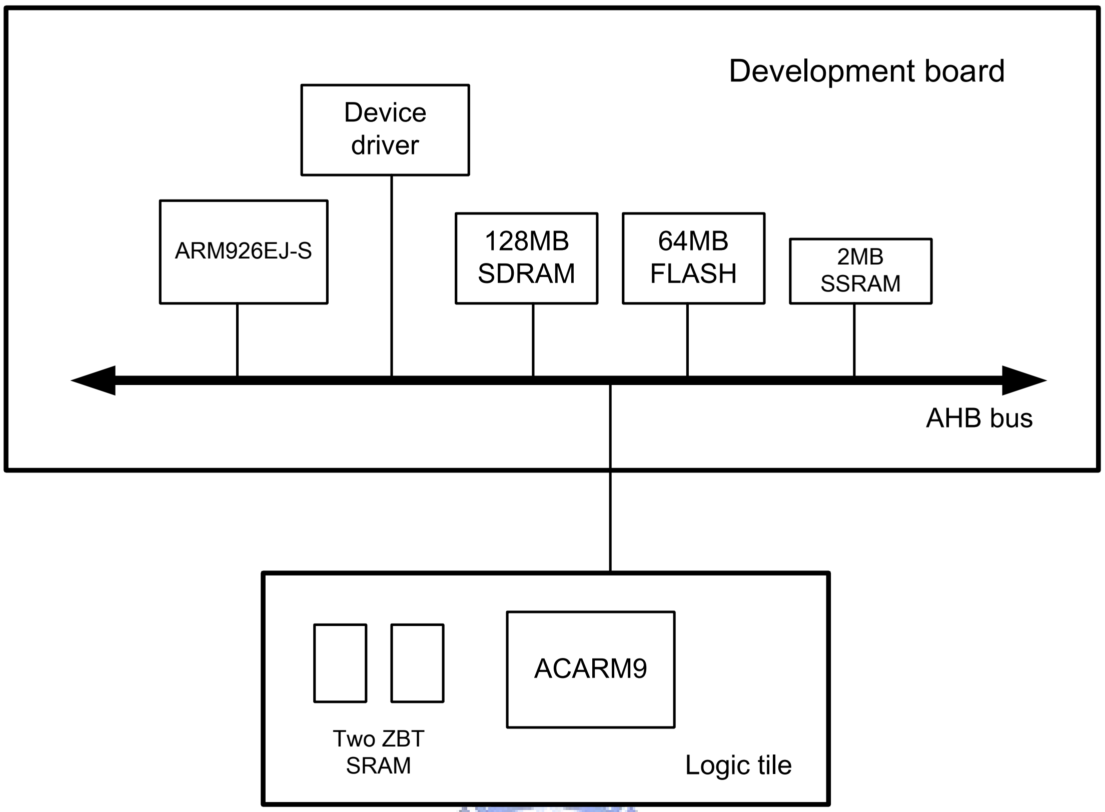


Fig. 4.1 Block diagram in system level

4.2.2 Control Flow in System Level

This section describes the control flow of the proposed system. First, all the software binary data are obtained from ADS tool and is written into the NOR flash in advance by a multi-ICE with semi-hosting mechanism, the software binary data include boot loader, MP3 system control program and MP3 decoder program for ACARM9. The MP3 bitstream is also written into the NOR-flash, the write address to the NOR-flash is also designated at the same time. Second, configure the booting program, after booting from development board, the MP3 system control program is loaded into SDRAM and executed by the ARM926EJ-S core in this step. Third, the MP3 system control program initializes all hardware drivers, including Vectored Interrupt Controller, Secondary Vectored Interrupt Controller, Advanced Audio

CODEC Interface, PS2 Keyboard/Mouse Interface, Real Time Clock Controller, Direct Memory Access Controller and Character LCD Display. Fourth, the MP3 system control program sets the address of MP3 bitstream, and then the MP3 decoder program is loaded from NOR-flash into two ZBTSRAM on the logic tile, in this step, the program of MP3 decoder is loaded into both ZBTSRAM on the logic tile and part of the encoded MP3 bitstream file are also loaded into the ZBTSRAM, therefore, both program and encoded bitstream are prepared for ACARM9 configured in FPGA inside the logic tile. Fifth, after all the initialization is done, ARM926EJ-S sends a signal to tell ACARM9 to begin to decode the bitstream in the ZBTSRAM. Sixth, once a frame is completed decoded, ACARM9 sends out an interrupt, ARM926EJ-S switches to the interrupt service routine; ARM926EJ-S moves the audio data from the ZBTSRAM on the logic tile to SSRAM on the development board and the new bitstream from NOR-flash into the ZBTSRAM for the next frame to be decoded; the audio data is moved to Advanced Audio CODEC by Direct Memory Access Controller, then the audio data can be played by Advanced Audio CODEC. Seventh, while ACARM9 finishes decoding procedure, it will send a finish signal to tell ARM926EJ-S, and then the MP3 control program can restart from next bitstream.

The major steps of control flow in system level are illustrated in Fig. 4.2. These are primary steps of system control flow and more delicate and efficient control techniques will be discussed in Section 4.4, Organization in Software Level. On the other hand, more details about the proposed core wrapped with an AHB interface will be discussed in Section 4.3, Organization in Hardware Level.

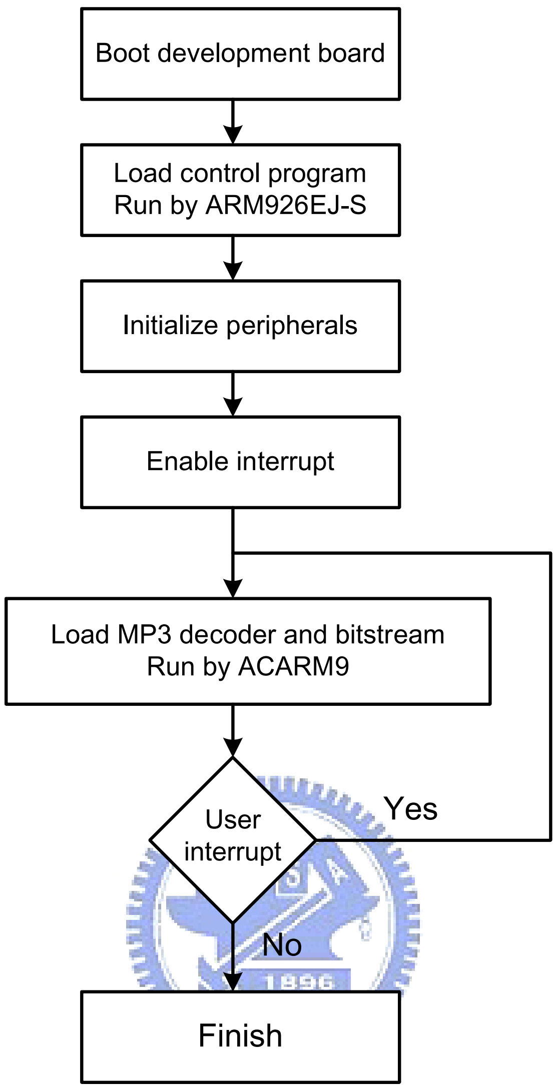


Fig 4.2 Control flow in system level

4.3 Organization in Hardware Level

This section describes hardware architecture of the proposed core wrapped with an AMBA bus interface in FPGA of the proposed system. Section 4.3.1 describes all the AHB peripherals in FPGA. Section 4.3.2 describes the core wrapped with AHB bus interface. Section 4.3.3 describes the implementation of interrupt logic. Section 4.3.4 describes control flow within AHB wrapper for ACARM9 in FPGA.

4.3.1 AHB Peripherals in FPGA

This section describes all the AHB peripherals in FPGA, as shown in Fig. 4.3. The ACARM9 should be wrapped to comply with the AHB specification in advance before it is configured into an FPGA or will never be used and activated in the target system.

As shown in Fig. 4.3, the AHB top-level block is the top level HDL (Hardware Description Language) configured in the FPGA and instantiates and interconnects the main block in the system; the AHB-APB System block includes the bridge from AHB to APB and all the APB peripherals including LED light and interrupt controller. On the other hand, the AHB Decoder block decodes the received address from the AHB Bus to different IP selection signals which are used to activate the corresponded IP to wake up and execute; the AHB Multiplexer block selects the right answer from all different results and selection is made by the IP selection signal generated by AHB Decoder. Furthermore, the AHB-Wrapper for ACARM9 block wraps ACARM9 with AHB-wrapper and includes three sub-blocks which are ACARM9 with AHB interface, and two ZBTSRAM controllers. ACARM9 with AHB Interface block will be discussed in Section 4.3.2 in detail. Both ZBTSRAM Controllers control all the data movements of the two ZBTSRAMs on the logic tile.

This section has discussed all the AHB peripherals in FPGA, and more details about core wrapped with AHB bus interface will be described in the next section.

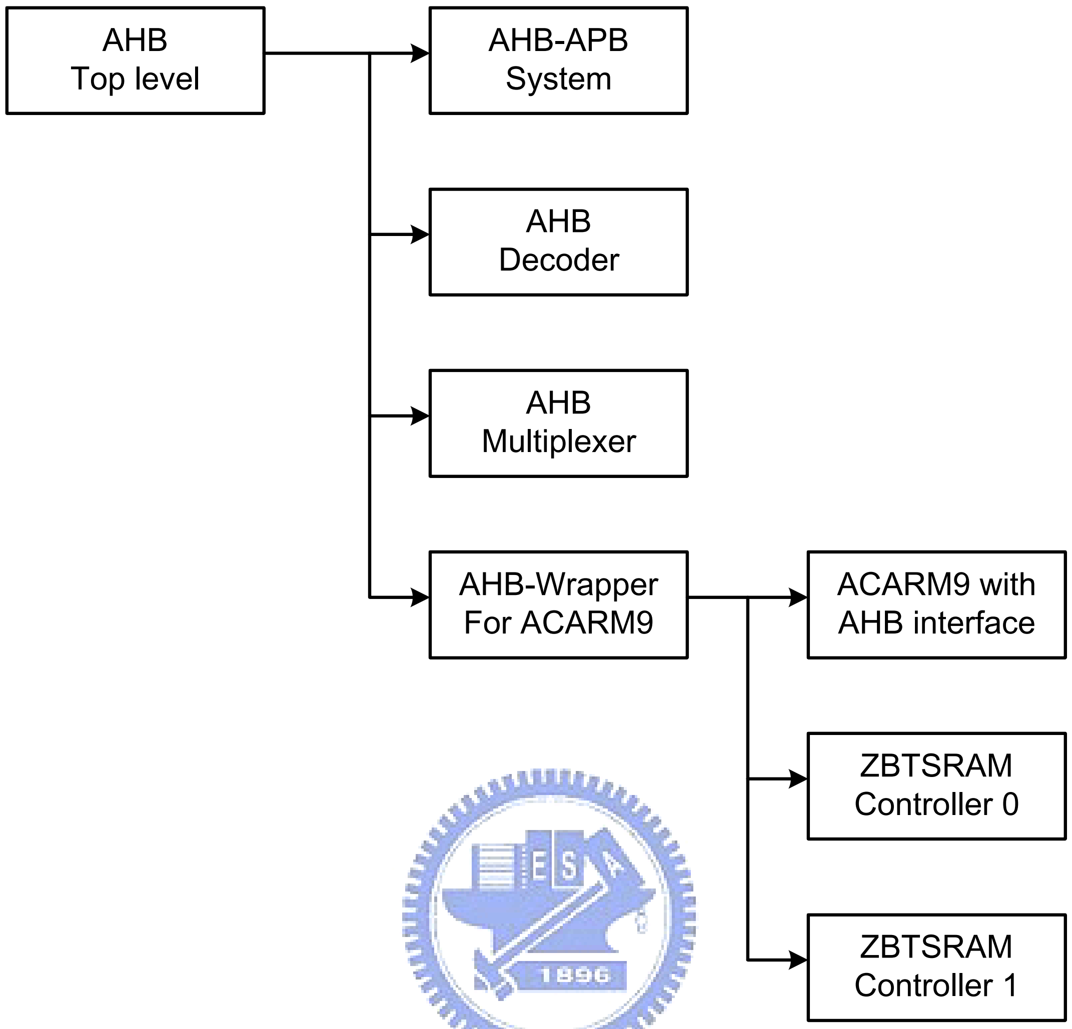


Fig. 4.3 AHB peripherals configured in the FPGA

4.3.2 Core Wrapper of AHB Bus Interface

This section discusses the core wrapped with AHB bus interface in FPGA. As can be seen in Fig.4.4, the FSMs of the AHB-interface block control all the behaviors between AHB Bus and ACARM9. As the bus in the Idle State which could be due to decoding complete or not valid operation, no tasks are executed in this cycle and other tasks will be executed in other non-idle states. As the FSM jumps into the Write State which is due to writing task requested by ARM926EJ-S , the data from AHB Bus will be written into Current Status Registers (CSR) which controls behaviors of the bus; on the other hand, the information of CSRs of ACARM9 will be read out to put on

AHB Bus as the bus jumps into the Read State which is due to reading task requested by ARM926EJ-S; for example, as ACARM9 finishes a decoding task, a termination signal will be put on the FINISH register of CSRs. The contents of FINISH register of CSRs will be read out to put on the AHB Bus to tell ARM926EJ-S that the decoding task has done as the bus is in Read State. While ARM926EJ-S sends a request to tell ACARM9 to execute the decoding task, the Pre-Run State will be entered. The action executed by ACARM9 in Pre-Run State is to reset the bus first to clear all internal registers of the bus to zero and assures the correctness of ACARM9's execution before entering into Run-State every time. ACARM9 does decoding procedure in the Run State and stays in the state until it finishes the decoding task. As can be seen in Fig. 4.4, Run-State goes after Pre-Run State continuously. This section has discussed the AHB bus interface of the core, ACARM9, with AHB-wrapper in FPGA and more details about control flow within AHB peripherals in FPGA will be described in next section.

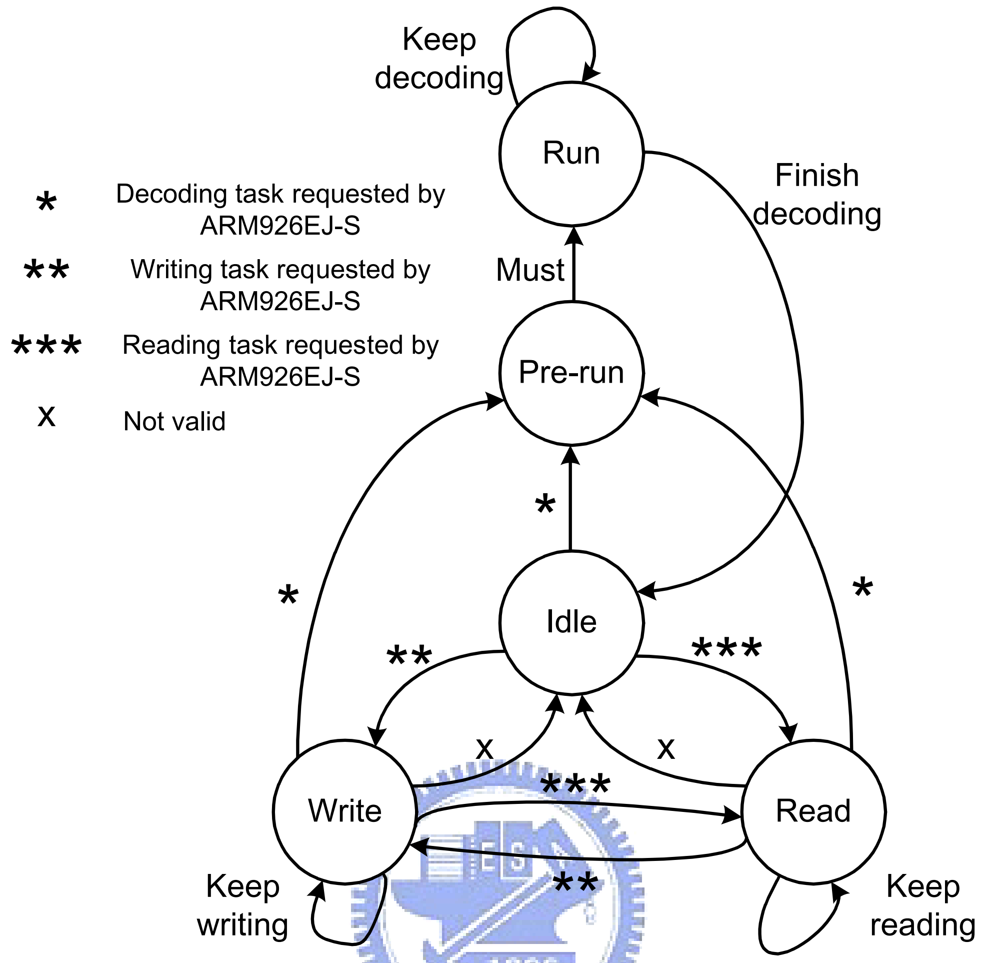

Keep decoding
Run
Finish decoding
* Decoding task requested by ARM926EJ-S
** Writing task requested by ARM926EJ-S
*** Reading task requested by ARM926EJ-S
x Not valid
Must
Pre-run
*
*
*
Idle
**
***
x
x
***
Write
Read
**
Keep writing
Keep reading

Fig. 4.4 FSMs of the AHB bus interface

4.3.3 Implementation of Interrupt Logic

The PrimeCell Vectored Interrupt Controller (VIC) which is shown in Fig. 4.5[5] is an Advanced Microcontroller Bus Architecture (AMBA) compliant, System-on-Chip (SoC) peripheral that is developed, tested, and licensed by ARM.

The PrimeCell VIC provides an interface to the interrupt system, and improves interrupt latency in two ways:

- moves the interrupt controller to the AMBA AHB bus, and
- provides vectored interrupt support for high-priority interrupt sources.

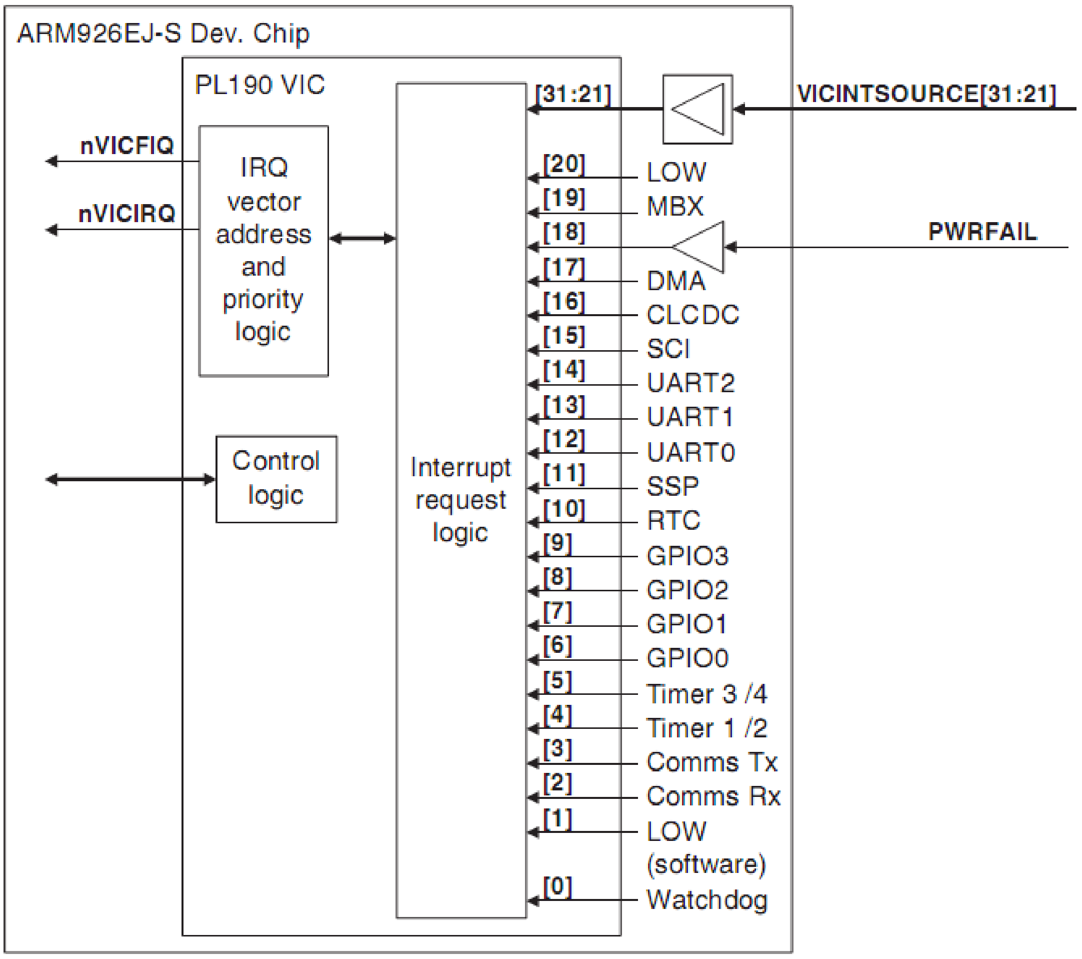


Fig. 4.5 VIC block diagram

The interrupt request logic receives the interrupt requests from the peripheral and combines them with the software interrupt requests. It then masks out the interrupt requests that are not enabled, and routes the enabled interrupt requests to either IRQ or FIQ. Fig. 4.6[5] shows a block diagram of the interrupt request logic.

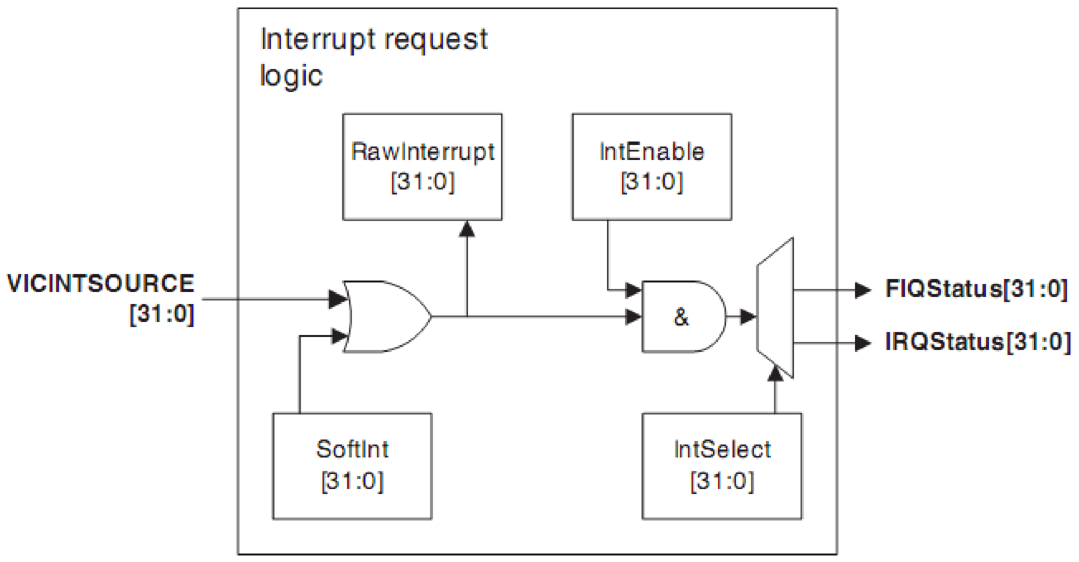


Fig. 4.6 Interrupt request logic

The interrupt request logic is designed in the ACARM9 core wrapper, the interrupt is enabled when ACARM9 finishes each decoding frame, the ACARM9 holds and waits for the interrupt service routine of ARM926EJ-S, the ARM926EJ-S moves the decoded audio data from ZBTSRAM to SSRAM and feeds the MP3 bitstream to ZBTSRAM for decoding, after the interrupt service routine, ACARM9 is re-enabled to decode the next frame.

4.3.4 Control Flow within AHB Wrapper for ACARM9 in FPGA

This section discusses the control flow within the block of AHB wrapper for ACARM9, as shown in Fig. 4.3. As in Fig. 4.7, the block AHB wrapper for ACARM9 is as the same one in Fig.4.3 and the block AHB Interface is the one discussed in Section 4.3.2. ARM926EJ-S, one SDRAM with 128MB, and two ZBTSRAM with 2MB are also displayed in Fig. 4.7.

Both ZBTSRAM in the logic tile can be used by either ARM926EJ-S or ACARM9 via the AHB wrapper. The current status registers (CSR) can control ACARM9 to execute or halt. As in Fig. 4.8, the memory map of a ZBTSRAM is divided into three parts which include IM (Instruction Memory), DM (Data Memory), and decoded audio data. The content of the first part comes from NOR-flash auto-loading as the whole system powers up. On the other hand, ARM926EJ-S requests the un-decoded MP3 bitstream moved from the NOR-flash to the second part of the ZBTSRAM as a DM. Therefore, the necessary information with both instruction and data are prepared for ACARM9 and ACARM9 can use the ZBTSRAM to execute decoding procedure. ZBTSRAM Controller is an essential part to access ZBTSRAM and needs all information received from a processor. As a result, a multiplexer is implemented and selects which processor possesses the ZBTSRAM. After decoding procedure by ACARM9 finishes, ARM926EJ-S will take over the ZBTSRAM at the moment and moves the decoded audio data to the SSRAM for playback on audio CODEC later.

This section has described the control flow within AHB wrapper for ACARM9 and organization in software level will be described in next section.

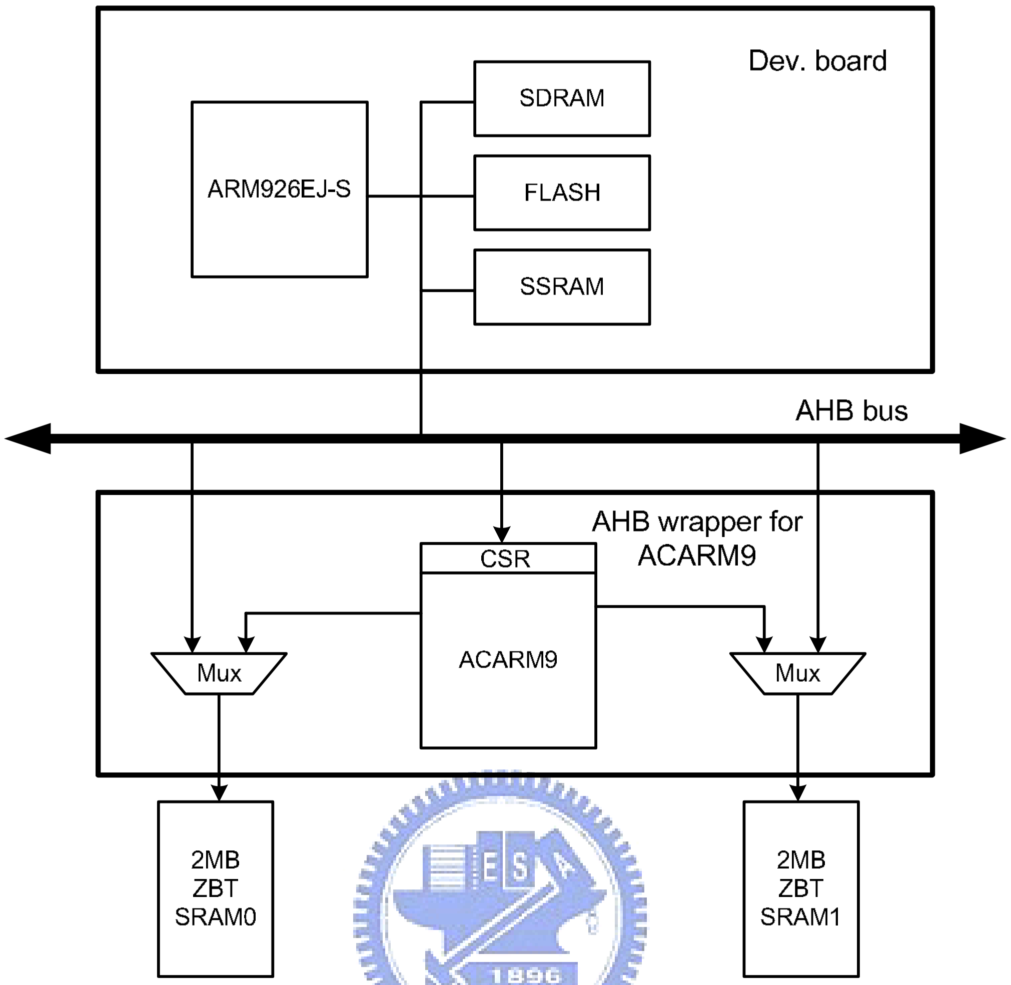


Fig. 4.7 AHB wrapper for ACARM9 (with ARM926EJ-S)

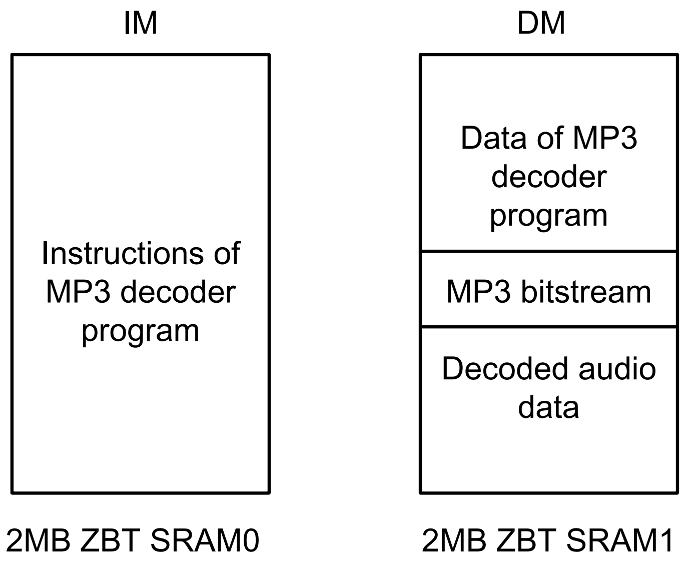

IM
DM
Instructions of MP3 decoder program
Data of MP3 decoder program
MP3 bitstream
Decoded audio data
2MB ZBT SRAM0
2MB ZBT SRAM1

Fig. 4.8 ZBTSRAM memory map

4.4 Organization in Software Level

This section describes all the software level design of the proposed MP3 decoder system. Section 4.4.1 describes the interrupt mechanism of the MP3 decoder system. Section 4.4.2 depicts the processor utilization of the ARM926EJ-S on platform baseboard and ACARM9. Section 4.4.3 describes the real-time synchronization of the platform-based MP3 decoder system.

4.4.1 Interrupt Mechanism

This section describes the interrupt mechanism, to introduce the vector interrupt controller and proposed interrupt priority list.

The ARM926EJ-S PXP Development Chip contains the primary interrupt controller and a secondary interrupt controller in the FPGA, as shown in Fig. 4.9[5]. The primary interrupt controller manages interrupts from internal devices and provides 11 pins for use by the external secondary interrupt controller and multiplexor presented in the FPGA. VICINTSOURCE31 is the output from the secondary controller. VICINTSOURCE [30:21] can be driven from individual interrupt signals from peripherals in the FPGA or on a RealView Logic Tile.

The MP3 decoder system adopts three vector interrupts; the VICINTSOURCE31 is the keyboard interface interrupt pin which is generated from secondary interrupt controller and connected to pin 31; the DMA interrupt is pin 17 from primary interrupt controller; the logic tile can generate an interrupt to VICINTSOURCE29 which is connected to pin 29. All the three interrupts are IRQ interrupt.

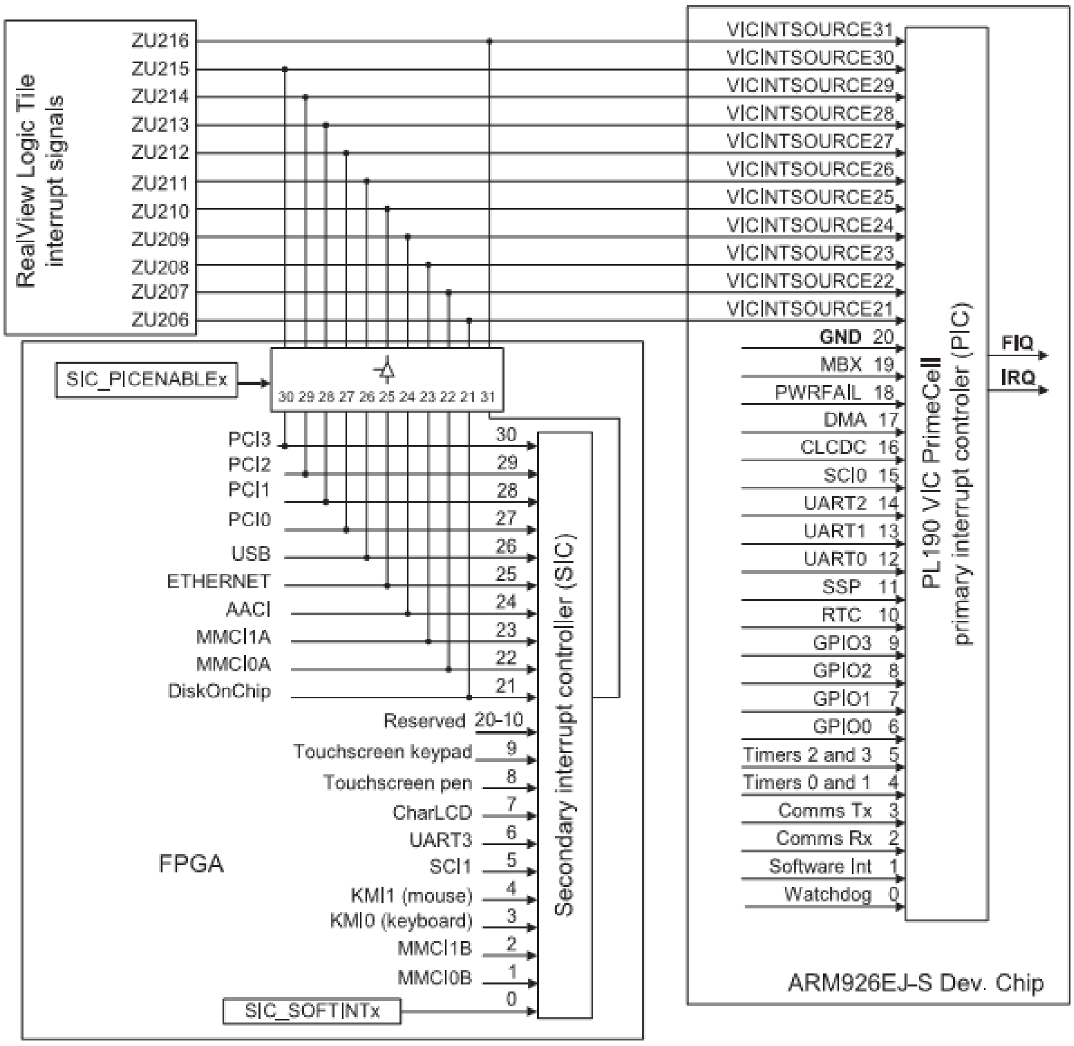


Fig. 4.9 External and internal interrupt sources

The interrupt priority block prioritizes the following requests:

- non-vectored interrupt requests,
- vectored interrupt requests, and
- external interrupt requests.

The highest-priority request generates an IRQ interrupt if the interrupt is not currently being serviced. Fig. 4.10[5] shows a block diagram of the interrupt priority logic.

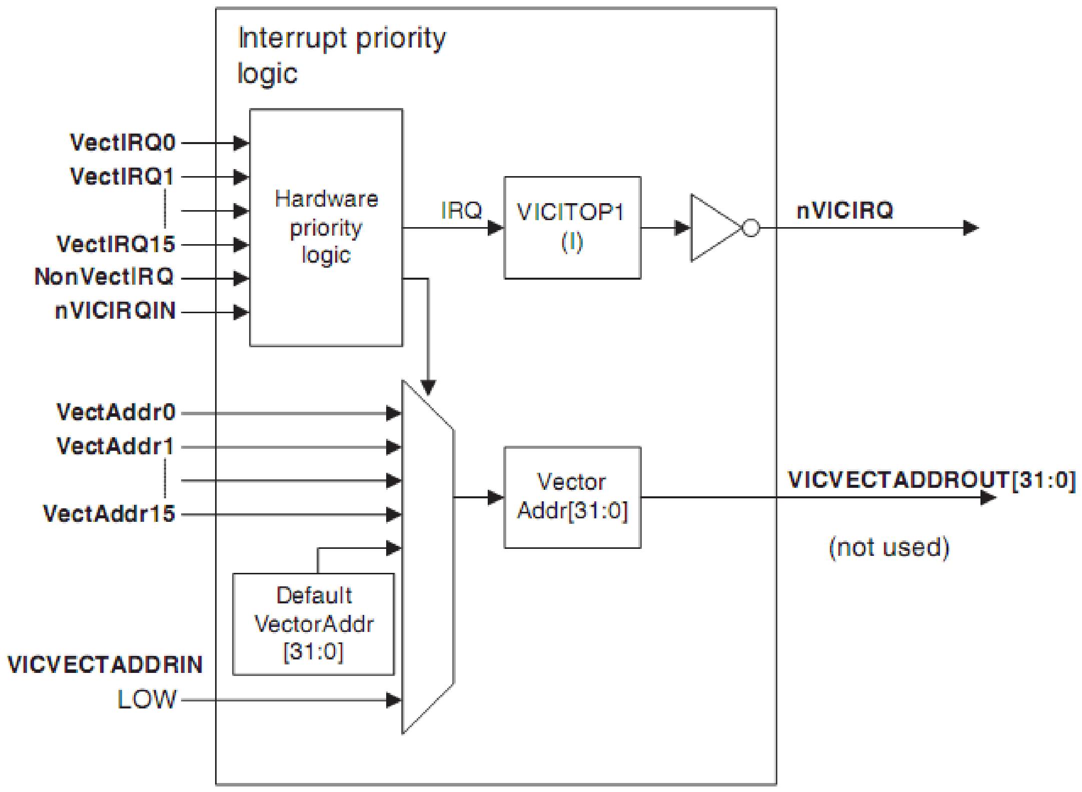


Fig. 4.10 Interrupt priority logic

A vectored interrupt is only generated if the following are true:

- it is enabled in the interrupt enable register, VICIntEnable,
- it is set to generate an IRQ interrupt in the interrupt select register, VICIntSelect, or
- it is enabled in the relevant vector control register, VICVectCntl0-VICVectCntl15.

This prevents multiple interrupts from being generated by a single interrupt request if the controller is incorrectly programmed.

The software can control the source interrupt lines to generate software interrupts. These interrupts are generated before interrupt masking, in the same way as external source interrupts. Software interrupts are cleared by writing to the software interrupt clear register, VICSoftIntClear (see Interrupt control registers). This is normally done at the end of the interrupt service routine.

The interrupt priority is listed in Table 4.2. The highest priority is keyboard interface interrupt which provides users to shuffle the songs and the next is DMA controller which deals with the movement of memory, the lowest priority is logic tile which reveals that ACARM9 has finished decoding.

Table 4.2 Interrupt priority list

Priority	Interrupt source	Pin number
0	SVIC (keyboard)	31
1	DMA	17
2	Logic tile (ACARM9)	29

4.4.2 Processor Utilization

This section describes the ARM926EJ-S and ACARM9 utilization. The ARM926EJ-S as shown in Fig. 4.11[5] is in charge of the system control flow, including booting, device driver setup, handling the interrupt service routine, controlling the ACARM9, data transferring between logic tile and development baseboard, and audio play. The ACARM9 is simply like MP3 decoder hardware, the only responsibility is to decode the MP3 bitstream.

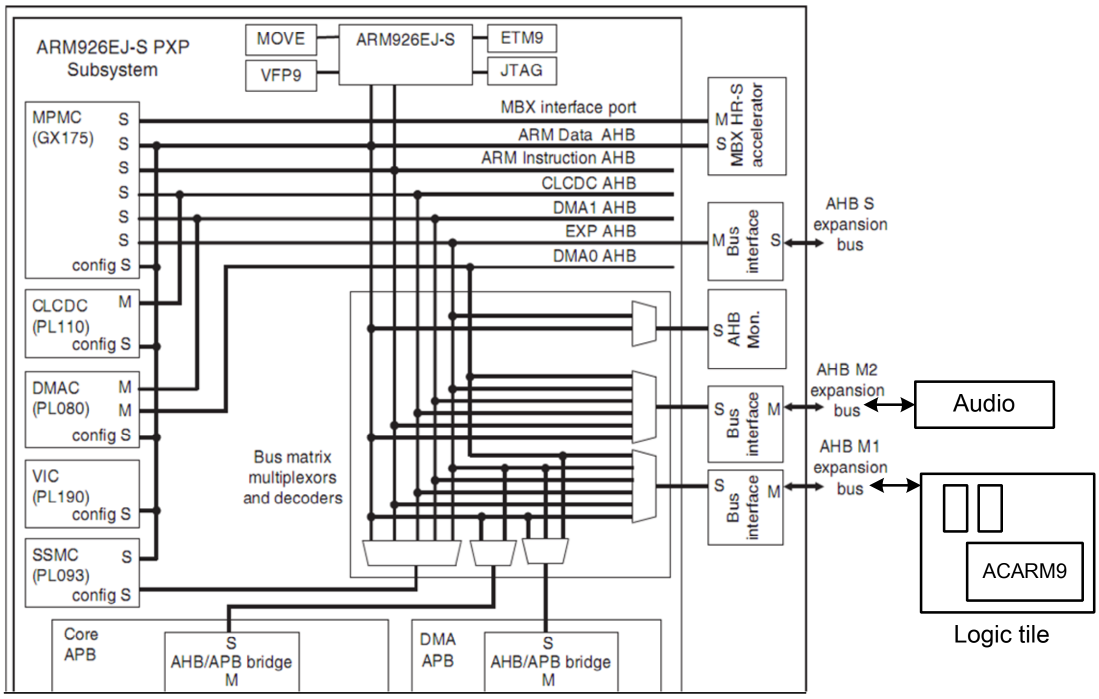


Fig. 4.11 ARM926EJ-S PXP Development Chip block diagram

The processor utilization is shown in Fig. 4.12. ARM926EJ-S on development board is to control the system flow and deal with interrupt service routine. The entire control program takes little execution time. The logic tile ISR includes the control of DMAC and data transfer. The DMA ISR tells ARM926EJ-S that DMAC transfer has completed. ARM926EJ-S re-enables ACARM9 for the next frame. ACARM9 is enabled for decoding by ARM926EJ-S and is halted after decoding one frame.

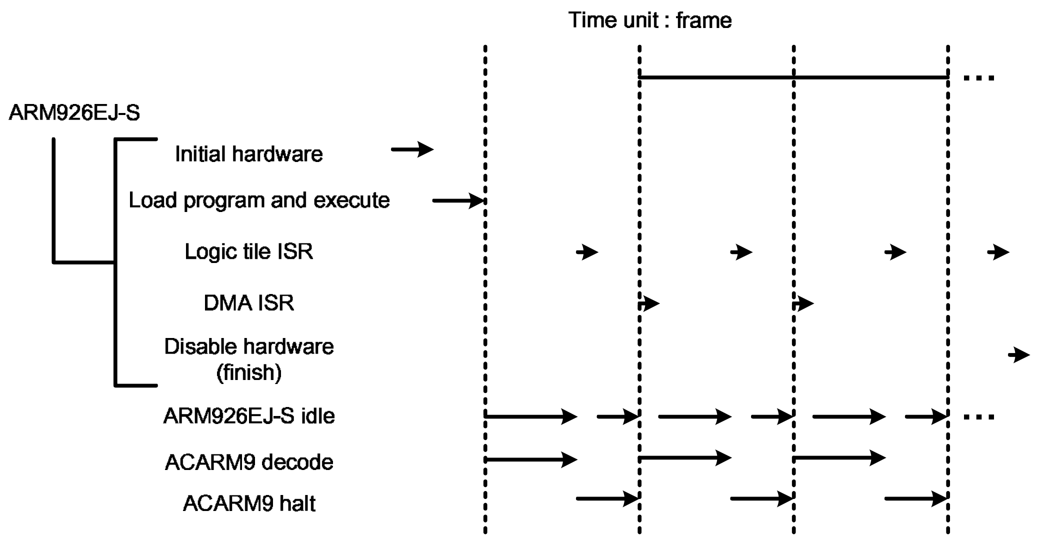


Fig. 4.12 Processor utilization scheme

The ZBTSRAM utilization is shown in Fig. 4.13. The ZBTSRAM can not be accessed by ARM926EJ-S and ACARM9 at the same time, therefore ACARM9 is halted when ARM926EJ-S accesses ZBTSRAM. The memory access can not be overlapped.

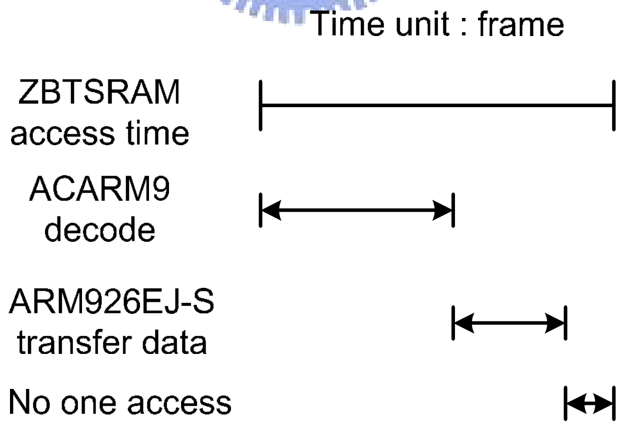


Fig. 4.13 ZBTSRAM utilization

4.4.3 Real-Time Synchronization

This section talks about the real-time synchronization issue. In order to decode MP3 bitstream and play the decoded audio data simultaneously, the control flow should be synchronized. The synchronization scheme is shown in Fig. 4.14. Because of the access behavior of ZBTSRAM, the access time can not be overlapped. The interrupt is designed between the two access requests. After DMAC completes transferring all data, the DMA ISR is asserted. The DMA ISR is to enable ACARM9 to decode the next frame. After each frame is decoded, the logic tile interrupt is asserted. The logic tile ISR is to perform data transfer. The two data transfers are to move bitstream from nor-flash to ZBTSRAM and to move decoded audio data from ZBTSRAM to SSRAM. The audio data in SSRAM will be moved by DMAC to the audio CODEC.

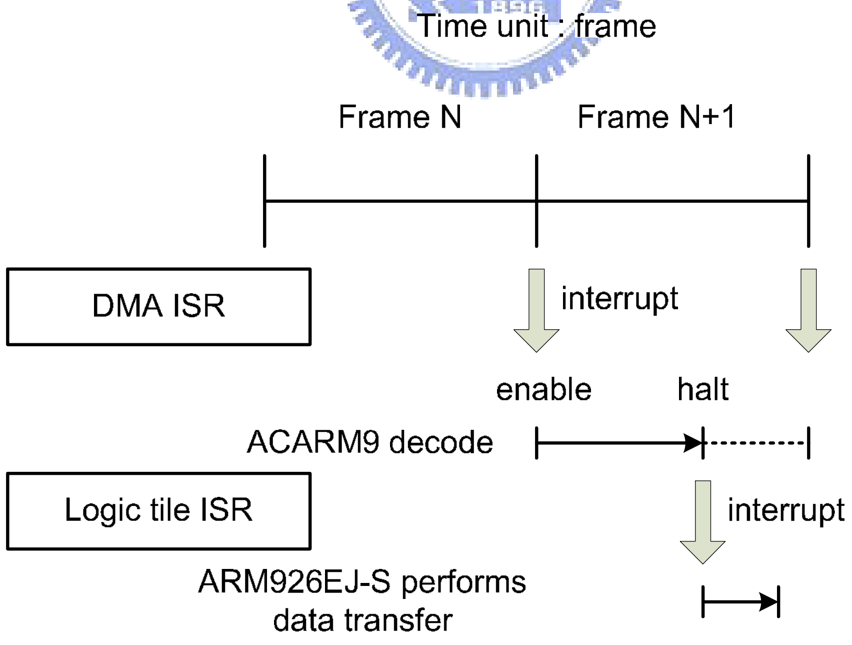


Fig. 4.14 Synchronization

The mailbox mechanism of data transfer from ZBTSRAM to SSRAM is shown in Fig. 4.15. The information tells ARM926EJ-S the address of data, size of data, frame number and decoding error message.

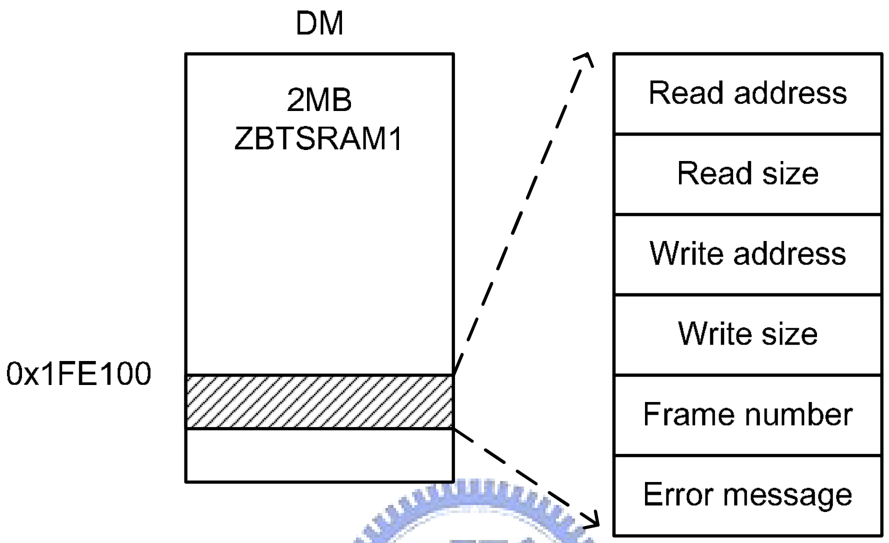


Fig. 4.15 Data transfer mailbox

The frame number indicates the access address of SSRAM. The addresses of even and add frame number are different which is shown in Fig. 4.16. SSRAM can be accessed by multiple AHB masters at the same time. To ensure the correctness of audio play, the double-buffer mechanism is designed. There is no overlapped read/write between ARM926EJ-S and DMAC. By all the mechanism above, the audio play can be played smoothly.

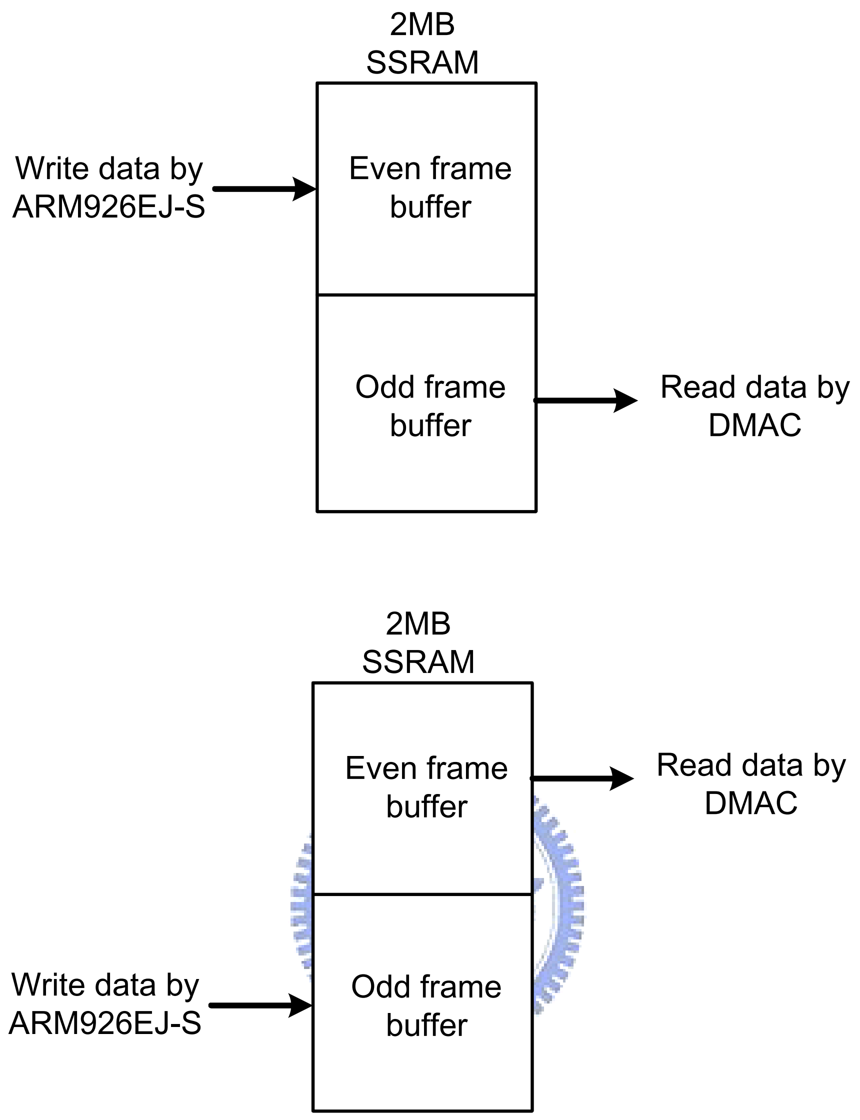


Fig. 4.16 SSRAM frame buffer

Chapter 5
MP3 Decoder Software Design

This chapter describes the MP3 decoder software design, the MP3 decoder software is compiled by ARM Developer Suite 1.2 to generate an executable binary file with ARM V5E instruction set architecture. The binary file is run by ACARM9 processor core, the final result is in PCM for audio CODEC. Section 5.1 describes the decoder software analysis. Section 5.2 describes the method of software porting. Section 5.3 describes the performance improvement.

5.1 Decoder Software Analysis

The decoder software profiling is shown in Fig. 5.1. The performance of each function is listed in Table 5.1. The sample bitstream format is stereo, 44.1 kHz sample rate, 128kbps bit rate and 60 seconds. The Subband includes PolyphaseStereo and FDCT32. The clock rate means the speed to decode 60-second bitstream in one minute (i.e., real time). The major execution time of decoder is PolyphaseStereo, which possesses 60% of the runtime. The improvement method is discussed in Section 5.3.

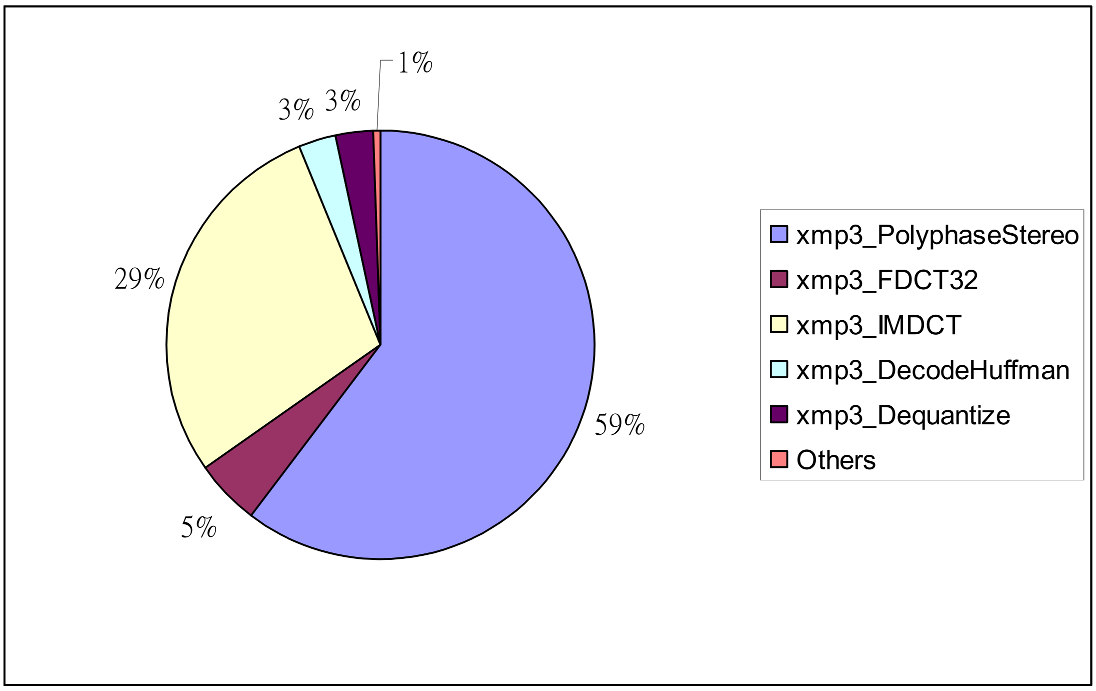


Fig. 5.1 MP3 decoder profiling before improvement

Table 5.1 Profiling details before improvement

MP3Decode	
Clock Rate	54.3 MHz
function:	
xmp3_Subband	64.98
--xmp3_PolyphaseStereo	60.01
--xmp3_FDCT32	4.97
xmp3_IMDCT	28.37
xmp3_DecodeHuffman	2.85
xmp3_Dequantize	2.6
Others	0.64

5.2 Software Porting

The MP3 decoder software is compiled as binary executable file and written into the NOR-flash. After booting, the binary file is loaded into the ZBTSRAM and ready for ACARM9. The file I/Os that read bitstream or write PCM file are modified from the standard library fread and fwrite to memory relocation. The way to relocation is done by ARM926EJ-S. The input bitstream is moved from NOR-flash to ZBTSRAM, The output PCM after decoding is waiting for the memory movement by ARM926EJ-S. The detail memory movement operation is shown as Fig. 5.2. The memory relocation is frame by frame based. Input and output data are moved by the interrupt service routine. The left side of Fig. 5.2 is the input bitstream buffer of decoder and the bytes consumed by one frame. After memory alignment by ACARM9, ARM926EJ-S replenishes input bitstream buffer. The output PCM data is moved from ZBTSRAM to SSRAM on platform baseboard after decoding one frame.

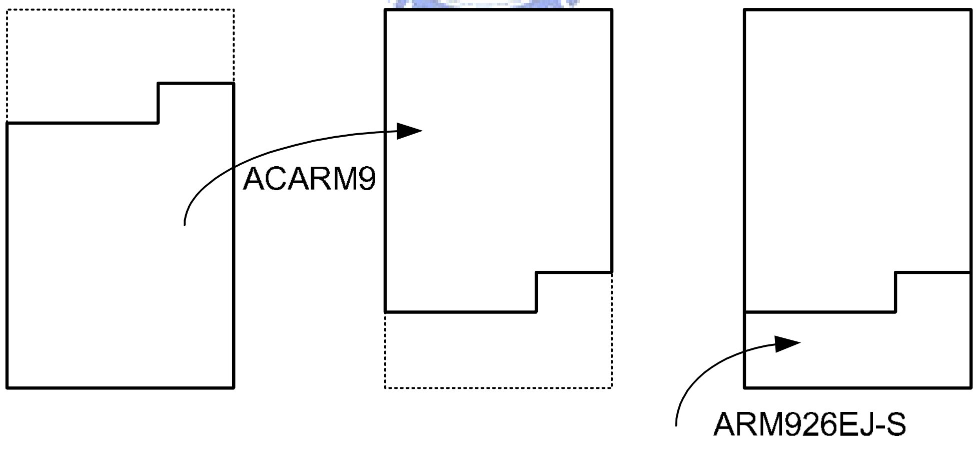


Fig. 5.2 Input bitstream buffer

5.3 Performance Improvement

The decoder can execute at 54.3 MHz for a real-time purpose; in Section 5.1, we know that PolyphaseStereo is the most time-consuming part. In this section, we replace this function with ARM assembly code and inline ARM specific instruction. By these two improvements, the final result is shown in Table 5.2.

The analysis of PolyphaseStereo function gives two results. A large amount of 64-bit calculation is adopted in the decoder software. For the general C program, the 64-bit multiplication can not be well handled; ARM has 64-bit long multiplication instruction to deal with this calculation. The standard library of 64-bit multiplication takes ten times execution time more than ARM specific instruction. The other improvement is to adopt the ARM Version 5 instructions. The version 5 instruction Count Leading Zero (CLZ) is frequently adopted in modern audio or video decode procedure. The CLZ function is shown in Fig. 5.3.

```
count leading zero in c
      if (!x)
            return (sizeof(int) * 8);

      numZeros = 0;
      while (!(x & 0x80000000)) {
            numZeros++;
            x <<= 1;
      }
arm inline assembly code
      __asm {
            clz numZeros, x
      }
```

Fig. 5.3 Count leading zero

After software improvement, this profiling result is shown in Fig. 5.4 and Table 5.2. The sample bitstream format is stereo, 44.1 kHz sample rate, 128kbps bit rate and 60-second. The Subband includes PolyphaseStereo and FDCT32. The overall speed up is three times, the clock rate for real-time decoding is reduced to 17.46 MHz. More detailed discussion is presented in Chapter 6.

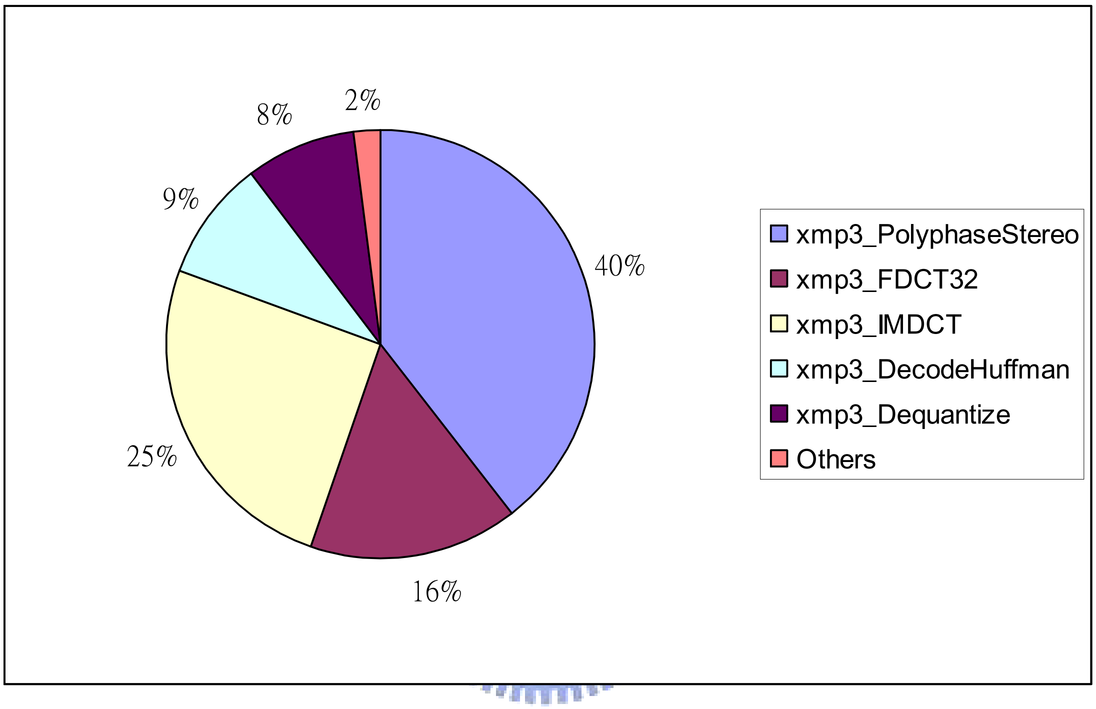


Fig. 5.4 MP3 decoder profiling after improvement

Table 5.2 Profiling details after improvement

MP3Decode	
Clock Rate	17.46 MHz
function:	
xmp3_Subband	54.44
--xmp3_PolyphaseStereo	38.76
--xmp3_FDCT32	15.68
xmp3_IMDCT	24.62
xmp3_DecodeHuffman	8.98
xmp3_Dequantize	8.21
Others	2.01

Chapter 6
Experimental Results

This chapter provides the experimental results of the proposed MP3 decoder system. Section 6.1 presents the experimental result. Section 6.2 discusses about the experimental results.

6.1 Experimental Results

The experiment runs on the ARM Developer Suite simulator to estimate the real-time clock rate; the MP3 decoder software runs on our proposed design ACARM9. ACARM9 is configured in an FPGA of Versatile LT-XC2V6000. Table 6.1 shows that the minimum clock rate to real-time decode the bitstream. The sample bitstream format is stereo, 44.1 kHz sample rate, 60 seconds and 128-320 kbps bit rate.

Table 6.1 Experimental results on ADS

bit rate	cycle count	clock rate (MHz)
128kbps	1,047,402,698	17.456
160kbps	1,066,374,067	17.772
192kbps	1,091,560,738	18.192
224kbps	1,107,073,611	18.451
256kbps	1,122,970,919	18.716
320kbps	1,143,467,949	19.057

The experimental result shows that a general MP3 bitstream with stereo, 44.1 kHz can be run at low clock rate. ACARM9 in FPGA can run all the bitstream and real-time decode under 20 MHz.

6.2 Discussions

The proposed MP3 decoder system runs under 20 MHz on ADS simulation. The assembly code function improves performance by three times; the reasons are lack of registers and multiplication overhead. The polyphase function has too many variables and by calling convention the compiler can not optimize this function; the bottleneck causes the program to generate a great number of register pushes and pops in order to provide enough registers, the pushes and pops are load and store instructions, the load and store instructions degrade the performance. The ARM instruction specific assembly function is to fully utilize all registers, there is no more excessive load and store instructions. The general multiplication libraries for ARM are designed by using 32-bit multiply instructions; in this decoder, 64-bit multiply instructions are adopted for multimedia calculation. The direct inline assembly code is more efficient than calling the one in libraries. Then the software is compiled and run by ACARM9. The software porting of memory relocation and interrupt service routine do not take too many cycles overhead, the decoder system on platform baseboard can be run without any error at 24MHz.

Chapter 7
Conclusions and Future Works

Chapter 2 describes preliminaries of this thesis. We give the overview of ARM9E-S and the MPEG 1 Layer-3 audio decoder. Also, we introduce the instruction set architecture (ISA) that we adopt for ACARM9 and overview of the audio decoding procedure. In the end, the overview of platform baseboard is presented. Chapter 3 describes the proposed processor core. Chapter 4 describes the proposed platform-based system. Chapter 5 describes the proposed MP3 decoder software design and Chapter 6 discusses the experimental results.

In the future work, a more efficient embedded system with a variety of functionalities will be developed. More user interface control and interrupt will be concerned; the hardware and software partition for various reasons are under research. On the other hand, more additional multimedia application like MPEG video decoding will be studied in the future.

References

[1] http://www.arm.com/

[2] The Helix MP3 Decoder, https://datatype.helixcommunity.org/Mp3dec

[3] Steve Furber, ARM System-on-Chip Architecture, Second Edition, Addison Wesley, 2000.

[4] ARM, RealView Platform Baseboard for ARM926EJ-S HBI-0117 User Guide.

[5] ARM, ARM926EJ-S Development Chip Reference Manual.

[6] ARM, ARM Architecture Reference Manual.

[7] ARM, Versatile/LT-XC2V4000+ Logic Tile User Guide.

[8] ARM, Versatile/PB926EJ-S FAQ v1.0 - Last updated September 2004.

[9] ARM, ARM Developer Suite Version 1.2, Getting Started.

[10] ARM, ARM Developer Suite Version 1.2, Developer Guide.

[11] ARM, ARM Developer Suite Version 1.2, AXD and armsd Debuggers Guide.

[12] ARM, ARM Developer Suite Version 1.2, Compilers and Libraries Guide.

[13] ARM, ARM Developer Suite Version 1.2, Linker and Utilities Guide.

[14] ARM, Application Note 115, Using the Audio Codec on ARM Development Boards.

[15] ARM, Application Note 119, Implementing AHB Peripherals in Virtex 2 Logic Tiles.

[16] ARM, Application Note 128, Logic Tile Flashing LED Example.

[17] ARM, AMBA Interrupt Controller Data Sheet.

[18] ARM, ARM9E-S Core Revision: r2p1 Technical Reference Manual.

[19] ARM, GX 175 Multiport Memory Controller Revision: r0p0 Technical Reference Manual.

[20] ARM, PrimeCell Advanced Audio CODEC Interface (PL041) Technical Reference Manual.

[21] ARM, PrimeCell PS2 Keyboard/Mouse Interface (PL050) Technical Reference Manual.

[22] ARM, PrimeCell DMA Controller (PL080) Revision: r1p1 Technical Reference Manual.

[23] ARM, PrimeCell Vectored Interrupt Controller (PL190)Revision: r1p2 Technical Reference Manual.

[24] Xilinx, Virtex-II Platform FPGAs: Complete Data Sheet.

[25] Xilinx, Virtex-II Platform FPGA User Guide.

[26] Xilinx, Libraries Guide ISE 6.li.

[27] Vijaykuinar Gurkhe, "OPTIMIZATION OF AN MP3 DECODER ON THE ARM PROCESSOR," TENCON 2003.

[28] K. Ramkishor, V. Gunashree, "Real Time Implementation of MPEG-4 Video Decoder on ARM7TDMI," Digital Multimedia Group eMuzed Bangalore India, Mobile Multimedia Solutions Sasken Communication Technologies Ltd Bangalore India, May 24 2001 Hang Kong.

[29] Wen-Kai Huang, I-Ting Lin, Shi-Wei Chen and Ing-Jer Huang, "A Cost-Effective Media Processor for Embedded Applications," Department of Computer Science and Engineering National Sun Yat-sen University Kaohsiung, Taiwan, 0-7803-8834-8/05/$20.00, 2005 IEEE.

[30] Chang-Hung Yang, Chin-Yu Huang, Tsui-Ying Hung, Tung-Ju Chiang, and Yung-Ruei Chang, "Software and Hardware co-design for MP3 Decoder," Department of Computer Science National Tsing Hua University Hsinchu, Taiwan, Institute of Nuclear Energy Research Atomic Energy Council Taoyuan, Taiwan, 1-4244-0549-1/06/$20.00, 2006 IEEE.

[31] ISO/IEC 11172-3:1993, "Information technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s -- Part 3: Audio".

[32] Yang-Fan Mu, "Study and Implementation of MPEG1 Layer3 Audio Decoding," Master’s thesis, Control Engineering Group Department of Engineering Science National Cheng Kung University, Tainan, Taiwan 701, R.O.C., Jun. 2004.

[33] Zih-Cheng Huang, "Implementation & Performance Analysis of MP3 Player over ARM9 Platform," Master’s thesis, Master of Science in Department of Computer Science and Engineering National Sun Yat-sen University Kaohsiung, Taiwan, Jul. 2005.

[34] Shao-Hean Hsu, "Implementation of MP3 Playout System on ARM-based SoC Development Platform," Master’s thesis, Master of Science in Department of Computer Science and Engineering National Sun Yat-sen University Kaohsiung, Taiwan, Jul. 2004.

[35] Jun-Sheng Zheng, "Hierarchical Interface Design Methodology: Using Real -Time MPEG1 Audio layer3 codec as a case," Master’s thesis, Department of Electrical Engineering National Cheng Kung University Tainan, Taiwan, R.O.C, Jul. 2002.

[36] Hsiao- Lun Liao, "Platform-based Hardware/Software CoDesign for MP3 Decoder," Master’s thesis, Department of Electrical Engineering National Cheng Kung University Tainan, Taiwan, R.O.C., Jul. 2004.

[37] Jianwei Wang, "Hardware/Software Codesign of MP3 Decoder with 36/32-point (I)DCT Accelerators," Master’s thesis, Microelectronics Department of Electrical Engineering Faculty of Electrical Engineering, Mathematics and Computer Science, Jul. 2005.

[38] KRISTER LAGERSTRÖM, "Design and Implementation of an MPEG-1 Layer

III Audio Decoder," Master's Thesis Computer Science and Engineering Program CHALMERS UNIVERSITY OF TECHNOLOGY Department of Computer Engineering, May 2001.

[39] Hung-Chih Lai, "REAL-TIME IMPLEMENTATION OF MPEG-1 LAYER 3 AUDIO DECODER ON A DSP CHIP," Department of Electrical and Control Engineering College of Electrical Engineering and Computer Science National Chiao-Tung University, Jun. 2001.

[40] Y. -C. Fong, "A High-Speed Area-Minimized Reconfigurable Adder Design," Master's thesis, National Chiao Tung University, Department of Electronics Engineering, Jul. 2006

[41] H. -K. Ling, "A High-Performance Reconfigurable Sub-Word Parallel Multiplier-Accumulator Design," Master's thesis, National Chiao Tung University, Department of Electronics Engineering, Jul. 2006.

[42] http://www.mp3-tech.org/

[43] Dimitrakopoulos, G.; Nikolos, D., "High-speed parallel-prefix VLSI Ling adders" , IEEE Trans. Computers, vol. 54, Issue 2, pp. 225-231, Feb. 2005.