具有 AHB 介面之 JPEG2000 編碼器系統設計

學生：黃琪文　　　　　　　　　　　　指導教授：吳炳飛 教授

國立交通大學電機與控制工程學系 (研究所) 碩士班

摘要

由於 JPEG2000 是最先進的影像壓縮格式，我們實驗室也致力於開發高效能 JPEG2000 晶片，並提出比傳統小波離散轉換 (Discrete Wavelet Transform) 更有效率的 QDWT (Quad Discrete Wavelet Transform)。 QDWT 的優勢在於可以比傳統 DWT 提早四分之三的時間將編碼資料送出至下級 EBCOT (Embedded Block Coding with Optimized Truncation) 。我們也開發高效能的算數編碼器，採用三級管線的平行化架構達到 1 CX-D pair/clock cycle 的輸入率。在本論文中會說明如何透過系統工作流程安排，分析系統內部每塊模組的工作時間，決定出效能最好的系統架構。

為了使我們開發的 JPEG2000 編碼器更具 IP 化，我們將其外掛一層 AHB (Advanced High-performance Bus) Slave 介面。AMBA (Advanced Microcontroller Bus Architecture)為 ARM 所制定的系統內部匯流排的溝通介面，是目前市面上最常被拿來使用的介面，因此，我們所設計的具有 AHB 介面的 JPEG2000 編碼器可應用於任何 ARM-based 的嵌入式系統。本論文的貢獻在於成功整合一顆具有平行化架構的 JPEG2000 Coprocessor，並呈現此架構確實可以大幅提升 JPEG2000 的效能。此外，也成功的為 JPEG2000 Coprocessor 掛上 AHB 介面，並使之與 ARM CPU 一起工作，完成整個 JPEG2000 的編碼流程。

AHB-based JPEG2000 Coprocessor System Design

Student：Chi-Wen Huang Advisor：Prof. Bing-Fei Wu

Department of Electrical and Control Engineering

National Chiao Tung University

ABSTRACT

Because JPEG2000 is the state-of-the-art image compression technology, our lab has made efforts in developing a high-performance JPEG2000 chip and developed QDWT (Quad Discrete Wavelet Transform) which is more efficient than the traditional DWT (Discrete Wavelet Transform) . QDWT only needs the quarter of compute time than the traditional DWT does to generate the coefficients to EBCOT (Embedded Block Coding with Optimized Truncation). We also develop a high-performance AC (Arithmetic Entropy Coder). The pipeline architecture is used in the AC and we only use three pipes to reach the input rate, 1 CX-D pair/clock cycle. We will explain that how to organize the best system architecture to achieve small area and high throughputs by arranging the system work flow properly and analyzing the timing of the individual modules.

If the ASIC developed can be popular to be integrated into different systems, the IP issue should be addressed. We wrapped the JPEG2000 Encoder developed by our team in AHB (Advanced High-performance Bus) Slave interface. AMBA, which is drawn up by ARM, is an on-chip communication standard for designing high-performance embedded microcontrollers and is wildly used in the consumer electronic market now. So, the AHB-based JPEG2000

Encoder we developed could be applied in an ARM-based embedded system.

The Contribution of this thesis is to integrate the QDWT, Pass Parallel EBCOT Tier1 and Pipeline AC as a JPEG2000 coprocessor and show this architecture really could improve the performance. Besides, wrap the JPEG2000 coprocessor in AHB slave interface and make it cooperate with ARM CPU to finish the coding procedures of JPEG2000.

致謝

首先，最應該感謝的人，當然就是指導我四年的吳炳飛教授了。感謝老師在我大學時就收我當專題生，猶記當時我根本不懂什麼是硬體設計，什麼叫系統整合，到現在有能力去 handle 一個系統，這都是老師一點一滴的栽培訓練，把作為一個硬體設計研發人員該有的 sense 教給我，讓我懂得如何設計開發，以符合市場和業界的需求。

另外，也要感謝老師提供我良好的學習環境和學習資源。在設備和開發工具上的資源可以說是應有盡有，讓我在這幾年中，可以學習和接觸到不同的開發環境，對我未來的發展奠定了良好的基礎。

再來，要感謝的人，就是曾經帶領過我的 Money(錢昱瑋學長)，旭哥(顏志旭學長)，強哥(胡益強學長)，重甫(林重甫學長)，在你們帶領我的期間，都讓我學到很多不同領域的東西，使我在各方面，都有顯著的成長。

當然，也要感謝其他實驗室的夥伴們，和大家一起合作，一起討論，讓我體會到什麼叫團隊，大家一起朝著共同的目標努力的那種感覺，有時讓我覺得做研究也是一件很快樂的事。在這裡要特別謝謝曾經和我一起開發 JPEG2000 系統的 VK(楊明達學長)，紹麒(呂紹麒學長)，沛君，晏阡，培恭，和你們一起合作，一起參加比賽，一起努力，這當中的點點滴滴都是很美好的回憶。

最後，要感謝一直在背後默默支持我的家人和摯友，謝謝你們辛苦的付出，讓我可以無後顧之憂，並在我低潮時，陪伴我，鼓勵我，今天可以順利完成研究所的學業，絕不是光靠我一個人就可以的，謝謝你們！

Table of Contents

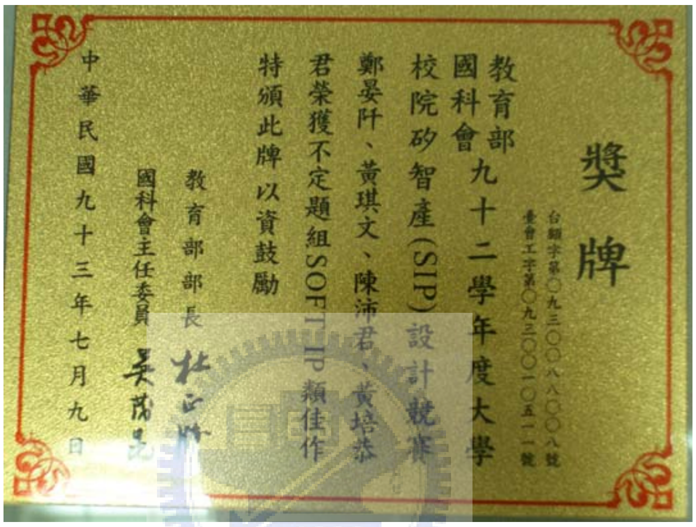
Lists of Figures

Lists of Table

Awards

＊　本論文曾經參與 **國科會教育部九十二學年度大學校院矽智產(SIP)設計競賽**

榮獲 不定題組 SOFT IP 類佳作



獎牌

台顧字第○九三○○八八○○八號
臺會工字第○九三○○一○五一一號

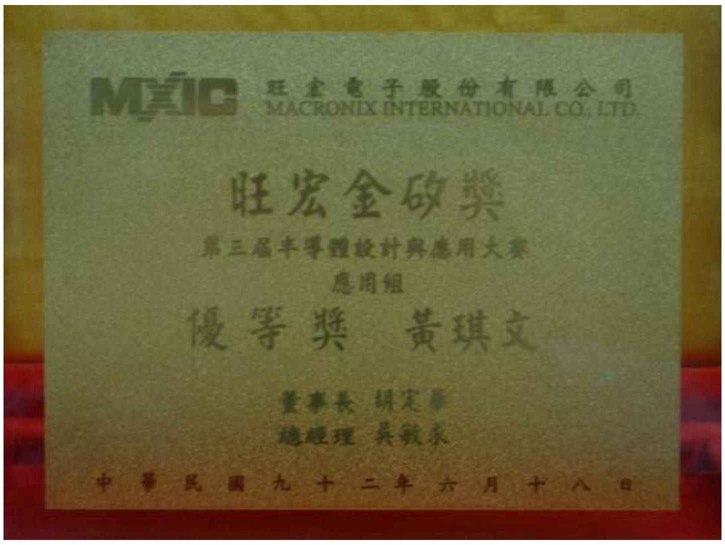
教育部
國科會 九十二學年度大學
校院矽智產(SIP)設計競賽
鄭晏阡、黃琪文、陳沛君、黃培恭
君榮獲不定題組SOFT IP類佳作
特頒此牌 以資鼓勵

教育部部長 [illegible]
國科會主任委員 [illegible]

中華民國九十三年七月九日

＊　本論文曾經參與 **旺宏金矽獎 第三屆半導體設計與應用大賽**

榮獲 應用組 優等獎



MXIC 旺宏電子股份有限公司
MACRONIX INTERNATIONAL CO., LTD.

旺宏金矽獎
第三屆半導體設計與應用大賽
應用組
優等獎　黃琪文

董事長 胡定華
總經理 吳敏求

中華民國九十二年六月十八日

Preface

Although the JPEG image compression is widely used in the multimedia products, ISO/IEC draws up a new image compression standard, JPEG2000 image coding system, which has higher quality, higher resolution and higher compression ratio than JPEG. The JPEG2000 can be applied in many regions such as internet, digital photography, medical imaging, wireless imaging, surveillance, printing and scanning … etc. In these applications, the high quality, high resolution and high compression ratio of JPEG2000 can make the performance better than JPEG. But the only weakness of JPEG2000 is its complicated algorithm. If only use software to implement the JPEG2000, except we have the fastest CPU and enough memory, the frame rate will not be accepted.

Our lab has made efforts in developing the JPEG2000 encoder hardware to increase the performance of JPEG2000. To make it more applicable or popular to the market, we have to lower the chip cost and make the chip area as small as possible. How we develop the high performance JPEG2000 ASIC will be shown in this thesis.

In the JPEG2000 hardware design, QDWT (Quad Discrete Wavelet Transform) [14], EBCOT (Embedded Block Coding with Optimized Truncation) [2] [13] and AC (Arithmetic Entropy Coder) [2] are enhanced individually first. How to integrate them properly to achieve the high performance will be explained in the following chapters. Besides, we wrap the JPEG2000 encoder in AHB wrapper to apply it in the ARM embedded system.

The outline of this thesis is the system overview in Chapter 1, JPEG2000 coprocessor hardware design and integration in Chapter 2, Arithmetic Entropy Coding hardware design in Chapter 3, AMBA, AHB-based JPEG2000 coprocessor hardware design and integration in Chapter 4, achievements and perspectives in Chapter 5.

Chapter 1 System Overview

1-1 Introduction

Before we tape out the JPEG2000 encoder chip, it’s better to verify the chip function on the FPGA first. The JPEG2000 encoder hardware is defined as a coprocessor in this thesis because we need a microcontroller to do some complicated calculation such as EBCOT (Embedded Block Coding with Optimized Truncation) Tier2 and packet header information.

Hence, it may be a good choice to use ARM/Integrator as the development platform for ARM CPU is the most popular CPU in the market now and other considerations are the image source and demonstration. We plan to apply the JPEG2000 encoder in the surveillance system, so a camera and internet would be included to make the system more powerful. The ARM/Integrator contains all we need, so it is the good platform for us.

1-2 ARM Integrator Platform

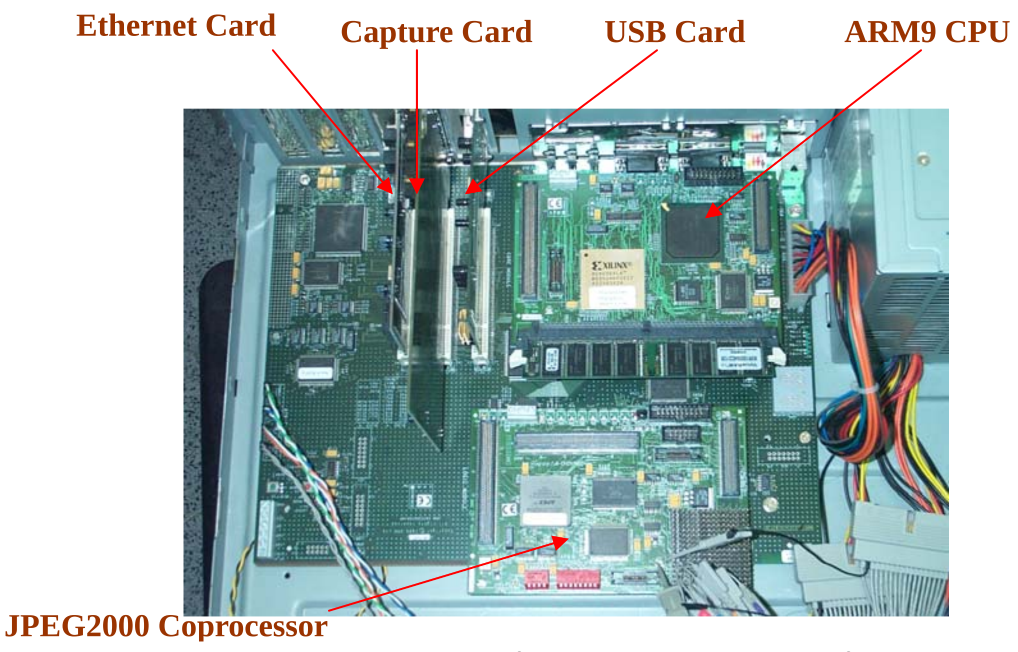


Figure 1-1 Picture of the ARM Integrator Platform

1-3 System Block Diagram

We draw the block diagram of the integrator as Figure 1-2:

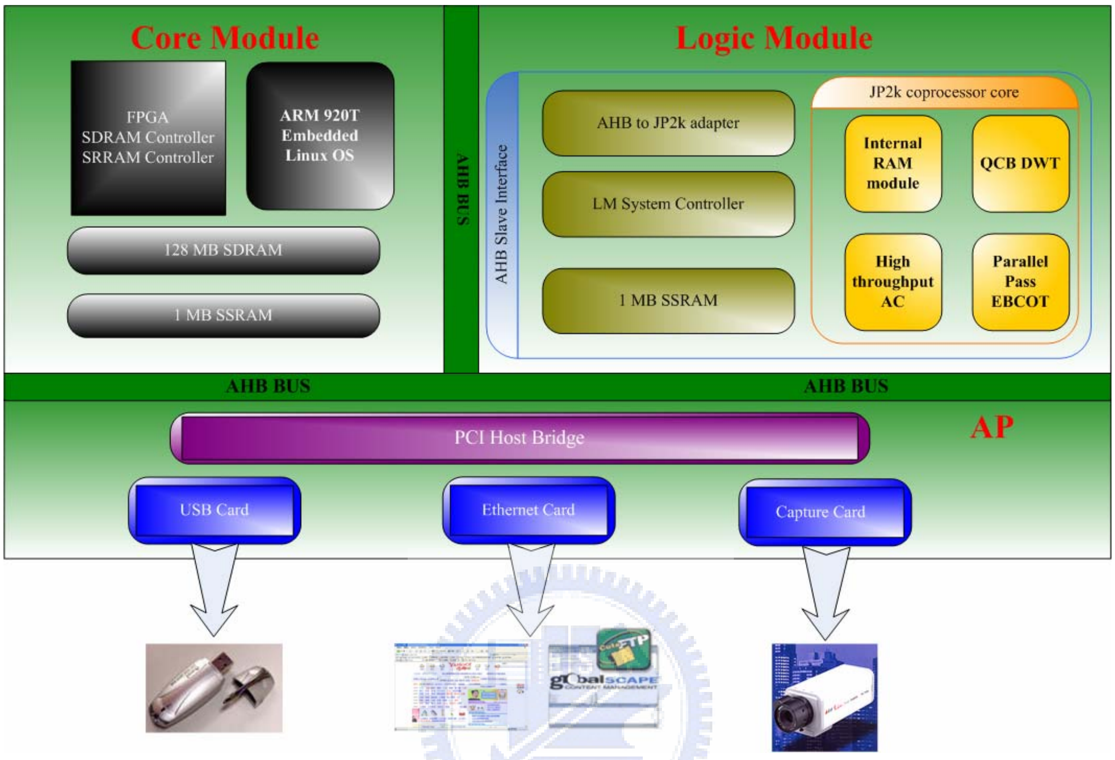


Figure 1-2 Overall System Block Diagram

1-3.1 Motherboard (Integrator/AP)

Integrator/AP is very similar to general PC (Personal Computer) motherboard; it has three PCI slots, two COM Ports, PS/2 port. Besides, AP has two sockets, one is for Core Modules, and the other is for Logic Modules. We can stack up four Core/Logic Modules on the individual socket, that is, we can use more than two Core/Logic modules at the same time, and this increases the applicability of the Integrator.

Three PCI slots are inserted three cards to develop a surveillance system.

＊ **Capture Card**: Use CCD camera to capture the real time image to be the source images of JPEG2000 Encoder. The JPEG2000 Encoder will do the real time encoding。

＊ **Ethernet Card:** Set up a FTP Server in the Integrator. Users can remote access the files stored in the memory or storage through FTP.

＊ **USB Card**　：There is no USB Port on the Integrator/AP. If USB devices were needed, we will need a USB card. For the surveillance system, we can save the images in the USB storages.

1-3.2 Core Module (Integrator/CM920T)

ARM Core is put on the core module; we can change different core modules according to different ARM cores. There are 32MB Flash, 1MB SRAM, 128MB SDRAM on the core module, Linux OS can be ported to develop the API (Application Program Interface) and the drivers for the peripherals.

Besides, we need ARM CPU to help doing the operation of Tier2 and packet header of the JPEG2000 Coprocessor.

1-3.3 Logic Module (Integator/LM-EP20K600E+)

There is an ALTERA FPGA, the content in which is about 1,000,000, 1 MB SSRAM, 32MB Flash on the Logic Module. The hardware of the JPEG2000 coprocessor is put here.

Chapter 2 JPEG2000 Coprocessor Hardware Design

2-1 Introduction

Because the QDWT (Quad Discrete Wavelet Transform), EBCOT (Embedded Block Coding with Optimized Truncation) Tier1 and AC (Arithmetic Entropy Coder) has been enhanced individually, the next is how to integrate them properly to achieve the high performance. It will be explained that how we organize the system architecture of the JPEG2000 coprocessor in this chapter.

Besides, the test circuit is added to make sure the individual module is workable. Hence, the JPEG2000 coprocessor has two operation modes, Test mode and Normal mode. When the JPEG2000 coprocessor is in the test mode, we can select the module we would like to test by its test id, then feed the test patterns to the input of the module and get the results from the respected output pads.

2-2 Main Module Introduction

Before starting to integrate the QDWT, EBCOT and AC, let us introduce the features of these modules first.

2-2.1 QDWT (Quad Discrete Wavelet Transform)

QDWT [14] cuts the input tile image into four parts as in Figure 2-1 left side. After it encoded part 1, it generates three code blocks to EBCOT as in Figure2-1 right side. When the part 2 is coded, another three code blocks to EBCOT are generated again. After the part 2 is coded, the next is part 3 and part 4; the encoding flow is as follows

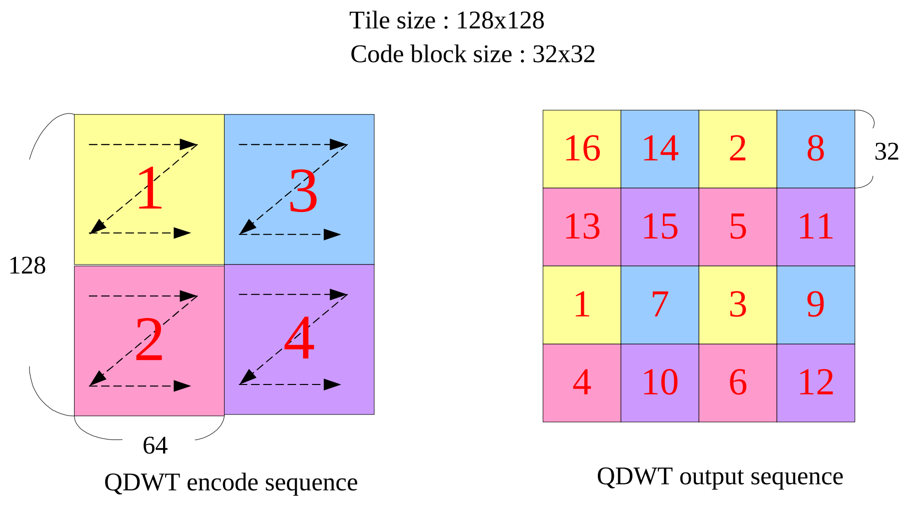


Figure 2-1 QDWT encode and output sequence

Base on this algorithm, when QDWT finishes encoding the quarter of the tile image, EBCOT could start to encode the code block data (the code block size is 32x32 bytes). That's why QDWT just needs the quarter of compute time than the traditional DWT to output the coefficients to EBCOT and this feature does increase the overall system performance.

2-2.2 Pass Parallel EBCOT Tier-1 and Arithmetic entropy Coding

EBCOT (Embedded Block Coding with Optimized Truncation) Tier1 is the entropy coder in the JPEG2000; it transforms the output coefficients of DWT to the optimized single bit-stream.

The pass-parallel architecture is used in this design[13], that is, the three coding passes are supposed to be coded in order originally, but it is not needed to code in this way right now. In the pass-parallel architecture, we can encode the three passes in every bit plane at the same time and can save about 25% processing time and reduce 4K bits of internal memories when code-block size is 32 x 32.

AC (Arithmetic Entropy Coding) is also the entropy coder in the JPEG2000, it cooperates with EBCOT.

AC receives the context label and symbol from EBCOT, then does the encoding operation and output the compressed image data. The pipeline architecture is used in AC hardware implementation, the throughput can increase at least 10 times than the standard and the AC can receive one CX-D pair per clock cycle. We will clearly describe how to design the high performance AC in Chapter 3.

2-3 JPEG2000 Coprocessor Architecture

To integrate all modules fast and to achieve the high performance, we have to do two things first:

- Analysis the overall system timing, make all modules to keep on working as possible as we can
- Define the module interface IO and timing properly

2-3.1 Analysis the overall system timing

After we know the work flow of the individual module, we can start to organize them in an efficient way. The ideal case is that every module is always working; the waiting condition is never happened. We analysis the system timing and organize all modules as in Figure 2-2

- Because the QDWT will generate three code blocks in three bands (LH, HL, HH) to EBCOT at the same time after encoding a quarter of a tile, we also use three EBCOTs to deal with these code blocks simultaneously. (See Figure 2-2)
- The transferred data unit is a code block (32x32=1024 bytes) between DWT and EBCOT, so a buffer is required to store the code block data. In general, ping-pong buffers will be setup to avoid the waiting condition happened. (See Figure 2-2) When the EBCOT is reading one buffer, QDWT can be writing another buffer. If two buffers are full, the waiting condition happens. When the waiting condition happens, it may

not a good idea to add more buffers because of the limitation of the on-chip memory. Furthermore, to add more buffers cannot solve the waiting condition, the efficient way is to find out the bottle neck and its performance to balance the operation time of two modules.

Three pair ping-pong buffers for three EBCOT are used. The sizes of the ping-pong buffer are **6K+768 bytes**.

- Because each EBCOT will generate three passes context labels and symbols at the same time, every pass is connected to an AC to deal with all passes simultaneously. There are 9 ACs in the JPEG2000 coprocessor. (See Figure 2-2)
- There are 9 ACs which are encoding at the same time, so, there may be more than two compressed image bytes generated in the same cycle. So, an FIFO is setup to collect the data first, then rearrange them and write to the SSRAM outside the JPEG2000 coprocessor.

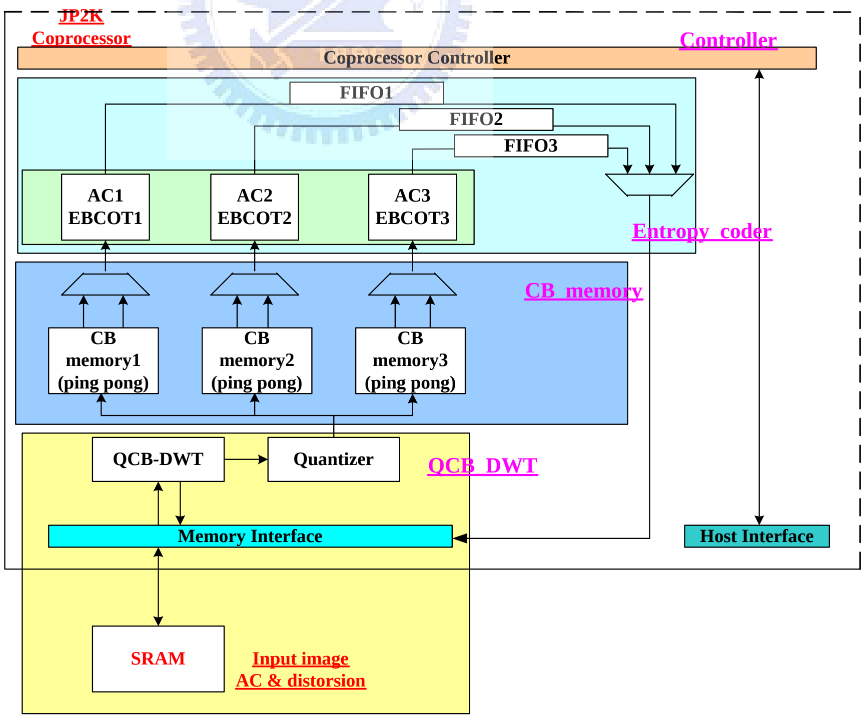


Figure 2-2 the Architecture of JPEG2000 Coprocessor

2-3.2 Define the module interface I/O and timing properly

When we name the interface I/O port, it is better to name it with a meaningful name. In general, we name an I/O port by its function in the design. For example, we often name input clock source CLK. In this way, we can increase the readability of a program. When other programmer reads your code, he can understand your codes in a short time.

In our design, every module has two operation modes: Test mode and Normal mode. So, the additional input ports and output ports are added in some main modules. Only when the test mode is enabled, these test I/O ports work. The test I/O ports have the different meanings in the different module. If we don’t use the general test I/O ports, it will become very complicated to integrate the test circuits because the test input patterns come from the test bench which is outside the chip. So, the test I/O ports in the different module are named in the same name for integrating the circuits easily.

The key point to shorten the time of the system integration is that to define the interface timing properly. Before we start coding a module, we have to know how to communicate with others and consider all the conditions happened in communication.

＊ **I/O of the JPEG2000 Chip**

Table 2-1 is the pin definition of the JPEG2000 coprocessor chip I/O. Test_Mode signal is for mode select. When Test_Mode=1, the coprocessor enters the test mode; when Test_Mode=0, the coprocessor enters the normal mode. Com_Sel is for component select when system is in test mode. CLK is the clock source and nReset is the system reset signal which is active low. Input_port and Output_port are two general I/O ports. FIFO_Empty is the signal to indicate if the FIFO outside the chip is empty or not. The rest pins are the I/O connected to the SSRAM outside the chip.

Table 2-1 JPEG2000 coprocessor pin definition

Name	**Direction**	**Width**	**Description**
Test_Mode	Input	1	Mode select. when HIGH: test mode When LOW: normal mode
Com_Sel	Input	7	Test component select.
Input_Port	Input	63	General input ports
Output_Port	Output	94	General output ports
CLK	Input	1	Clock source
nReset	Input	1	System reset signal. Active LOW
FIFO_Empty	Input	1	Indicate FIFO outside the chip is empty or not
Sn_CE	Output	1	Control signal. SSRAM chip enable
SnWR	Output	1	Control signal. SSRAM write enable
SnOE	Output	1	Control signal. SSRAM output enable
SADDR	Output	19	SSRAM address bus
SRDATA	Input	32	SSRAM read data bus
SnWBYTE	Output	4	Control signals. SSRAM byte select

∗ Pin Map : Normal Mode

In different operation mode, the Input_Port and Output_Port are mapped to the corresponded pin map tables. When the JPEG2000 coprocessor is in normal mode, these I/O ports are mapped like in Table 2-2.

When the coprocessor is in test mode, there are 36 test modules and the individual pin map table is in the Appendix.

Table 2-2 Pin map table in normal mode

Pin	**Signal Map**	**Description**
Input_Port[0]	Image_ini	Indicates to start a image initialization
Input_Port[1]	Tile_EN	Indicate to start a tile encoding
Input_Port[2]	Tile_Done_ack	Acknowledge for Tile_Done signal
Input_Port[11:3]	NumTile	Indicate how many tiles in this image
Input_Port[19:12]	Tile_x	Indicate the length of a tile
Input_Port[27:20]	Tile_y	Indicate the width of a tile
Input_Port[29:28]	Numlayer	Indicate how many layers
Input_Port[30]	SSRAM_Ready	Indicate the SSRAM_Addr data is valid or not
Input_Port[62:31]	SSRAM_Addr	The compressed data will be saved in the SSRAM. SSRAM_Addr indicates the beginning of SSRAM address
Output_Port[0]	LLbandMove_ini	Indicate the LL band is prepared to be coded.
Output_Port[1]	Tile_Done	Indicate a tile is completely compressed.
Output_Port[2]	Image_ini_ack	Acknowledge for Image_ini signal
Output_Port[3]	CB1_Done	Indicates a code block is completely coded.
Output_Port[4]	QDWT_req	The request signal comes from QDWT
Output_Port[36:5]	Address	The compressed data will be saved at this address
Output_Port[68:37]	OutBus	The 32-bit data bus for compressed data
Output_Port[69]	RAM_EN	Data valid signal

2-4 Operation Flow Chart

Figure 2-3 is the operation flow chart of the JPEG2000 coprocessor.

After power on, the system will enter IDLE state until Chip Enable is asserted. After the coprocessor is enabled, it will enter Normal mode or Test mode according to the Test_mode signal. If Test_mode=1, then the system enters Test mode, otherwise, enters Normal mode.

When the system is in the Normal mode, it has to get the information of the image configuration first. Then wait until Tile_EN=1, the coprocessor will start coding a tile. After finishing a tile-coding, it will check if all tiles are coded. If not, it will wait until Tile_EN=1 again, if it does, the coprocessor will return to the IDLE state.

When the system is in the Test mode, the host outside has to select the test module by its ID defined in Table 2-3. Then host feeds the corresponding test input patterns in the test module and get the results from output pads. We can compare the results with the respected sequences and dump the report in the text file.

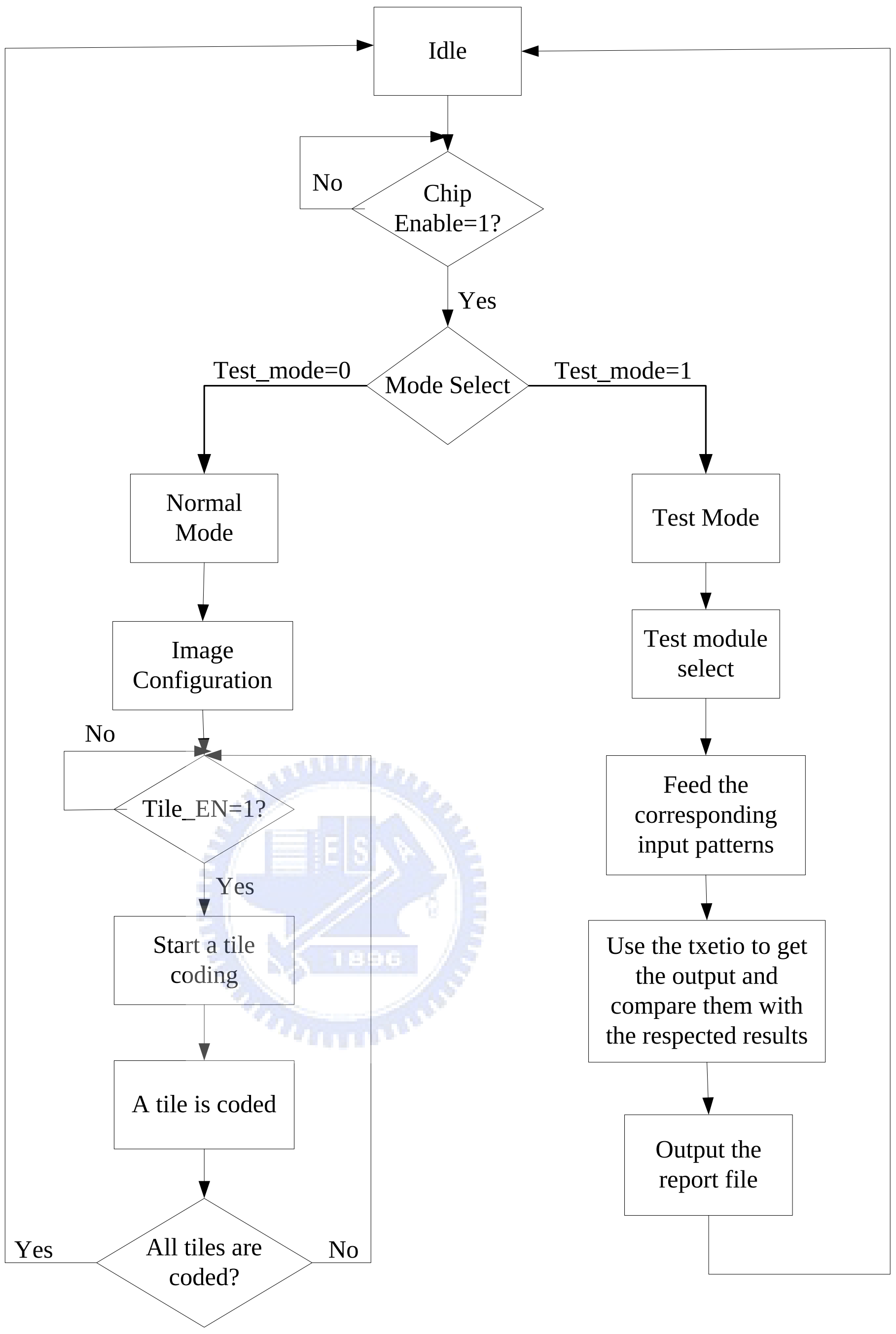


Figure 2-3 JPEG2000 coprocessor operation flowchart

2-5 Coprocessor Controller

The coprocessor controller controls the work flow of the JPEG2000 Encoder. It does three things:

- Control the work flow of QDWT

- Control the work flow of EBCOT
- Generate the SSRAM address, where the compressed image data will be stored, to FIFO Controller.

2-5.1 the Control of QDWT

First of all, the controller has to receive the image configuration data which includes the number of tiles in the current image. Then, wait until a tile data are ready, the QDWT controller will be started. After the QDWT controller finishes its work, the coprocessor controller will check if this is the last tile. If the current tile is not the last tile, the coprocessor controller will return to wait until the next tile is ready. If the current tile is the last tile, the coprocessor controller will check if the EBCOT finishes its work, if not, wait until the EBCOT finishes its work, if it is, an image coding is done. (See Figure 2.4 left side).

For the QDWT controller, after a tile data are ready, it will generate the necessary information which QDWT requires. Then, check if ping-pong buffers are empty. When two buffers are full, the QDWT controller will wait until one of both is empty. If two buffers are not full, check if the QDWT finishes its work, if not, waits until it finishes, if it does, check if the current code block is the last. If the current code block is the last, the QDWT controller finishes its work, if not, the QDWT controller will return to “Prepare QDWT data” state again.

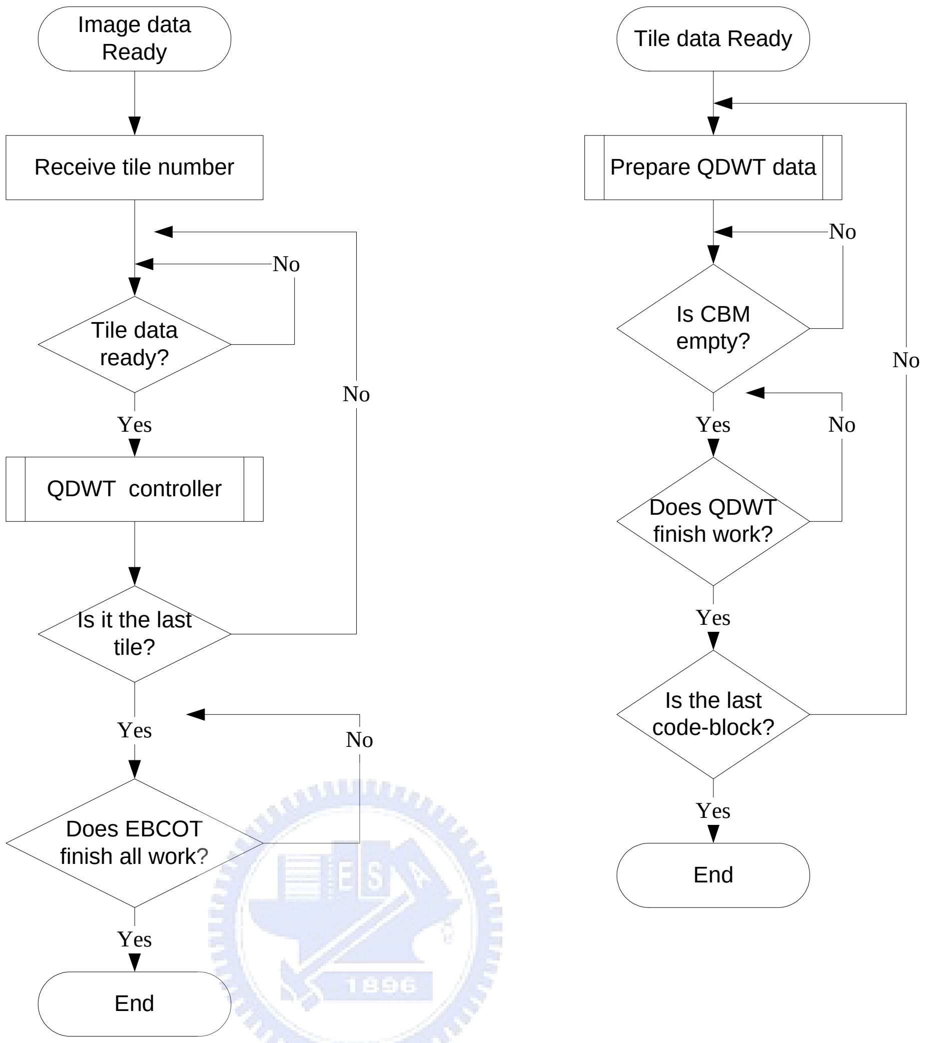


Figure 2-4 the control flow for QDWT

2-5.2 The Control of EBCOT

For the EBCOT controller, it will wait until the code blocks for three bands are ready, then, send the initial information of the current code block to the EBCOT. Then, waits until three EBCOTs finish their work, check if the code blocks of LH, HL and HH bands are coded, if not, return to wait until another three code-block data are ready, if it does, wait until the code-block data of LL band are ready. After sending the information of the LL band code block, EBCOT controller waits until the EBCOT finishes the last code-block coding, then a tile coding is done.

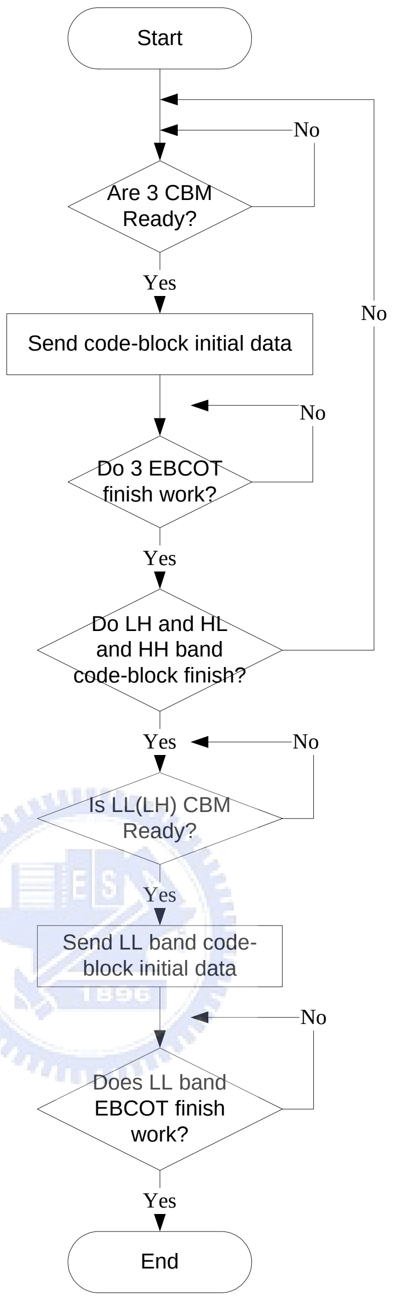

Start
No
Are 3 CBM Ready?
Yes
No
Send code-block initial data
No
Do 3 EBCOT finish work?
Yes
Do LH and HL and HH band code-block finish?
Yes
No
Is LL(LH) CBM Ready?
Yes
Send LL band code-block initial data
No
Does LL band EBCOT finish work?
Yes
End

Figure 2-5 the control flow for EBCOT

2-6 Test Circuit Design

Test circuit is added to make sure all modules are workable and find out where the chip bug is quickly, the. As mentioned above, we add the test input ports and output ports for test circuits. The multiplexers to multiplex these ports for the dedicated module when system entering test mode are applied.

Figure 2-6 shows the block diagram in test mode.

There are four main parts: Code-Block-Memory part, EBCOT-AC part, System-Controller part and DWT part.

Code-Block-Memory (CBM) part contains three CBM components, CBM1, CBM2 and CBM3.

EBCOT-AC part contains three entropy coders and one FIFO controller. Every entropy contains one Tier1 and three ACs.

DWT part contains two single port rams, two dual port rams, line buffer, DWT row processor and DWT column processor.

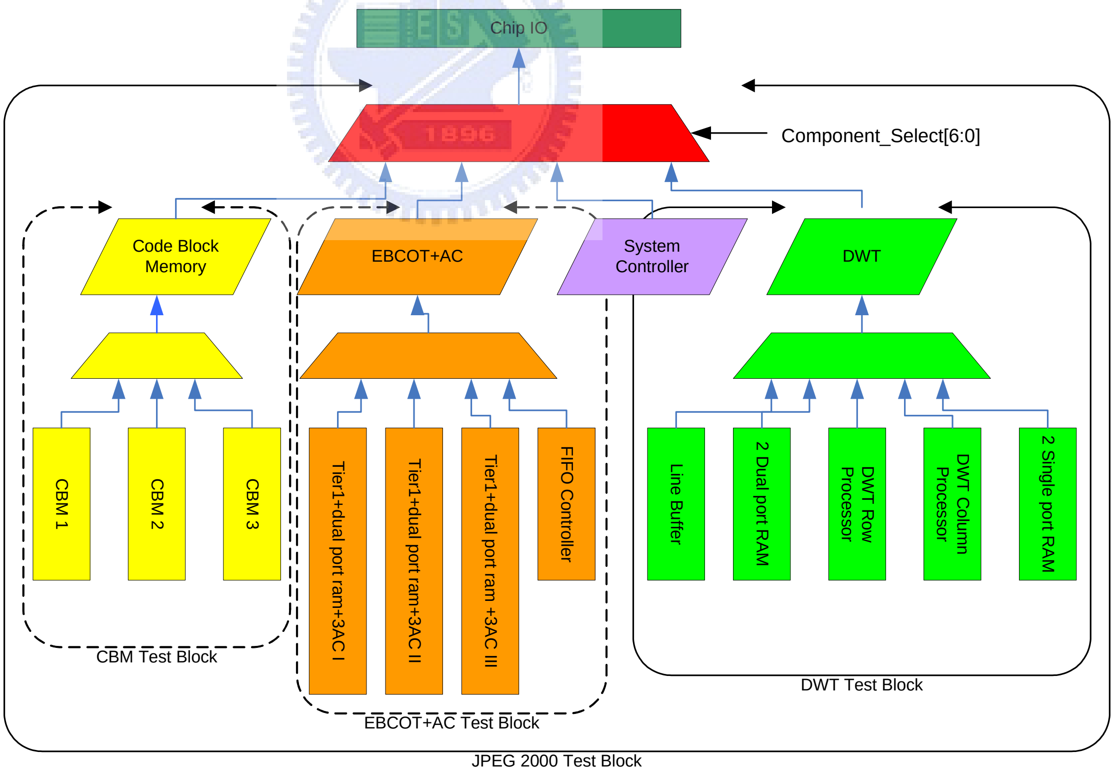


Figure 2-6 Test mode block diagram

When the chip enters test mode, we can select the module we want to test by the module ID. Table 2-3 lists all module IDs.

Table 2-3 Test Module ID

Module Name	**Module ID**	**Module Name**	**Module ID**
DWT_RA1SD1	0000010	HL_AC1	0101001
DWT_RA2SD1	0000000	HL_AC2	0101010
DWT_RA1SD2	0000011	HL_AC3	0101011
DWT_RA2SD2	0000001	HH_Tier1	0110101
DWT_ROW	0000100	HH_Tier1_RAM	0110100
DWT_COL	0000110	HH_SIPO	0110111
Line_buffer	0000101	HH_FIFO	0110110
LH_Tier1	0100101	HH_AC1	0110001
LH_Tier1_RAM	0100100	HH_AC2	0110010
LH_SIPO	0100111	HH_AC3	0110011
LH_FIFO	0100110	FIFO Controller	0111000
LH_AC1	0100001	Coprocessor Controller	1000000
LH_AC2	0100010	CBM1	1100010
LH_AC3	0100011	CBM1_RAM	1100011
HL_Tier1	0101101	CBM2	1100100
HL_Tier1_RAM	0101100	CBM2_RAM	1100101
HL_SIPO	0101111	CBM3	1100110
HL_FIFO	0101110	CBM3_RAM	1100111

2-7 Achievement

We planed to tape out the JPEG2000 coprocessor chip in January 2004 and the chip specification is in Table 2-4. The comparison with others will be put in Chapter 5.

Table 2-4 JPEG2000 coprocessor chip specification

Technology	TSMC 0.25 um
Package	208-pin CQFP
Core size	3.15mm x 3.15mm
Die size	3.95mm x 3.99mm
Operation frequency	41 MHz
Internal rams	8.25 KB
Power consumption	740 mW

Chapter 3 Arithmetic Entropy Coding

3-1 Introduction

In JPEG2000 encoder, AC (Arithmetic entropy Coding) is following EBCOT Tier1, it receives the decision (D) and context (CX) pairs from Tier1 and does more efficient compression.

Because the output rates of the previous stage EBCOT Tier1 increase, we need a corresponding AC with high input rates and can generate the compressed image data as soon as possible. In this chapter, we will explain how to increase the throughputs by using pipeline architecture.

3-2 AC Operations

Let us introduce the AC operations defined in the JPEG2000 standard first [2].

3-2.1 Recursive interval subdivision

The recursive probability interval subdivision of Elias coding [1] is the basis for the binary arithmetic coding process. With each binary decision the current probability interval is subdivided into two sub-intervals, and the code string is modified (if necessary) so that it points to the base (the lower bound) of the probability sub-interval assigned to the symbol which occurred.

In the partitioning of the current interval into two sub-intervals, the sub-interval for the MPS (More Probable Symbol) is ordered above the sub-interval for the LPS (Less Probable Symbol). Therefore, when the MPS is coded, the LPS sub-interval is added to the code string.

This coding convention requires that symbols being recognized as MPS or LPS, rather than 0 or 1. Consequently, the size of the LPS interval and the sense of the MPS for each decision must be known in order to code that decision.

3-2.2 Coding conventions and approximation

The coding operation are done using fixed precision integer arithmetic and using an integer representation of fractional values in which 0x8000 is equivalent to decimal 0.75. The interval A is kept in the range $0.75 \le A < 1.5$ by doubling it whenever the integer value falls below 0x8000.

The code register C is also doubled each time when A is doubled. Periodically – to keep C from overflowing – a byte of compressed image data is removed from the high order bits of the C-register and placed in an external compressed image data buffer. Carry-over into the external buffer is prevented by a bit stuffing procedure.

Keeping A in the range $0.75 \le A < 1.5$ allows a simple arithmetic approximation to be used in the internal subdivision. The interval is A and the current estimate of the LPS probability is Qe, a precise calculation of the sub-intervals would require:

A-(Qe*A)= sub-interval for the MPS

Qe*A= sub-interval for the LPS

Because the value of A is of order unity, these are approximated by

A-Qe= sub-interval for the MPS

Qe= sub-interval for the LPS

Whenever the MPS is coded, the value of Qe is added to the code register and the internal is reduced to A-Qe. Whenever the LPS is coded, the code register is left unchanged and the interval is reduced to Qe. The precision range required for A is then restored, if necessary, by renormalization of both A and C.

With the process illustrated above, the approximations in the interval subdivision process can sometimes make the LPS sub-interval larger than the MPS sun-interval. If, for example, the value of Qe is 0.5 and A is at the minimum allowed value of 0.75, the approximate scaling gives 1/3 of the interval to the MPS and 2/3 to the LPS. To avoid this size inversion, the MPS and LPS intervals are exchanged whenever the LPS interval is larger than then the MPS interval. This MPS/LPS conditional exchange can only occur when normalization is needed.

3-3 Description of the Arithmetic Encoder

The ENCODER (Figure 3-1) initializes the encoder through the INITENC procedure. CX and D pairs are read and passed to ENCODE until all pairs have been read. The probability estimation procedures which provide adaptive estimates of the probability for each context are imbedded in ENCODE. Bytes of compressed image data are output when necessary. When all of the CX and D pairs have been read, FLUSH will output the final bytes, terminate the encoding and generate the required terminating marker.

3-3.1 Encoder code register convention

The flow charts given in this chapter assume that the register structures for the encoder are shown in Table 3-1

Table 3-1 Encoder register structures

	MSB			LSB
C-register	0000 cbbb	bbbb bsss	xxxx xxxx	xxxx xxxx
A-register	0000 0000	0000 0000	1aaa aaaa	aaaa aaaa

The “a” bits are the fractional bits in the A-register (the current interval value) and the “x” bits are the fractional bits in the code register. The “s” bits are spacer bits which provide useful constraints on carry-over, and the “b” bits indicate the bit positions from which the completed bytes of the compressed image data are removed from the C-register. The “c” bit is a carry bit.

The detailed description of bit stuffing and the handling of carry-over will be given in the later part of this chapter.

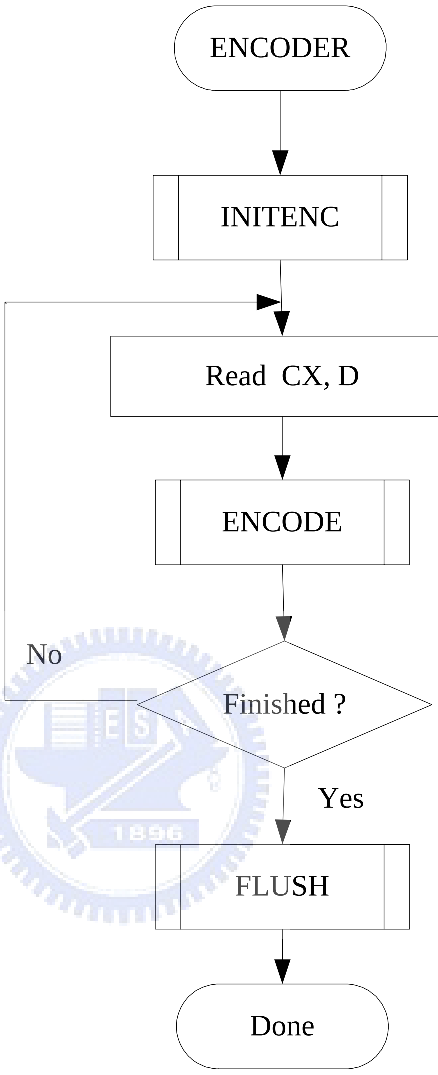


Figure 3-1 Encoder for the MQ-coder

3-3.2 Encoding a decision (ENCODE)

The ENCODE procedure determines whether the decision D is a 0 or not. Then a CODE0 or a CODE1 procedure is called appropriately. Often embodiments will not have an ENCODE procedure, but will call the CODE0 or CODE1 procedures directly to code a 0-decision or a 1-decision. Figure 3-2 shows this procedure.

3-3.3 Encoding a 1 or a 0

When a given binary decision is coded, one of two possibilities occurs – the symbol is either the more probable symbol or it is the less probable symbol. CODE1 and CODE0 are illustrated in Figure 3-3 and Figure 3-4. In these figures, CX is the context. For each context, the index of the probability estimate which is to be used in the coding operations and the MPS value are stored. MPS (CX) is the sense (0 or 1) of the MPS for context CX.

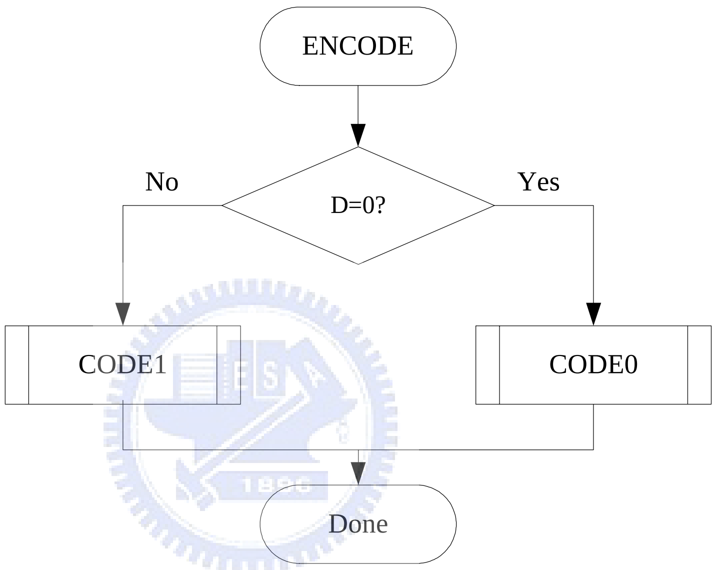


Figure 3-2 ENCODE procedure

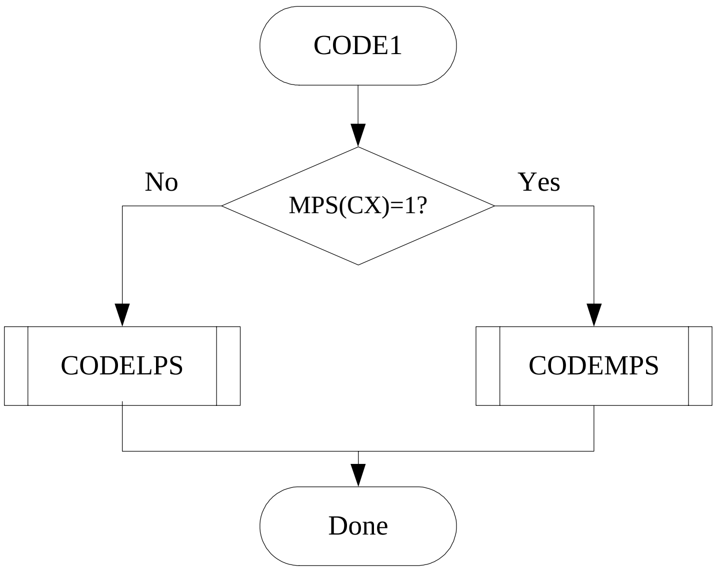


Figure 3-3 CODE1 procedure

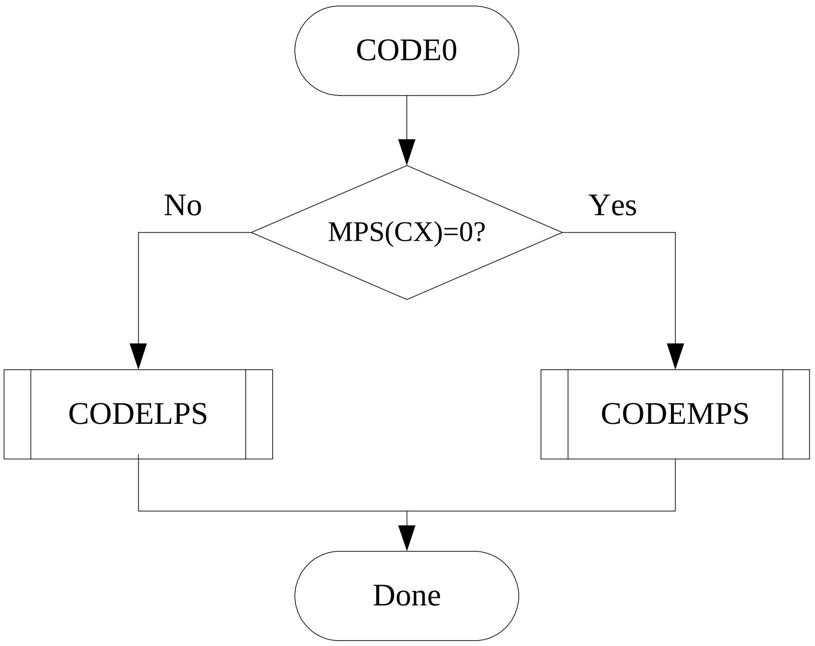


Figure 3-4 CODE0 procedure

3-3.4 Encoding an MPS or LPS (CODEMPS and CODELPS)

The CODELPS (Figure 3-5) procedure usually consists of a scaling of the interval to Qe(I(CX)), the probability estimate of the LPS determined from the index I stored for context CX. The upper interval is first calculated so it can be compared to the lower interval to confirm that Qe has the smaller size. It is always followed by a renormalization (RENORME). In the event that the interval sizes are inverted, however, the conditional MPS/LPS exchange occurs and the upper interval is coded. In either case, the probability estimate is updated. If the SWITCH flag for the index I(CX) is set, then the MPS(CX) is inverted. A new index I is saved at CX as determined from the next LPS index (NLPS) column in Table 3-2.

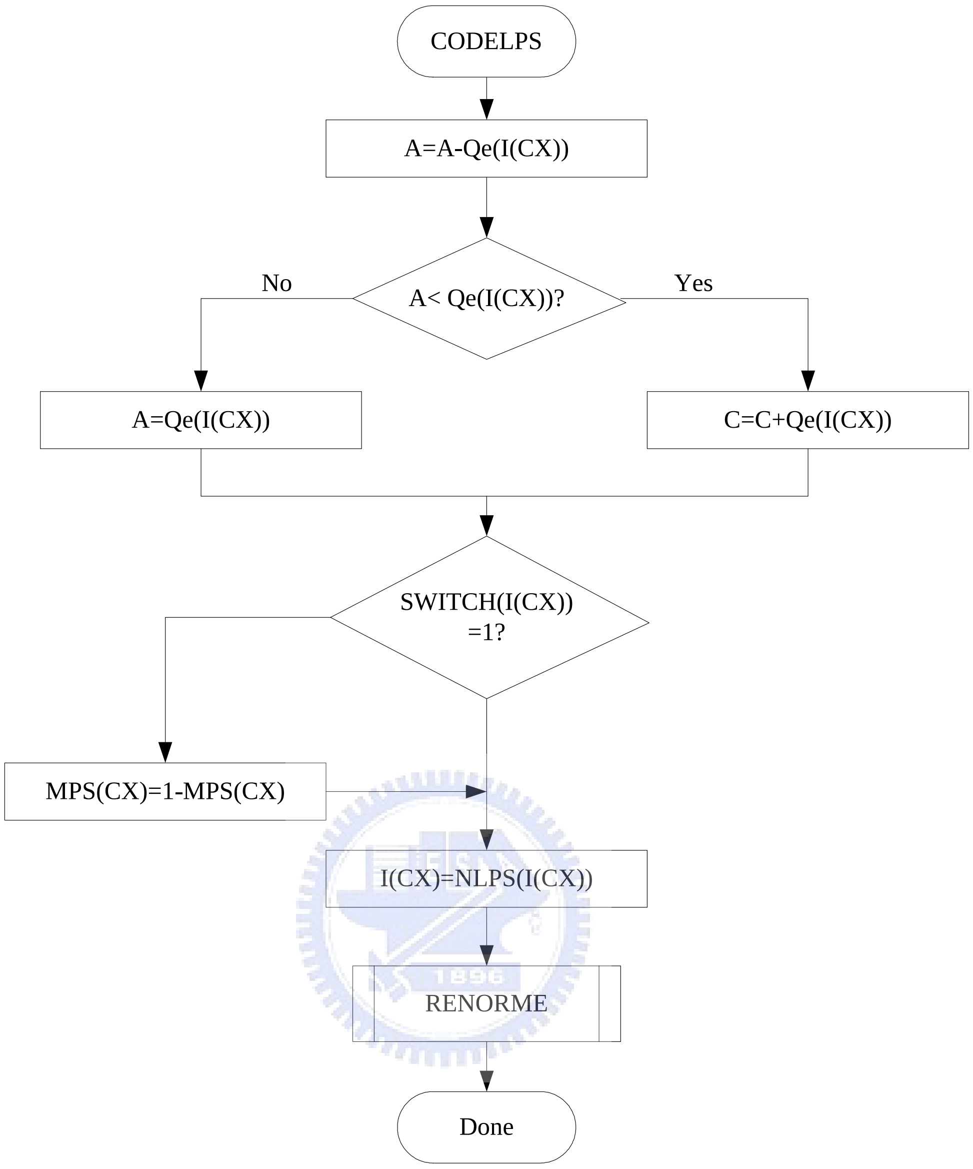


Figure 3-5 CODELPS procedure with conditional MPS/LPS exchange

Table 3-2 Qe values and probability estimation

Index	**Qe_Value**	**NMPS**	**NLPS**	**SWITCH**
0	0x5601	1	1	1
1	0x3401	2	6	0
2	0x1801	3	9	0
3	0x0AC1	4	12	0
4	0x0521	5	29	0

5	0x0221	38	33	0
6	0x5601	7	6	1
7	0x5401	8	14	0
8	0x4801	9	14	0
9	0x3801	10	14	0
10	0x3001	11	17	0
11	0x2401	12	18	0
12	0x1C01	13	20	0
13	0x1601	29	21	0
14	0x5601	15	14	1
15	0x5401	16	14	0
16	0x5101	17	15	0
17	0x4801	18	16	0
18	0x3801	19	17	0
19	0x3401	20	18	0
20	0x3001	21	19	0
21	0x2801	22	19	0
22	0x2401	23	20	0
23	0x2201	24	21	0
24	0x1C01	25	22	0
25	0x1801	26	23	0
26	0x1601	27	24	0
27	0x1401	28	25	0
28	0x1201	29	26	0
29	0x1101	30	27	0

30	0x0AC1	31	28	0
31	0x09C1	32	29	0
32	0x08A1	33	30	0
33	0x0521	34	31	0
34	0x0441	35	32	0
35	0x02A1	36	33	0
36	0x0221	37	34	0
37	0x0141	38	35	0
38	0x0111	39	36	0
39	0x0085	40	37	0
40	0x0049	41	38	0
41	0x0025	42	39	0
42	0x0015	43	40	0
43	0x0009	44	41	0
44	0x0005	45	42	0
45	0x0001	45	43	0
46	0x5601	46	46	0

3-3.5 Probability estimation

Table 3-2 shows the Qe value associated with each Qe index. The Qe values are expressed as hexadecimal integer’s fractions.

The estimator can be defined as a finite-state machine – a table of Qe indexes and associated next states for each type of renormalization (i.e., new table positions) – as shown in Table 3-2. The change in state occurs only when the arithmetic coder interval register is renormalized. This is always done after coding the LPS, and whenever the interval register is less than 0x8000 after coding the MPS.

After an LPS renormalization, NLPS gives the new index for the LPS probability estimate. If the switch is 1, the MPS symbol sense is reversed.

The index to the current estimate is part of the information stored for context CX. This index is used as the index to the table of values in NMPS, which gives the next index for an MPS renormalization. This index is saved in the context storage at CX. MPS (CX) does not change.

The procedure for estimating the probability on the LPS renormalization path is similar to that of an MPS renormalization, except that when SWITCH (I (CX)) is 1, the sense of MPS(CX) is inverted.

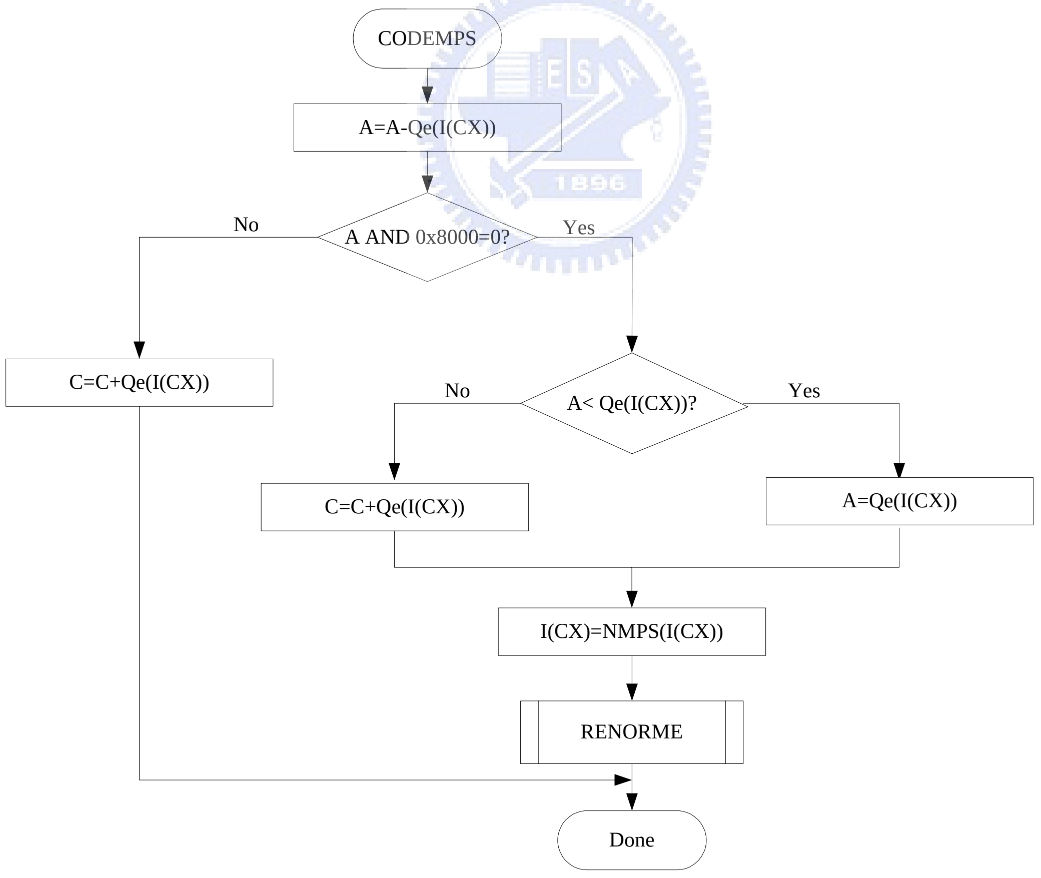


Figure 3-6 CODEMPS procedure with conditional MPS/LPS exchange

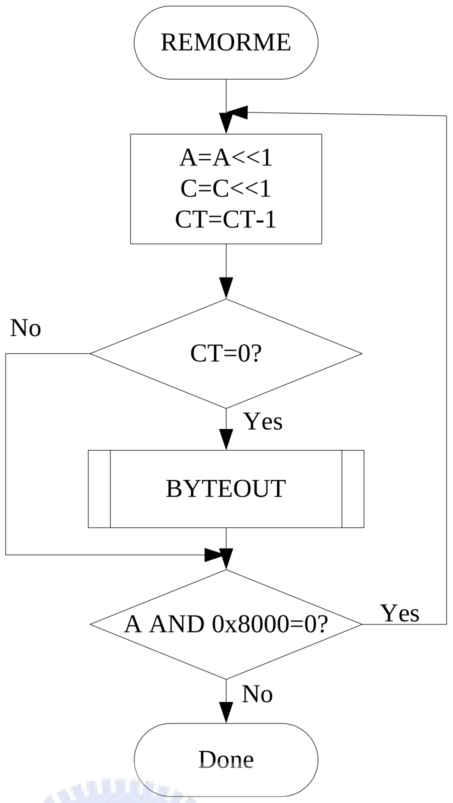


Figure 3-7 Encoder renormalization procedure

3-3.6 Renormalization in the encoder (RENORME)

The RENORME procedure for the encoder renormalization is illustrated in Figure 3-7. Both the interval register A and the code register C are shifted, one bit at a time. The numbers of shifts is counted in the counter CT, and when CT is counted down to zero, a byte of compressed image data is removed from C by the procedure BYTEOUT. Renormalization continues until A is no longer less than 0x8000.

3-3.7 Compressed image data output (BYTEOUT)

The BYTEOUT routine called from RENORME is illustrated in Figure 3-8. This routine contains the bit-stuffing procedures which are needed to limit carry propagation into the completed bytes of compressed image data. The conventions used make it impossible for a carry to propagate through more than the byte most recently written to the compressed image data buffer

The procedure in the block in the lower right section does bit stuffing after a 0xFF byte; the similar procedure on the left is for the case where bit stuffing is not needed.

B is the byte pointed to by the compressed image data buffer pointer BP. If B is not a 0xFF byte, the carry bit is checked. If the carry bit is set, it is added to B and B is again checked to see if a bit needs to be stuffed in the next byte. After the need for bit stuffing has been determined, the appropriate path is chosen, BP is incremented and the new value of B is removed from the code register “b” bits.

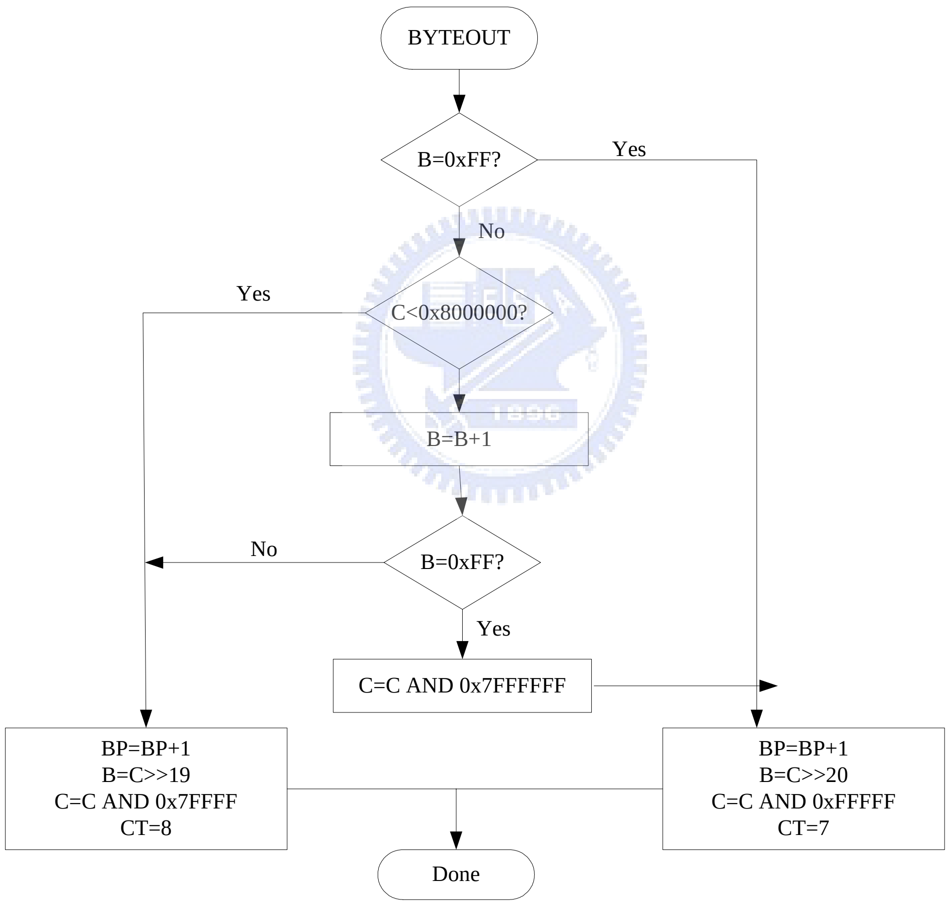


Figure 3-8 BYTEOUT procedure for encoder

3-3.8 Initialization of the encoder (INITENC)

The INITENC procedure is used to start the arithmetic coder. After MPS and I are initialized, the basic steps are shown in Figure 3-9.

The interval register and code register are set to their initial values, and the bit counter is set. Setting CT=12 reflects the fact that there are three spacer bits in the register which need to be filled before the field from which the bytes are removed is reached. BP always points to the byte preceding the position BPST where the first byte is placed. Therefore, if the preceding byte is a 0xFF byte, spurious bit stuff will occur, but can be compensated for by increasing CT.

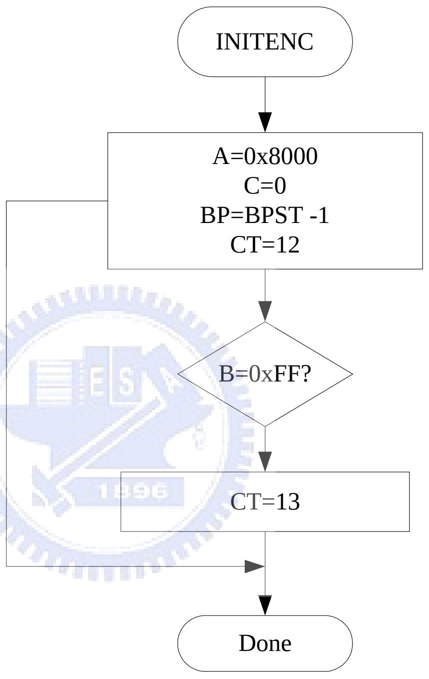


Figure 3-9 Initialization of the encoder

3-3.9 Termination of coding (FLUSH)

The FLUSH procedure shown in Figure 3-10 is used to terminate the encoding operations and generate the required terminating marker. The procedure guarantees that the 0xFF prefix to the marker code overlaps the final bits of the compressed image data. This guarantees that any marker code at the end of the compressed image data will be recognized and interpreted before decoding is complete.

The first part of the FLUSH procedure sets as many bits in the C-register to 1 as possible as shown in Figure 3-11. The exclusive upper bound for the C-register is the sum of the C-register and the interval register. The low order 16 bits of C are forced to 1, and the result is compared to the upper bound. If C is too big, the leading 1-bit is removed, reducing C to a value which is within the interval.

The byte in the C-register is then completed by shifting C, and two bytes are then removed. If the byte in buffer B, is an 0xFF then it is discarded. Otherwise, buffer B is output to the bit stream.

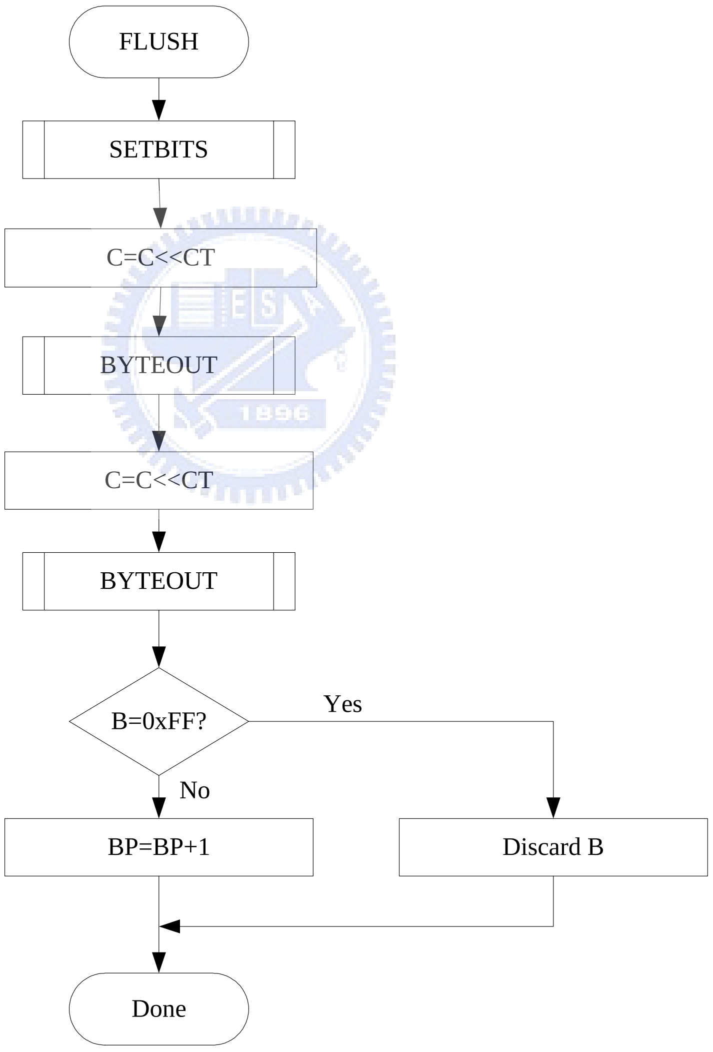


Figure 3-10 FLUSH procedure

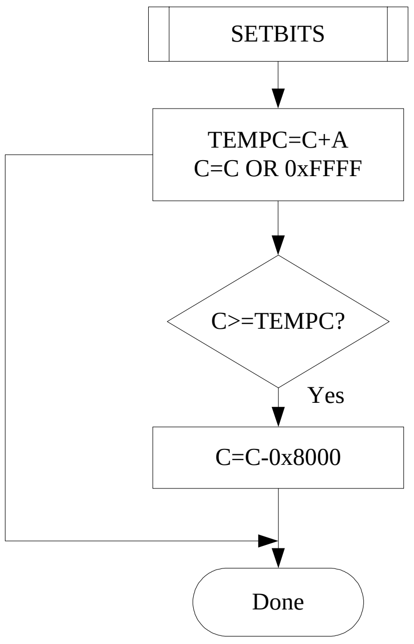


Figure 3-11 Setting the final bits in the C register

3-4 Method for Enhance Performance

In order to increase the throughput, the direct way is using pipeline architecture. Our goal is to receive one CX-D per clock cycle and to output the image compressed data as soon as possible. To achieve this, we combine all procedures except FLUSH procedure and separate them into independent operations as possible as we can. Hence, some operations could be done at the same clock cycle. The less the pipes we use, the sooner the compressed data will be outputted.

To reduce the complexity of separating the encoding procedures into the independent operations, we try to find out the regularity and simplify the algorithm. Using LUT (Look Up Table) is an efficient way to break up the loop and can finish the operation in one cycle.

We use three pipes and separate the AC encoding procedures into 12 operations. One CX-D pair operation could be finished in four clock cycle. The architecture is as below: (See Figure 3-12)

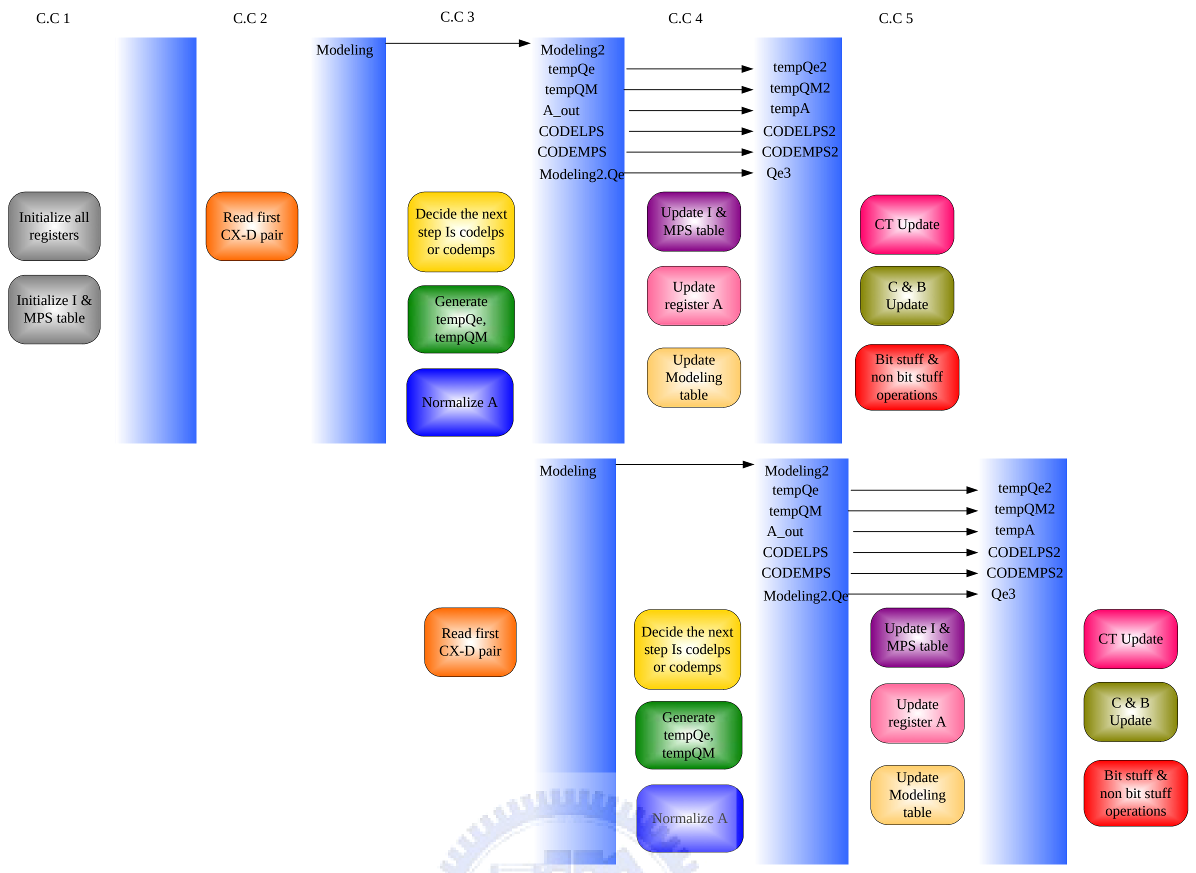


Figure 3-12 AC pipeline architecture

∗ **Initialize All Registers** and **Initialize I & MPS table**

Before starting AC Encoder, it is necessary to initialize all registers and tables. The initialization only needs one-clock-cycle time.

∗ **Read CX & D**

Read CX-D is stared after initialization cycle. It is able to push a CX-D pair to the pipes at every clock cycle.

∗ **Decide the next step is CODELPS or CODEMPS**

After getting the CX-D pair, Encoder will look up the MPS table by context value, then decides to enter the CODELPS procedure or CODEMPS procedure according to the D value.

If D xor MPS(CX)=0, then enter **CODEMPS** procedure.

If D xor MPS(CX)=1, then enter **CODELPS** procedure.

＊ **Generate tempQe and tempQM**

At the beginning of the CODELPS and CODEMPS procedures, A=A-Qe(I(CX)) has to be calculated first, next, check if A>Qe(I(CX)) or A>0x8000, we rearrange these equations as below:

A=A-Qe(I(CX))>Qe(I(CX)) ?➔ A> 2Qe(I(CX)) ?

A=A-Qe(I(CX))>0x8000 ? → A> Qe(I(CX))+0x8000 ?

Let tempQe=2Qe(I(CX))

tempQM=Qe(I(CX))+0x8000

In the next two clock cycles, tempQe and tempQM will be used to decide if it is needed to update the A-register and C-register.

＊ **Normalize Register A**

The interval A is kept in the range $0.75 \le A < 1.5$ by doubling it whenever the integer value falls below 0x8000.

Whenever A<0x8000, register A has to be normalized to make $A \ge 0x8000$. In order to finish the normalization in one clock cycle, we check how many leading 0-bit of register A, then do the shift left with the numbers of leading 0-bit directly.

Example：

If $A \ge 0x8000$, then A=A.

If $0x4000 \le A < 0x8000$, then A=(A<<1).

If $0x2000 \le A < 0x4000$, then A=(A<<2)

…….

If $0 \le A \le 1$, then A=(A<<15);

At this stage, not only finish the normalization, but recode the numbers of the leading

0-bit of register A. At the next stage, Register C has to be shifted left with the same bits as register A.

* **Update I & MPS table**
  - **I (CX)** is the pointer for LUT. In the encoding procedure, we can look up the Qe(I(CX)), NMPS(I(CX)), NLPS(I(CX)), SWITCH(I(CX)) by the I(CX).
  - **MPS (CX)** recodes the attribution of the symbol. It is used to decide the input symbol belongs to LPS(Less Probable Symbol) or MPS (More Probable Symbol). For the symbol '1', if MPS (CX) is 1, this symbol belongs to MPS; on the contrary, it belongs to LPS. On the other hand, for symbol '0', if MPS(CX) is 0, this symbol belongs to MPS, on the contrary, it belongs to LPS。

  At this stage, the I (CX) is updated according to CX.

* **Update Register A**

  At this stage, register A is updated to A=A-Qe or A=Qe according to tempQe and tempQM.

* **Update Modeling Structure**

  Modeling structure contains Qe, NMPS, NLPS, SWITCH, it is used in probability estimation.

* **CT Update**

  CT is a counter; it is decreased by the same bits when register A does the normalization. Whenever CT counts down to 0, the BYTEOUT procedure will be processed, then CT will be reset to 8 or 7. As the BYTEOUT procedure is processing, if the bit stuffing operation is chosen, CT is set to 7. If the non bit stuffing operation is chosen, CT is set to 8.

If register A is shifted left with **E1** bits, there are several cases happened about CT like below:

- CT>E1

  CT=CT-E1
- CT=E1 and Enter bit stuffing operation

  CT=7
- CT<E1 and Enter bit stuffing operation

  CT=7-(E1-CT)
- CT=E1 and Enter non bit stuffing operation

  CT=8
- CT<E1 and Enter non bit stuffing operation

  CT=8-(E1-CT)

The cases of CT=E1 and CT<E1 can be merged. So the cases only remain three:

- CT>E1

  CT=CT-E1
- $CT \le E1$ and Enter bit stuffing operation

  CT=7-(E1-CT)
- $CT \le E1$ and Enter non bit stuffing operation

  CT=8-(E1-CT)

* **C & B Update** and **Bit stuff & no bit stuff operations**

Register C recodes the probably lower boundary and register B points to the compressed image data. The operations for this two registers are the most complicated, because many cases have to be considered. We wish to finish all operations of C-register in one clock cycle, so any possible case has to be concerned.

Three operations of register C have to be finished at the same clock cycle.

That's why we have to set up two registers, tempB1 and lC_out, to do the different operations at the same time.

Define

E is the bit numbers of A-register normalization.

lC_out =normalize(C, E)

tempB1=lC_out + normalize(Qe,E)

lC_out2 =normalize(C, CT)

tempB2=normalize(C, CT)+normalize(Qe,CT)

There are several cases happened about register C:

- Non bit stuffing operation
  - ● C is not changed

    C((18+E) downto 0)=lC_out((18+E) downto 0)

    C(27 downto (19+E))=(others=>'0');
  - ● C=C+Qe

    C((18+E) downto 0)=tempB1((18+E) downto 0)

    C(27 downto (19+E))=(others=>'0');
- Bit stuffing operation
  - ● C is not changed

    C((19+E) downto 0)=lC_out((19+E) downto 0)

    C(27 downto (20+E))=(others=>'0');

● C=C+Qe

C((19+E) downto 0)=tempB1((19+E) downto 0)

C(27 downto (20+E))=(others=>'0');

There are several cases happened about register B:

⅄ C=C+Qe

● Bit stuffing operation

B='0' & tempB2[26:20] or B=tempB2[27:20]

● No bit stuffing operation

B=tempB2[26:19]

⅄ C is not changed

● Bit stuffing operation

B='0' & lC_out2[26:20] or B=lC_out2[27:20]

● No bit stuffing operation

B=lC_out2[26:19]

＊ **FLUSH Procedure**

Before terminating AC encoder, it is necessary to run FLUSH procedure to generate the terminating marker. This procedure uses about 7 clock cycles; it doesn't affect the overall performance.

＊ **Additional registers for pipeline architecture**

For pipeline architecture, we have to setup additional registers for every pipe, because the value at different stage has different meaning. That is why pipeline architecture will increase the chip area.

3-5 State Machine

We setup the state machine for AC according to the pipeline architecture. (See Figure 3-13) There are five states, IDLE, INITENC, READ_CX, ENCODE, FLUSH, FLUSH1, and FINISH.

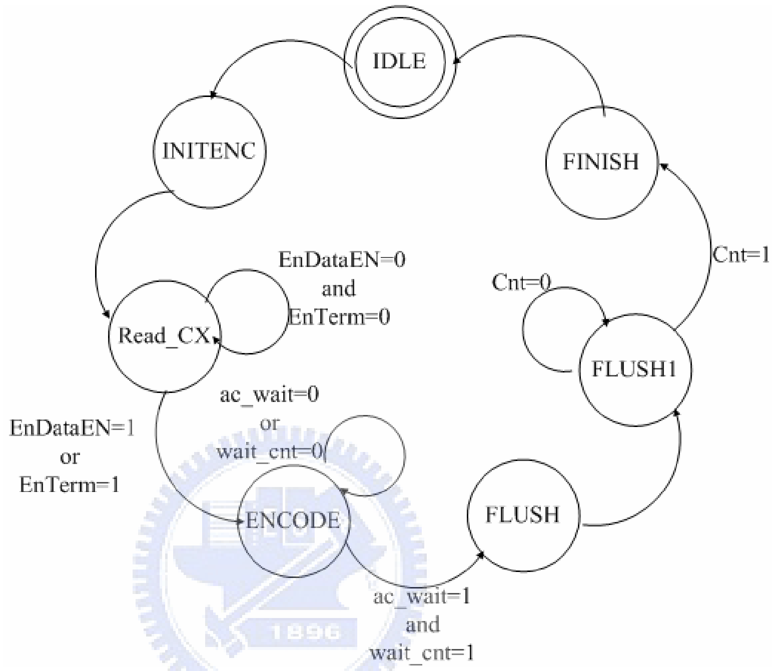


Figure 3-13 AC encoder state machine

- **IDLE**: when system is powered on, the AC encoder enters the IDLE state.
- **INITENC:** initialize the AC encoder in the INITENC state.
- **READ_CX:** stay in this state until the first CX-D pair comes in. If terminal signal is set in this state, the encoder will enter the ENCODE state.
- **ENCDOE:** AC is encoding data continuously.
- **FLUSH:** before the AC coder is terminated, it will enter FLUSH state.
- **FLUSH1:** this state is also for FLUSH procedure.
- **FINISH:** when the AC encoder is completely terminated, it will enter FINISH state then go back to the IDLE state.

3-6 Pin Definition

For communicating with EBCOT correctly, we define the interface of the AC in the Table 3-3.

Table 3-3 Pin Definition

Name	**Direction**	**Width**	**Description**
Clk	Input	1	Clock signal
EnEN	Input	1	AC Encoder chip enable
FIFO_Busy	Input	1	Indicate FIFO is full
EnDataEN	Input	1	Input data valid signal
En_CX	Input	5	Context label from EBCOT
En_D	Input	1	Symbol from EBCOT
EnTerm	Input	1	Indicate AC to terminate the process
EnDone	Output	1	Indicate AC has finished coding
EnCD_EN	Output	1	Output data valid signal
AC_Busy	Output	1	Indicate AC is busy or not
EnCD	Output	8	Encoded data
CD_length	Output	8	Numbers of encoded data
State	Output	3	Indicate AC state

3-7 Timing Diagram

The interface timing is drawn in Figure 3-14.

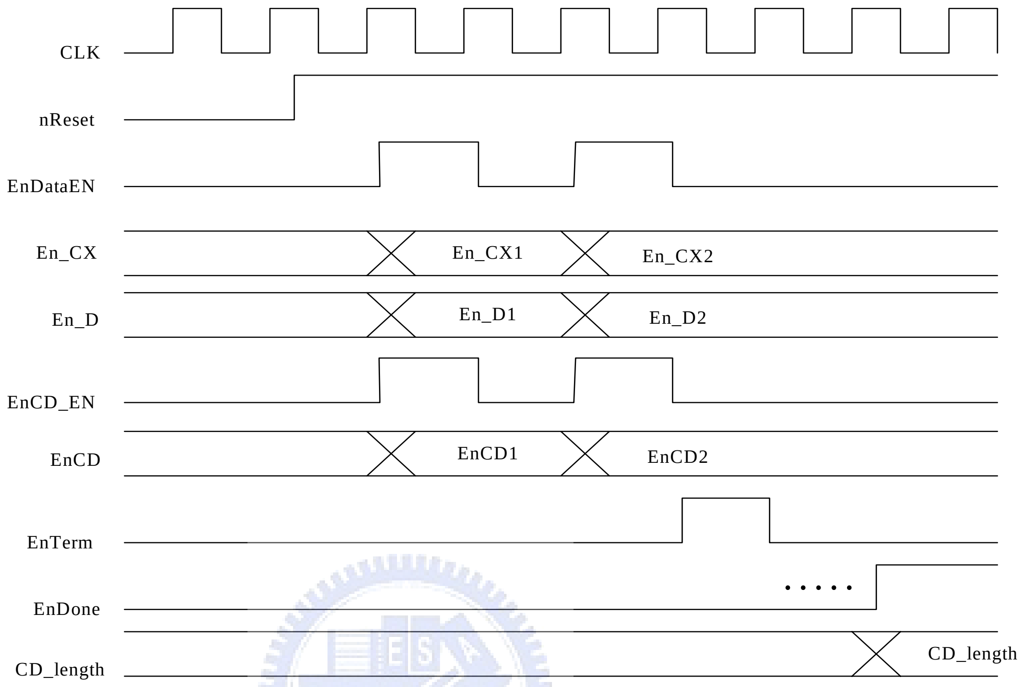


Figure 3-14 AC Timing Diagram

3-8 Achievements & Comparison

3-8.1 Achievements

∗ **ASIC**

⅄ Process	: TSMC 0.25 um
⅄ Operation Frequency	: 71 MHz
⅄ Gate Counts	: 8.97 K
⅄ Power	: 16.8 mW
⅄ Throughput	: 1 CX-D pair/c.c.
⅄ Core size	: 474 um x 474 um
⅄ Die size	: 1169 um x 1169 um

＊ **Altera FPGA**

- Part : APEX20K1000EFC672-2
- Operation Frequency: 39.98 MHz
- Area : 5 %
- Throughput : 1 CX-D pair/c.c.

3-8.2 Comparison

We find out some AC codec developed by others and list their performance in Table 3-4. Three-stage pipeline architecture is usually used to enhance the performance of the AC. The throughputs are almost the same. The throughput of Wu is 2 CX-D pairs/clock cycle but its maximum operation frequency is only 40MHz.When our chip runs at the 71MHz, the throughput is almost the same as Wu's.

Because our process is more advanced than others, the comparison between area and power may not be fair.

Table 3-4 comparison with others

Design	Architecture	Process	Die size	Operation Freq.	Power	Throughput
Our AC encoder	3 stage pipeline	TSMC 0.25um	1.169mmx 1.169 mm	71 MHz	16.8mW	1 CX-D pair/clock cycle
Wang et al.[16]	3 stage pipeline	TSMC 0.35um	2mmx2mm (Codec)	200 MHz	74.91mW	1 CX-D pair/clock cycle
Hsiao et al.[17]	pipeline	0.35 um CMOS	3.345mmx 3.318mm	142.8 MHz	131.8mW	1.103 clock cycles/CX-D pair
Wu et al.[18]	3 stage pipeline	TSMC 0.35 um	Gate count: 10.597 K	40 MHz	N/A	2 CX-D pairs/clock cycle

Chapter 4   AHB Wrapper Design

4-1 Introduction

To make the JPEG2000 coprocessor more applicable, it is a good choice to wrap it in AHB (Advanced High-performance Bus) interface. The ARM CPUs are widely used in the embedded systems, so the chip which is compatible with AMBA will be more applicable and popular in the market.

The JPEG2000 chip we addressed is defined as the coprocessor which cooperates with an ARM processor. In this chapter, we will explain how the JPEG2000 coprocessor communicates with the ARM CPU and introduce what AMBA is. Besides, the most important is how we wrap the JPEG2000 coprocessor with AHB interface.

4-2 Work Theory

The *Advanced Microcontroller Bus Architecture* (AMBA) specification defines an on-chip communications standard for designing high-performance embedded microcontrollers.

Three distinct buses are defined within the AMBA specification:

➤ **The Advanced High-performance Bus (AHB)**

The AMBA AHB is for high-performance, high clock frequency system modules. The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macro cell functions. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques.

- **The Advanced System Bus (ASB)**

  AMBA ASB is an alternative system bus suitable for use where the high-performance features of AHB are not required.

  We don’t use ASB in our design.

- **The Advanced Peripheral Bus (APB)**

  The AMBA APB is for low-power peripherals.

  AMBA APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. APB can be used in conjunction with either version of the system bus.

4-2.1 Objectives of the AMBA specification

The AMBA specification has been derived to satisfy four requirements:

- To facilitate the right-first-time development of embedded microcontroller products with one or more CPUs or signal processor.
- To be technology-independent and ensure that highly reusable peripheral and system macro cells can be migrated across a diverse range of IC processes and be appropriate for full-custom, standard cell and gate array technologies.
- To encourage modular system design to improve processor independence, providing a development road-map for advanced cached CPU cores and the development of peripheral libraries.
- To minimize the silicon infrastructure required to support efficient on-chip and off-chip communication for both operation and manufacturing test.

4-2.2 A typical AMBA-based microcontroller

An AMBA-based microcontroller typically consists of a high-performance system backbone bus (AMBA AHB or AMBA ASB), able to sustain the external memory bandwidth, on which the CPU, on-chip memory and other Direct Memory Access (DMA) devices reside. This bus provides a high-bandwidth interface between the elements that are involved in the majority of transfers. Also located on the high-performance bus is a bridge to the lower bandwidth APB, where most of the peripheral devices in the system are located. (See Figure 4-1)

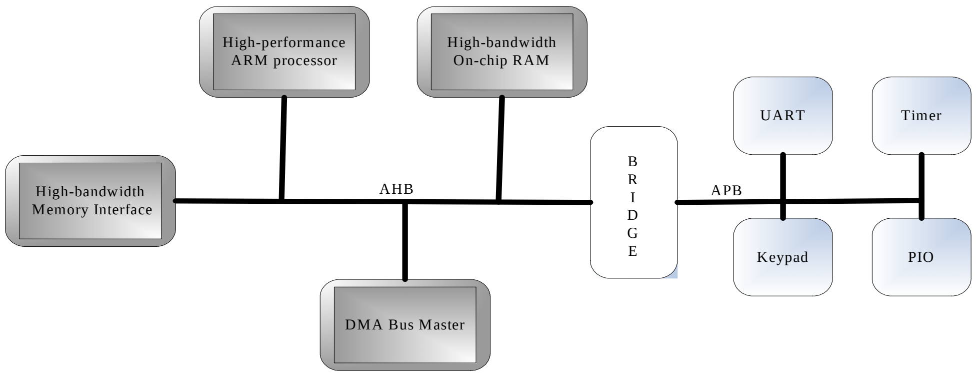

High-performance ARM processor
High-bandwidth On-chip RAM
UART
Timer
High-bandwidth Memory Interface
AHB
BRIDGE
APB
Keypad
PIO
DMA Bus Master

Figure 4-1 A typical AMBA AHB -based system

4-2.3 AMBA AHB

AHB is a new generation of AMBA bus which is intended to address the requirements of high-performance synthesizable designs. AMBA AHB is a new level of bus which sits above the APB and implements the features required for high-performance, high clock frequency systems including:

- Burst transfers
- Split transactions
- Single cycle bus master handover
- Single clock edge operation
- Non-tristate implementation
- Wider data bus configurations (64/128 bits)

4-2.4 Bus interconnection

The AMBA AHB bus protocol is designed to be used with a central multiplexer interconnection scheme. Using this scheme all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to control the read data and response signal multiplexer, which selects the appropriate signals from the slave that is involved in the transfer.

Figure 4-2 illustrates the structure required to implement an AMBA AHB design with three masters and four slaves.

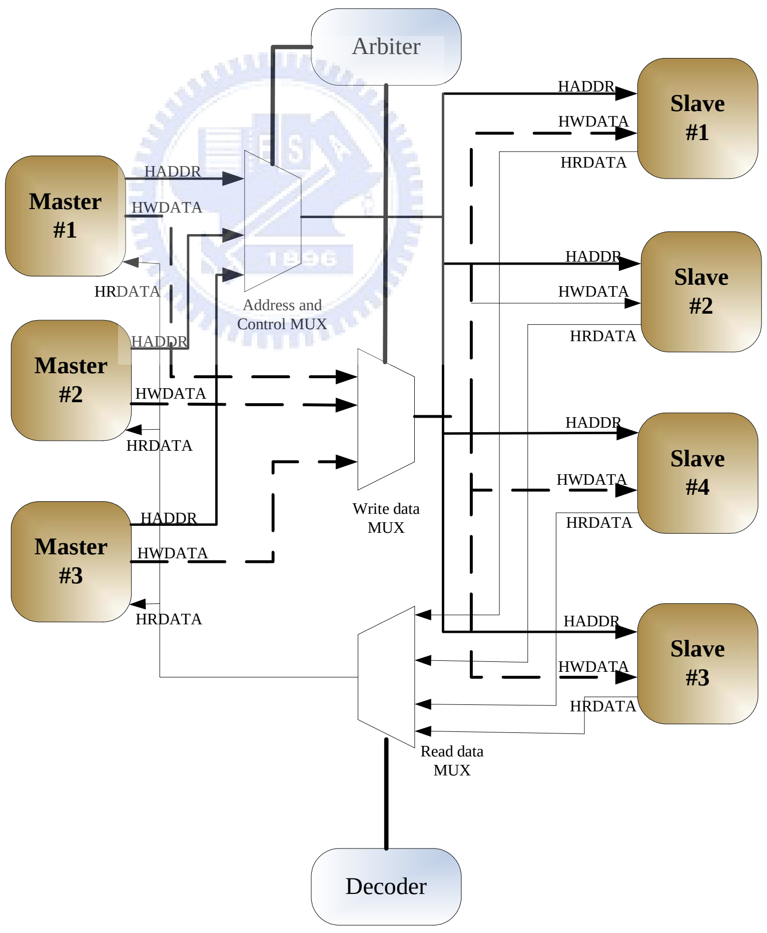


Figure 4-2 Multiplexer interconnection

4-2.5 Overview of AMBA AHB operation

Before an AMBA AHB transfer can commence the bus master must be granted access to the bus. This process is started by the master asserting a request signal to the arbiter. Then the arbiter indicates when the master will be granted use of the bus.

A granted bus master starts an AMBA AHB transfer by driving the address and control signals. These signals provide information on the address, direction and width of the transfer, as well as an indication if the transfer forms part of a burst. Two different forms of burst transfers are allowed:

- Incrementing bursts, which do not wrap at address boundaries
- Wrapping bursts, which wrap at particular address boundaries

A write data bus is used to move data from the master to a slave, while a read data bus is used to move data from a slave to the master.

Every transfer consists of:

- An address and control cycle
- One or more cycles for the data

The address cannot be extended and therefore all slaves must sample the address during this time. The data, however, can be extended using the **HREADY** signal. When LOW this signal causes wait states to be inserted into the transfer and allows extra time for the slave to provide or sample data.

During a transfer the slave shows the status using the response signals, **HRESP [1:0]**:

- **OKAY** The OKAY response is used to indicate that the transfer is progressing normally and when **HREADY** goes HIGH this shows the transfer has completed successfully.
- **ERROR** The ERROR response indicates that a transfer error has occurred and the transfer has been unsuccessful.

- **RETRY and SPLIT** Both the RETRY and SPLIT transfer responses indicate that the transfer cannot complete immediately, but the bus master should continue to attempt the transfer.

In normal operation a master is allowed to complete all the transfers in a particular burst before the arbiter grants another master access to the bus. However, in order to avoid excessive arbitration latencies it is possible for the arbiter to break up a burst and in such cases the master must re-arbitrate for the bus in order to complete the remaining transfers in the burst.

4-3 Timing Analysis

Before defining the overall system architecture, it is better to analyze the timings of all components. Because the ram (SSRAM) in the Logic Module is the single port ram, ARM CPU, QDWT and system controller will access the ram in the coding procedure. So we have to find out the best way to organize them to achieve good performance. The tile size we define is 128 x 128; QDWT will code two layers, so there are five QCB coding time units.

First of all, the coprocessor has to wait until the ARM CPU move one tile image to SSRAM, the SSRAM bus belongs to the ARM CPU at this time. Then the memory bus hands to the QDWT. After QDWT finishes coding the QCB1, the coded data are stored in the CBM1 (one of the ping-pong buffers). When entropy coder starts coding the data outputted from QCB1 (CBM1), we let the QDWT to code the QCB2 and the coded data are stored in the CBM2 (the other one of ping-pong buffers). When QDWT is coding QCB2, the memory bus still belongs to QDWT, so the system controller can’t write the compressed image data outputted from the entropy coder back to SSRAM. So two FIFO are setup to store the output data and address from the system controller. When the system controller gains the memory bus, it will read the data and address from two FIFO and write data back to the SSRAM according to the corresponding address. After ping-pong buffers are full, QDWT will hand the memory bus to the system controller. The system controller will hand the memory bus back to the QDWT when one buffer of ping-pong buffers is empty and the QDWT does no finish coding all QCBs. If the QDWT has finished a tile coding, and entropy coder has finished either, after the system controller write all compressed data back to SSRAM, the memory bus will hand to ARM CPU to prepare the next tile image.

The timing analysis is illustrated as below:

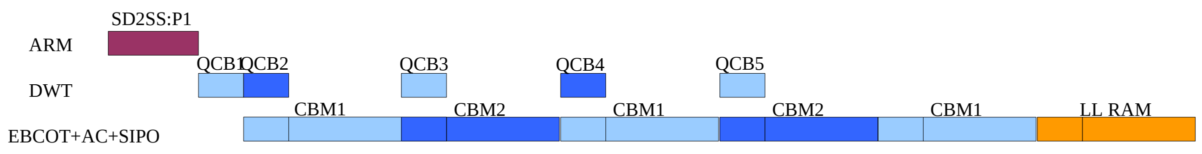


Figure 4-3 JPEG2000 Coprocessor timing analysis

4-4 AHB JPEG2000 Coprocessor Block Diagram

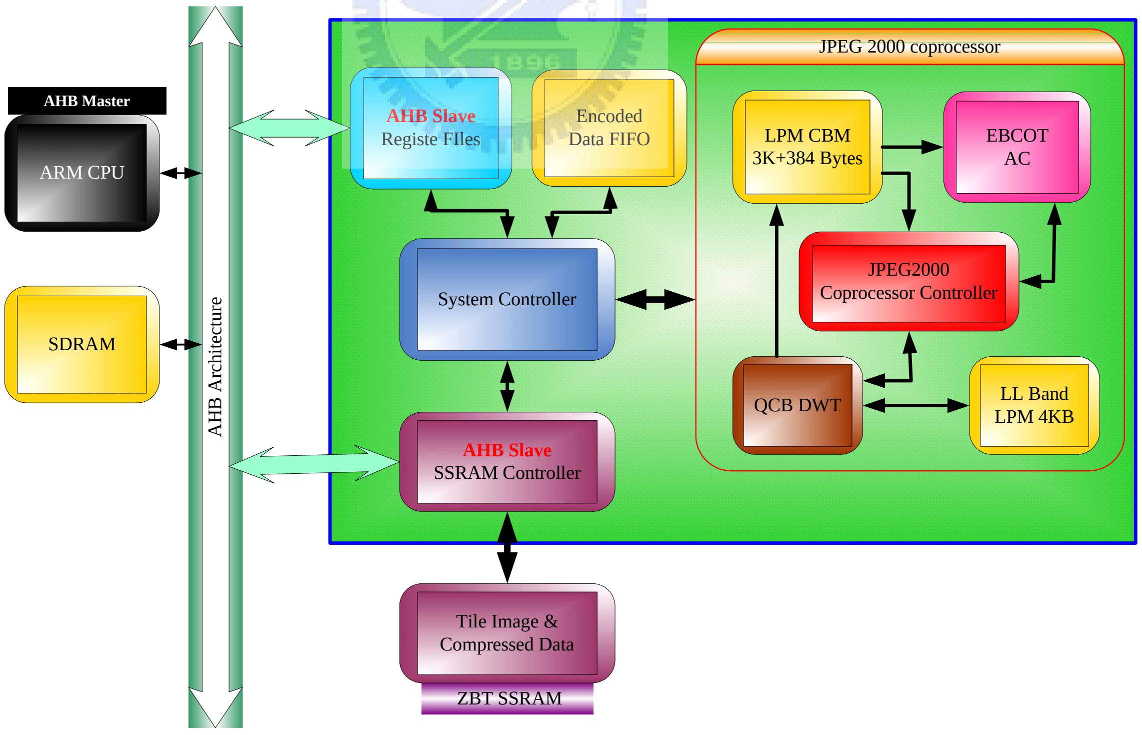


Figure 4-4 AHB-based JPEG2000 Coprocessor block diagram

In the integrator, the JPEG2000 coprocessor is an AHB slave device. ARM CPU can control the JPEG2000 coprocessor in AHB timing. The modules in the FPGA are introduced as below:

- **JEPG2000 Coprocessor**: this is the JPEG2000 coprocessor we introduced in the chapter 2.
- **System Controller**: this is the overall system controller. It controls the work flow of all system. We will explain its control flow in the following section.
- **Encoded Data FIFO**: the compressed image data will be stored in the FIFO temporarily until the system gains the memory bus.
- **AHB Slave Register Files**: the register definition is in section 4-5. We wrap this module in AHB salve interface because ARM CPU will access this module.
- **AHB Slave SSRAM Controller**: this is the ram controller for SSRAM in the logic module. we wrap it in AHB slave interface because ARM CPU will access SSRAM, too.

  Some others modules are not appeared in the above block diagram, but it is necessary for AHB architecture.

- **AHB Decoder**: it decodes the address from AHB master (ARM CPU) and tells which AHB slave is selected by AHB master.
- **AHBMuxS2M** : this is the multiplexer to multiplex the read data bus from AHB slaves.
- **AHBAPBSys**: there are several modules included in this component. This is AHB to APB bridge and APB to control the switches and the LEDs in the logic module. We show the system status in the LED.

4-5 Register Definition

For image configuration, we use two registers to store the image information. One status/command register is setup, in write mode, it is a command register, in read mode, it is a status register to show the overall system current status. The registers are defines in the following address:

Table 4-1 Register Definition

Address	**Register Name**	**Description**
0xC4000000	Status/Command [31:0]	Status=0x80: ARM CPU owns the system bus Status=0x40: JPEG2000 coprocessor owns the system bus Ststus=other value: JPEG2000 coprocessor stays in idle Command=0x40 : Enable the JPEG2000 Coprocessor Command=0x80 : hand over the system bus to ARM CPU Command=other value: ARM CPU still owns the system bus
0xC4000004	SSRAM_Addr [31:0]	Tell the JPEG2000 coprocessor the beginning address where the compressed data should be stored in
0xC4000008	Image_Information [31:0]	For image configuration [31:23] number of tiles [22:15] width of the tile [14:7] length of the tile [6 :5]number of layers

4-6 Work Flow

This is the work flow of ARM CPU.

Capture an image by a camera first and then do the pre-operation for the source image. The pre-operation contains to separate the image to the tiles (tile size is 128x128) and to do the normalization for every pixel and to reorder the coding order for tiles. Then, ARM CPU will send the image configuration information to the JPEG2000 coprocessor. Up to now, just finish the image initialization operation. After the image initialization, ARM CPU starts to move a tile data to the SSRAM on the LM (Logic Module) and then write command register in 0x40 to enable the JPEG2000 coprocessor. When the JPEG2000 coprocessor is coding the tile, ARM CPU keeps on polling the status register until status=0x80. If status is equal to 0x80, it means the JEPG2000 coprocessor has coded the tile. ARM CPU will check if all tiles has been coded, if not, it will move the next tile to the SSRAM and enable the coprocessor again; if it does, ARM CPU will capture a new image and run the overall procedure from the beginning.

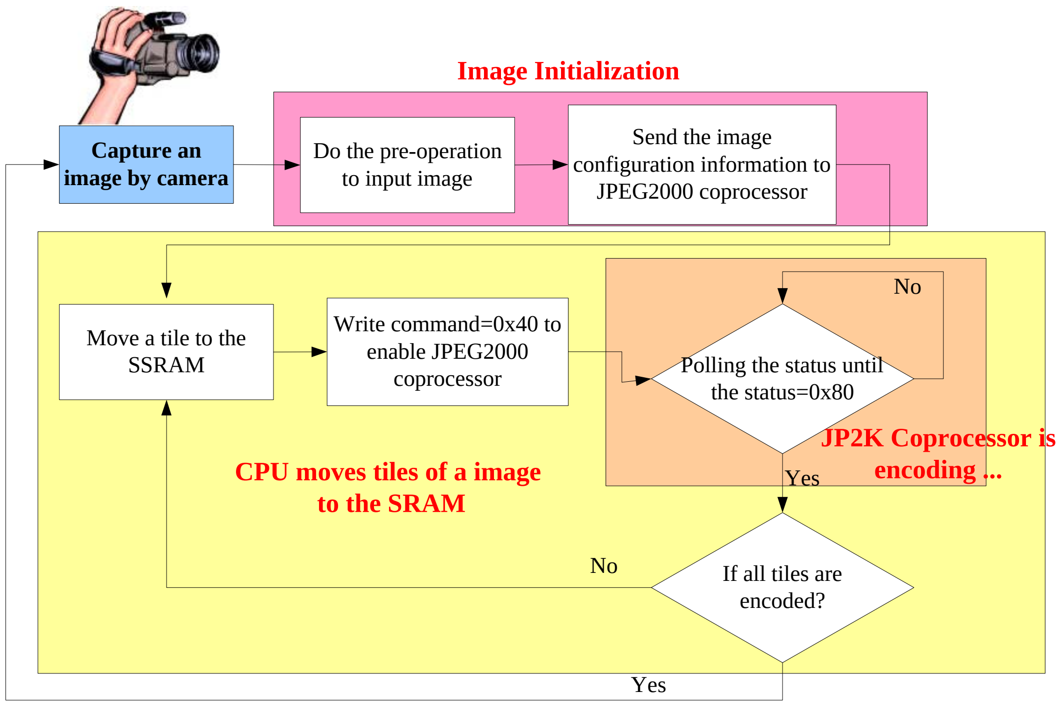


Image Initialization
Capture an image by camera
Do the pre-operation to input image
Send the image configuration information to JPEG2000 coprocessor
Move a tile to the SSRAM
Write command=0x40 to enable JPEG2000 coprocessor
No
Polling the status until the status=0x80
JP2K Coprocessor is encoding ...
Yes
CPU moves tiles of a image to the SRAM
No
If all tiles are encoded?
Yes

Figure 4-5 Overall system work flow

4-7 System Controller Design

The system controller is designed according the timing analysis mentioned in section 4-3. The following two figures are the system controller flow chart and its state machine.

Figure 4-6 is the flow chart of the system controller to control the system work flow.

After power on, the system controller will enter IDLE state. Wait until the status register is equal to 0x40, it will enter the Image-Configuration state and receive the image information. Then, the controller will do the image initialization for the JPEG2000 coprocessor. After the image initialization, the controller enables the coprocessor to encode a tile and then the memory bus will be transferred to the DWT. DWT reads data from the SSRAM and generates coefficients to the ping-pong buffers. When buffers are full, the controller will transfer the memory bus to the AC, hence, AC could move the compressed data back to the SSRAM. When all compressed data are moved back, the controller will check if the tile is finished coding, if not, the memory bus will be transferred back to the DWT; if it does, the controller will set the status register to 0x80 to ask ARM CPU to move the next tile. Then, the controller will check if all tiles are encoded, if not, the controller will wait until the status register is equal to 0x40, it means the next tile is ready and the controller will enable the coprocessor again. If all tiles are coded, the controller will go back to the IDLE state.

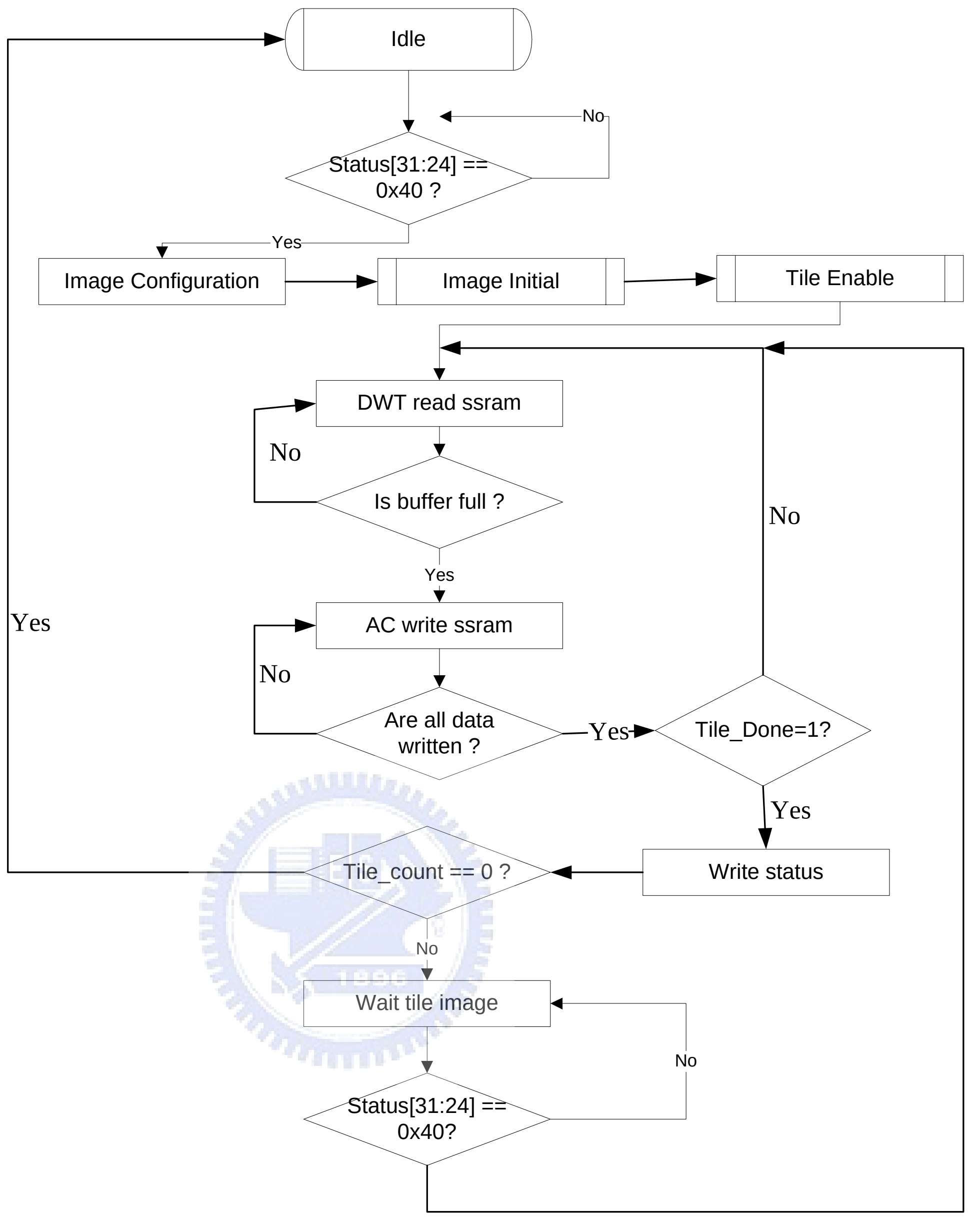


Figure 4-6 System controller flow chart

We setup a state machine for the system controller (See Figure 4-7).

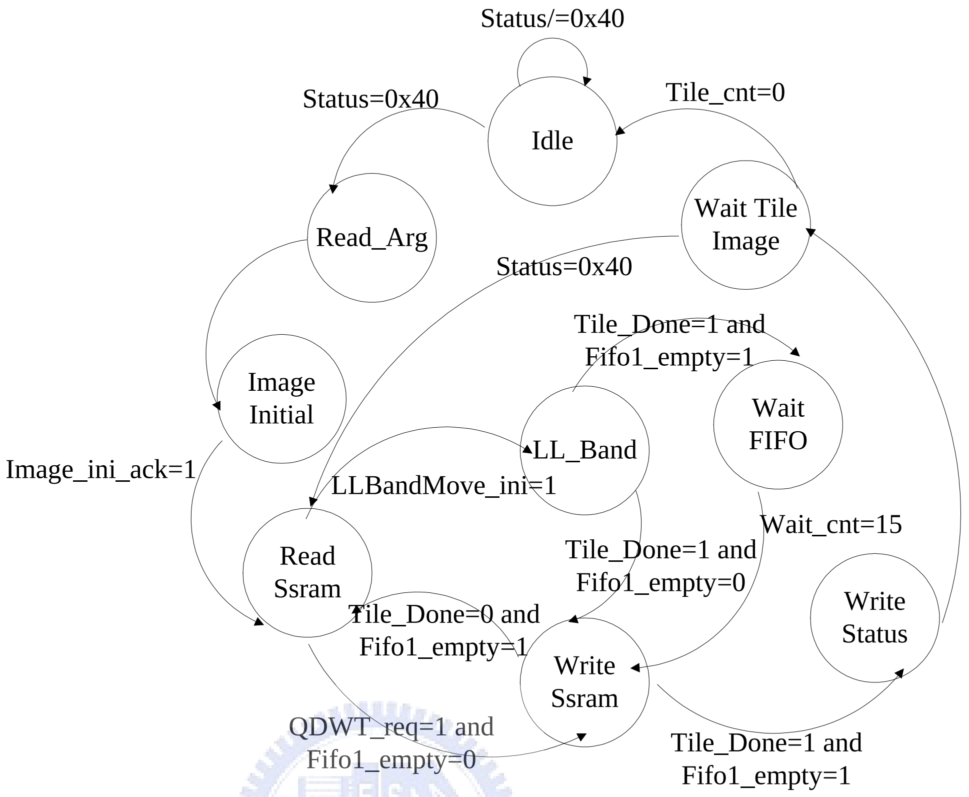


Figure 4-7 System controller state machine

∗ **IDLE:** do nothing in this state, stay until status register is equal to 0x40.

∗ **READ_ARG**: receive the image information.

∗ **IMAGE_INITIAL:** do the image initialization for the JPEG2000 coprocessor.

∗ **READ_SSRAM:** DWT is reading the SSRAM.

∗ **WRITE_SSRAM:** AC is writing the SSRAM.

∗ **WRITE_STATUS:** set the status register to 0x80.

∗ **LL_BAND:** DWT is reading LL_BAND LPM.

∗ **WAIT_FIFO:** wait until FIFO becomes empty.

∗ **WAIT_TILE_IMAGE:** wait until the next tile image is ready.

4-8 Pin Definition

According to the definition of the AHB Slave interface, we defines the AHB-based JPEG2000 coprocessor chip I/O as in Table 4-2.

Table 4-2 AHB-based JPEG2000 chip pin definition

Name	Source	Direction	Width	Description
HCLK Bus clock	Clock source	Input	1	This clock times all bus transfers. All signal timings are related to the rising edge of HCLK
HRESETn Reset	Reset controller	Input	1	The bus reset signal is active LOW and is used to reset the system and the bus.
HSIZE Transfer size	Master	Input	2	Indicate the size of the data transfer, which is typically with the length of byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes, up to a maximum of 1024 bits.
HTRANS Transfer type	Master	Input	2	Indicate the type of the current transfer, which can be NONSEQ, SEQ, IDLE or Busy.
HWRITE Transfer direction	Master	Input	5	When HIGH this signal indicates a write transfer and when LOW a read transfer.
HADDR Address bus	Master	Input	32	The 32-bit system address bus
HREADY Transfer done	Slave	Inout	1	When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. NOTE: Slaves on the bus require HREADY as both an input

				and an output signal.
HDATA Data bus	Mater or Slave	Inout	32	This bidirectional bus contains write data bus and read data bus. The write data bus is used to transfer data from the master to the bus slaves during write operation. The read data bus is used to transfer data from the bus slaves to the bus master during read operation.
HRESP Transfer response	Slave	Output	2	The transfer response provides additional information on the status of the transfer. Four different responses are provided, OKAY, ERROR, RETRY, SPLIT.
HBUSREQ Bus request	Master	Output	1	A signal from bus master x to the bus arbiter which indicates that the bus master requires the bus.
HLOCK locked transfer	Master	Output	1	When HIGH this signal indicates that the master requires locked access to the bus and no other master should be granted the bus until this signal is LOW.
SCLK SRAM clock	FPGA	Output	1	Clock signal of SSRAM
SDATA data bus	FPGA	Inout	32	Bidirectional data bus of SSRAM
SADDR Address bus	FPGA	Output	19	Address bus of SSRAM
SnOE	FPGA	Output		SSRAM output enable
SnCE	FPGA	Output		SSRAM chip enable
SnWR	FPGA	Output		SSRAM write enable
SnCKE	FPGA	Output	3	SSRAM clock enable

4-9 Memory Distribution

This is the memory distribution of the SSRAM for JPEG2000 coprocessor.

- 0xC2000000 ~ 0xC200FFFF : input tile image for QDWT
- 0xC2010000 ~ 0xC2028000: the compressed image data from entropy coder. They are in bit plane order and pass order. We allocate 256 bytes for every bit plane pass in a code block.
- 0xC2028000 ~ 0xC2029000: for rate and distortion and header information. It is allocated 256 bytes for every rate and distortion. The header information only uses 4 bytes and is located in the fixed address predefined.

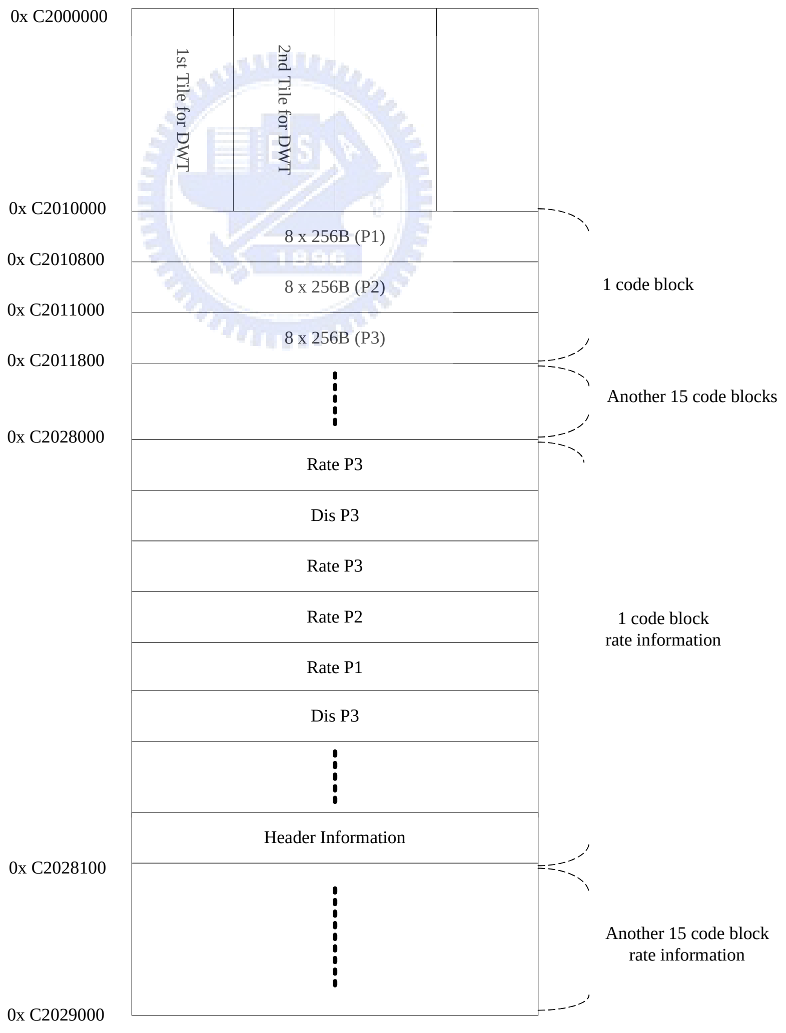


Figure 4-8 Memory distribution for JPEG2000 Coprocessor

Chapter 5

Achievements and Perspectives

5-1 Achievements

∗ **FPGA-based (Hardware Only)**

The performance of the JPEG2000 coprocessor we achieve in the FPGA level is list in Table 5-1

Table 5-1 the performance of the JPEG2000 coprocessor on FPGA

Operation Frequency	25 MHz
Throughput	4 Mpixels/s
Compression Format	Lossless
Frame Rate	256 x256 YCbCr 20 FPS
	512 x512 Grey 10 FPS

We have developed a version of JPEG2000 encoder previously and the architecture of the hardware is different from the present version. The comparison between these two versions and the comparison of operation time between the software and the hardware for individual version are listed in Table 5-2.

Table 5-2 the comparison between software and hardware

Events	**Seconds (New)**	**Seconds (Old)**
ARM moves one tile to SSRAM (Software)	4.228	0.17
The operation time of JPEG2000 coprocessor compression (Hardware)	0.007371	0.63
The operation time of packet header functions (Software)	1.225	0.02
Operation frequency of ARM CPU	100	120
Operation frequency of FPGA	20	25

The operation time in Table 5-2 is calculated by ARM CPU at the run time. We can see the software part is the bottle neck for the present version; on the contrary, the hardware is the bottle neck for the previous version. So, for the hardware of the JPEG2000 only, we do decrease 98.83% computing time. The frame rate of the JPEG2000 coprocessor can reach **135 tiles/sec** while the clock frequency is 20 MHz. The throughput of the hardware is **2.21 Mpixels/sec** at the operation frequency 20 MHz. The performance is enhanced greatly.

∗ **ASIC-based**

The specification of the JPEG2000 coprocessor chip is list in Table 5-3. We can compare our performance with those introduced in Section 5-2.

Table 5-3 JPEG2000 coprocessor chip specification

Technology	TSMC 0.25 um
Package	208-pin CQFP
Core size	3.15mm x 3.15mm
Die size	3.95mm x 3.95mm
Operation frequency	41 MHz
Internal rams	8.25 KB
Throughput	9.37 MB/s
Power consumption	740 mW

5-2 JPEG2000 Codec in the Market

"Analog Devices Inc., a global leader in high-performance semiconductors for signal processing applications, today (June 12, 2003) announced that NHK, a major national Japanese broadcast company, has incorporated ADI's JPEG2000 image compression chip into its latest Hi-Vision high-definition television (HDTV) advancement. This May (2003), NHK Science & Technical Research Laboratories (STRL) announced the development of the world's first single, real-time encoder/decoder board based on the JPEG2000 standard, enabled by ADI's JPEG2000 chip." The information is referred to Reference [8].

We find out that there are three companies have developed JPEG2000 ASIC, ADI, AMPHION, and TECHSOFT.

∗ **TECHSOFT (Javelin530)** [7]

The Javelin530 JPEG2000 codec is a high-performance application specific solution enabling leading edge image, video, and audio compression/ decompression/ transmission applications. The core is compliant with the ISO/IEC 15444-1 JPEG 2000 Image Coding Core System Standard for both lossy and lossless compression/decompression of images.

Table 5-4 Preliminary IC specification

Package	208-pin LQFP
Fabrication technology	0.25 u
Fabrication foundry	TSMC
Internal memory	15 KB
Clock	166 MHz
External SDRAM required	256K x 32 x 2 (VCD)

	512K x 32 x 2 (VCD)
Input format (Audio)	I2S compatible
Input format (Video)	YCbCr (4:2:2)
Gate count	100K
Power consumption	150 mW
SRAM interface	ARM
Video bit rate	2.0~50.0 Mbits/s 720 x 480x 29.97 fps
Cost per chip	$10.00
Die area	5mm x 5mm

∗ **ANALOG DEVICES, Inc (ADV202) [6]**

The ADV202 is a single-chip JPEG2000 CODEC targeted at video and high bandwidth image compression applications that will benefit by the enhanced quality and feature set provided by the JPEG2000 (J2K) -ISO/IEC 15444-1 image compression standard. The features are listed below

- Complete single-chip JPEG2000 compression/decompression solution for video and still images.
- Patented SURF (Spatial Ultra-efficient Recursive Filtering) technology enables low power and low cost wavelet based compression
- Supports both 9/7 and 5/3 wavelet transform with up to 6 levels of transform
- Programmable tile/image size with widths up to 2048 pixels in three-component 4:2:2 interleaved mode, and up to 4096 pixels in single-component mode. Maximum tile/image height is 4096 pixels.
- Input rate of 65 Msamples/sec for irreversible mode or 40 Msamples/sec for

reversible mode.

- 12mm x 12mm 121-ball fpBGA, speed grade 115 MHz, price $35.18
- 13mm x 13mm 144 fpBGA, speed grade 150 MHz, price $47.06

＊ **AMPHION (CS6590 JPEG2000 Codec)[9]**

The CS6590 JPEG2000 codec is a high performance application specific solution enabling leading edge image compression, decompression and transmission applications. The core is compliant with ISO/IEC 15444-1 JPEG2000 Image Coding System Standard and makes possible both lossless and lossy compression and decompression of image data at ratios of up to 50:1.

Table 5-5 CS6590 ASIC Cores

Fabrication technology	0.18 u
Fabrication foundry	TSMC
Memory	59 KB
Clock	150 MHz
Gate count	210K
Compression ratios	50:1
Throughput (Msamples/s)	Encoding : 60 Decoding : 20

5-3 Improvement in the future

To compare our chip with others in the market, our chip may still not so powerful. But we believe our chip has great potential to be developed as a high performance JPEG2000 codec because our memory usage is saving and we have known where the bottle neck is.

In the following, we will explain how to improve the performance in the future.

∗ For Arithmetic Entropy Coder (AC)

- ⅄ If the previous stage, EBCOT Tier-1, is in pass serial mode, the AC we design can provide good performance, the area of AC is almost the same with EBCOT and it can deal with the output of EBCOT every clock cycle. But if the EBCOT is in pass parallel mode, using three ACs we design to deal with three passes from EBCOT will cause the area too large, so it is better to improve the AC.
- ⅄ If we hope the AC could deal with three passes from EBCOT at the same time and the area is small, we may do it in two way:
  - ● Analyze the distribution of context label for three passes

    CX1：0~13　　CX2：14, 15, 16　　CX3：0 ~ 18

    The context labels are in the range 0 to 18. We set up a table with 19 integer elements to save these labels. Pass 3 needs an individual table because every context label may happen. Pass 1 and Pass 2 can share one table because the range of the context label for pass 1 and pass 2 are different. In this way, we can reduce the area of one table.
  - ● Only remove one table can not reduce much area. Because the situation, that AC has to receive CX-D pairs every clock cycle in three passes, will not happen frequently, we can reduce the area efficiently by reducing the throughput and this reduction will not affect the overall system throughput but can reduce the overall system area.

- It is essential to develop the AC codec which can be applied in the JPEG2000 codec.

- For AHB-based JPEG2000 encoder, it needs AHB-based DMA controller to enhance its system performance. Right now, it is ARM CPU to move the tile image from SDRAM in the core module to SSRAM in the logic module. On one hand, it wastes much CPU time to move the tile image, if there were a DMA controller, CPU can do other calculation when DMA is moving the tile image. On the other hand, CPU uses more than 10 clock cycles to move one word of a tile image. If there were a DMA controller, it only uses one clock cycle to move one word. That's why DMA controller can enhance much performance.

  In the ARM/Integrator platform, if we want to add the DMA controller in the JPEG2000 coprocessor, the role of the JPEG2000 coprocessor in the ARM/Integrator has to be a AHB Master.

- The QDWT only supports the image size of a multiple of 128. If it can support any image size, this JPEG2000 coprocessor will be more valuable.

- The memory allocation for compressed image data now is not good enough to achieve the best performance because we separate the compressed data pass by pass. In this way, we need 96KB memory space, so we have to use the off-chip memory, SSRAM, in the logic module to store the compressed data. Another disadvantage is the memory bus of the SSRAM will be very busy and it is hard to pipeline the work of the components. If we can rearrange the compressed data from three passes, the extra 2.25 KB FIFO will be needed but the memory space for the compressed data only need 16 KB for tile size 128x128. So, 70 % of memory space is reduced and we don't need to store the compressed data in the off-chip memory but in the internal memory inside the FPGA. The internal ram size of this FPGA is 40KB, so the space is enough. On the other hand, we can avoid the situation that the memory bus is too busy to pipeline the work. We think this can enhance much system performance in FPGA application.

＊ In the future, if it is planed to develop an AHB-based JPEG2000 coprocessor chip, the power consumption has to be concerned and the chip area can't be too big. So, we had better not to use too much internal ram.

Although the JPEG2000 is not popular in the consumer electronics now because of its area and cost, it is still valuable to develop JPEG2000 ASIC. For medical image, the cost is not the most important part for people, the high image quality and high resolution can help doctors to make sure if our body is healthy. For a criminal case, the image quality is also much concerned. The high image quality and resolution can help police to recognize the criminal or easily find out the clue to break the criminal case.

So, we believe that JPEG2000 still has a region belonged to it to yield unusually brilliant results. That's why we do our best to finish this system and we know it worth.

Reference

[1] F. Jelinek, *Probabilistic Information Theory*, McGraw-Hill, New York, 1986

[2] ISO/IEC, ISO/IEC 15444-1, Information Technology-JPEG2000 image coding system, 2000.

[3] JPEG 2000 影像壓縮技術, 吳炳飛 胡益強 瞿忠正 蘇崇彥, 全華科技圖書股份有限公司, 92 年 4 月

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

[5] http://www.eedesign.com.tw/article/document/dc965.htm#3

[6] http://www.analog.com/index.html

[7] http://www.techsoft.com.tw/Chinese/intro.htm

[8] http://www.analog.com/Analog_Root/sitePage/pressReleaseHome/0,2145,ContentID%253D22292%2526aind%253D%2526resourceWebLawID%253D,00.html

[9] http://www.amphion.com/

[10] Yun-Tai Hsiao, Hung-Der Lin, Kun-Bin Lee, Chein-Wei Jen, "High-speed memory-saving architecture for the embedded block coding in JPEG2000," Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on , Volume: 5 , 2002, pp. 133 -136

[11] JPEG2000 戴顯權/陳政一 紳藍出版社 p165-p172 2002 年 11 月

[12] http://www.jpeg.org/jpeg2000/index.html

[13] Pei-Chun Chen, "Design of an efficient Pass-Parallel Context Formation Codec for JPEG2000", National Chiao Tung University, July 2004

[14] B.F. Wu and C.F. Lin, "Analysis and architecture design for high performance JPEG2000 coprocessor," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 2, pp.

225-228, May, 2004.

[15] *Advanced Microcontroller Bus Architecture* Specification Rev2.0, ©copyright ARM Limited 1999.

[16] 王經楷, A High-throughput and Low-Power Arithmetic CODEC Design for Multiple Image Compression Standards, Departments of Electronics Engineering, National Chiao Tung University, June 2002

[17] Yun-Tai Hsiao, Hung-Der Lin, Kun-Bin Lee and Chein-Wei Jen, HIGH-SPEED MEMORY-SAVING ARCHITECTURE FOR THE EMBEDDED BLOCK CODING IN JPEG2000, Departments of Electronics Engineering, National Chiao Tung University, IEEE 2002.

[18] Ping-Hsun Wu, The Research on Chip Implementation of JPEG2000 Tier-1 Encoder, Department of Electrical Engineering, National Tsing Hua University, June 2002.

Appendix

A-1 Development Flow

∗ We use the C language to develop the firmware for the JPEG2000 coprocessor. For ARM/Integrator platform, we can use ADS (ARM Developer Suite) to develop, build, and debug C, C++ or ARM assembly language programs.

∗ How to write Synplify Pro and Quartus II script files

Because we use Synplify Pro to do the synthesis and use Quartus II to do the Place & Route, we'll explain how to write the script files below:

⅄ **Synplyfy Pro**

At the DOS command prompt:

% synplify_pro –batch Tcl_script_name.tcl

The example of the Synplify script file:

```
==============================================================

Project –new D:/Test/proj.prj

Project –save

# device options

Set_option –technology APEX20KE
Set_option –part EP20K1000E

Set_option –package FC672

Set_option –speed_grade -2

# add_file options

Add_file –vhdl    D:/Test/bp_table.vhd

Add_file –vhdl    D:/Test/CBM.vhd

# compilation/mapping options

Set_option symbolic_fsm_compiler false

# map options

Set_option –frequency 100.0

Set_option –result_file “CBM.vqm”

Project –run

Exit
```

- **Quartus II 3.0**

For the first time to create a new project, it is recommended to use GUI to do the settings we require. Then, we can use the function provided by Quartus II, “Generate Tcl Files for Project”. This function will generate the tcl file which contains all settings of this project.

For the second time to run the implementation in the Quartus II, we can run it in the batch mode.

At the DOS command prompt:

a. change the directory to where the script file is saved

```
% c:/quartus/bin/quartus_sh -s
```

b. Create or load the project

```
% source Tcl_file_name.tcl

% exit
```

c. Compile with project.csf

```
% c:/quartus/bin/quartus_cmd project_name –c project_name.csf
```

A-2 Verification Environment

＊ Pre-simulation and Post-simulation in ModelSim

Before we synthesize the RTL code, we do the pre-simulation in ModelSim to make sure the function of the hardware we designed is right. Modify the RTL codes until pre-simulation is pass, we do the post-simulation after the hardware is implemented by Quartus II. If the result of the post-simulation is the same as the result of the pre-simulation, it means the percentage of the hardware is workable is 90% above.

＊ To make sure that the hardware is workable, we will read out the data stored in the memory and write to a text file through the AXD (ARM Debugger for Windows). Then, we will compare this file with the expected file.

＊ After the hardware is correct, the firmware will packet the compressed data in JPEG2000 header, then, we can decode it by JPEG2000 decoder. The JPEG2000 decoder is developed by BCB.

A-3 Pin Map Table for JPEG2000 Coprocessor In Test Mode

＊ DWT Pin Map Table

Table A- 1 DWT Pin Map Table

Test_out pins	Pin Map	Description
Output ports for test mode		
Test_out(67:52)	dual_ram_1_dataout_test (Com_Sel=0000000)	//dual ram_1output port
	dual_ram_2_dataout_test (Com_Sel=0000001)	//dual ram_2output port
Test_out(99:68)	ram_1_out_test (Com_Sel=0000010)	//single ram_1 output port
	ram_2_out_test (Com_Sel=0000011)	//single ram_2 output port
Test_out(100)	out_valid_row (Com_Sel=0000100)	//row output valid
	out_valid_sync (Com_Sel=0000101)	//sync output valid
	out_valid_col (Com_Sel=0000110)	//column output valid
Test_Out(116: 101)	data_out_row (Com_Sel=0000100)	//row output
	data_out_sync (Com_Sel=0000101)	//sync output
	data_out_col (Com_Sel=0000110)	//column output
Input ports for test mode		
Test_In(2)	dual_ram_1_cena_test (Com_Sel=0000000)	//dual ram_1 enable write port
	dual_ram_2_cena_test (Com_Sel=0000001)	//dual ram_2 enable write port
	ram_1_cen_test (Com_Sel=0000010)	//single ram_1 enable port
	ram_2_cen_test (Com_Sel=0000011)	//single ram_2 enable port
Test_In(3)	dual_ram_1_wena_test (Com_Sel=0000000)	//dual ram_1 write port
	dual_ram_2_wena_test (Com_Sel=0000001)	//dual ram_2 write port

	ram_1_wen_test (Com_Sel=0000010) ram_2_wen_test (Com_Sel=0000011)	//single ram_1 r/w port //single ram_2 r/w port

Test_In(9 :4)	dual_ram_1_write_add_test (Com_Sel=0000000)
	dual_ram_2_write_add_test(Com_Sel=0000001)
	ram_1_add_test (Com_Sel=0000010)
	ram_2_add_test (Com_Sel=0000011)
		//dual ram_1 write address port
		//dual ram_2 write address port
		//single ram_1 address port
		//single ram_2 address port

Test_In(10)	dual_ram_1_cenb_test (Com_Sel=0000000) dual_ram_2_cenb_test (Com_Sel=0000001)	//dual ram_1 enable read port //dual ram_2 enable read port
Test_In(11)	dual_ram_1_wenb_test (Com_Sel=0000000) dual_ram_2_wenb_test (Com_Sel=0000001)	//dual ram_1 read port //dual ram_2 read port
Test_In(17:12)	dual_ram_1_read_add_test (Com_Sel=0000000) dual_ram_2_read_add_test (Com_Sel=0000001)	//dual ram_1 read address port //dual ram_2 read address port
Test_In(33:18)	dual_ram_1_datain_test (Com_Sel=0000000) dual_ram_2_datain_test (Com_Sel=0000001)	//dual ram_1 datain port //dual ram_2 datain port
Test_In(41: 10)	ram_1_datain_test (Com_Sel=0000010) ram_2_datain_test (Com_Sel=0000011)	//single ram_1 datain port //single ram_2 datain port

＊ EBCOT Tier-1 Pin Map Table

Table A- 2 EBCOT Pin Map Table

		Test_mode '100	**Test_mode '101**
	Test_mode		
	Ecx_Sel		
Test_0	CLK		
Test_1	nReset		
INPUT			

	Tier1_ini		
	Tier1Start		
	orient		
	Height		
	Width		
	Weighting		
	Numbitplane		
	data_CB		
	FIFO1		
	FIFO2		
	FIFO3		
Test_2			Test_CENA
Test_3			Test_WENA
Test_4…13			Test_AA
Test_14…15			Test_DA
Test_16			Test_CENB
Test_17…26			Test_AB
OUTPUT			
	Tier1_ini_ack		
	Tier1Ready		
	rdAdd_CB		
	CX1		
	D1		
	EnDataEN		
	TerminalAC1		

	CX2		
	D2		
	EnDataEN2		
	TerminalAC2		
	CX3		
	D3		
	EnDataEN3		
	TerminalAC3		
	RD3		
	Dot3		
	RDOUT		
Test_52…53			Test_QB
Test_54		Tier1_ini_ack	
Test_55		Tier1Ready	
Test_56…65		rdAdd_CB	
Test_66…69		te_CX1	
Test_70		te_D1	
Test_71		te_EnDateEN	
Test_72		TerminalAC1	
Test_73…74		te_CX2	
Test_75		te_D2	
Test_76		te_EnDataEN2	
Test_77		TerminalAC2	
Test_78…81		te_CX3	
Test_82		D3	

Test_83		EnDataEN3	
Test_84		TerminalAC3	
Test_85		RDout	
Test_86…89		DCState	
Test_90…92		Tier1State	
Test_93		te_buf1_full	
Test_94		te_buf2_full	
Test_95		te_buf1_empty	
Test_96		te_buf2_empty	
Test_97…100		te_NBC_B	
Test_101		te_rlc	
Test_102…105		te_Reg_change_A	
Test_106…115		te_AddrE	
Test_116…120		te_Rsig1E	
Test_121…125		te_Rsig2E	
Test_126…130		te_RsignE	
Test_131…134		te_RMagE	
Test_135		te_PS12S	
Test_136		te_PS1E	
Test_137		te_PS2E	
Test_138		te_PS3S	
Test_139		te_PS3E	
Test_140		te_WDS	
Test_141		te_WDE	
Test_142		te_RDS	

Test_143…145		te_bitplane	

＊ FSM Controller Pin Map Table

Table A- 3 FSM Controller Pin Map Table

Test_out port	**Pin Map**	**Description**
FSM for test mode (Com_Sel = 1000000)		
Test_out(55 : 52)	FSM_insert	//FSM
Test_out(56)	Reset	//async. Reset
Test_out(57)	tile_num_valid	
Test_out(58)	EC_INIT_ACK	
Test_out(59)	EC_QCB_ACK	
Test_out(60)	EC_QCB_Done	
Test_out(64 : 61)	QCB_counter	
Test_out(65)	Tier1ready	
Output ports for test mode (Com_Sel = 1000000)		
Test_out(71 : 66)	CB_Y	//code block size_Y
Test_out(77 :72)	CB_X	//code block size_X
Test_out(78)	tile_num_valid	// tile_num_valid
Test_out(79)	Tile_INIT_OK	// EC_Tile_INIT
Test_out(80)	EC_QCB_INIT_temp	// EC_QCB_INIT
Test_out(81)	EC_QCB_Done_ACK_reg	// EC_QCB_Done_ACK
Test_out(82)	Tier1_nReset_reg	// Tier1_nReset //reset for EBCOT
Test_out(83)	Tier1Start_t	// Tier1Start //start EBCOT when wait_r state

Test_out(84)	reset_all_t	// reset_all_t
Test_out(90: 85)	tile_numY	// tile_num_Y
Test_out(96:91)	tile_numX	// tile_num_X
Output ports for test mode (Com_Sel = 100000x)		
Test_out(109: 97)	CB_HL_rd (Com_Sel=1000000) CB_HL_cd_cp1 (Com_Sel=1000001) CB_LH_cd_cp1 (Com_Sel=1000010) CB_HH_cd_cp1 (Com_Sel=others)	//rd starting address //AC starting address
Test_out(122: 110)	CB_LH_rd (Com_Sel=1000000) CB_HL_cd_cp2 (Com_Sel=1000001) CB_LH_cd_cp2 (Com_Sel=1000010) CB_HH_cd_cp2 (Com_Sel=1000011)	//rd starting address //AC starting address
Test_out(135 :123)	CB_HH_rd (Com_Sel=1000000) CB_HL_cd_cp3 (Com_Sel=1000001) CB_LH_cd_cp3 (Com_Sel=1000010) CB_HH_cd_cp3 (Com_Sel=1000011)	//rd starting address //AC starting address

＊ AC Pin Map Table

Table A- 4 AC Pin Map Table

Test_out port	**Pin Map**
Test_out(52)	tEnDone1
Test_out(53)	tEnCD_EN1
Test_out(54)	AC_busy1
Test_out(62:55)	tEnCD1
Test_out(70:63)	tCD_length1

Test_out(73:71)	tState1

∗ CBM Pin Map Table

Table A- 5 CBM Pin Map Table

Test_out port	**Pin Map**
For CBM Controller in Test Mode	
Test_out(52)	tbuf1_full
Test_out(53)	tbuf2_full
Test_out(56:54)	B1_P_state
Test_out(59:57)	B2_P_state
Test_out(62:60)	obit_plane
Test_out(63)	tCBM_INIT_ACK;
Test_out(64)	tCBM_Done
Test_out(73:65)	tEBCOT_Data_out
Test_out(74)	oNCP_valid_CB
Test_out(79:75)	oNCP_CB
Test_out(80)	oNZB_valid_CB
Test_out(83:81)	oNZB_CB
For CBM Buffers in Test Mode	
Test_in (2)	B1_CLK
Test_in (3)	B2_CLK
Test_in (4)	M1_CEN
Test_in (5)	M2_CEN
Test_in (15:6)	M1_Addr
Test_in (25:16)	M2_Addr

Test_in (34:26)	M1_D
Test_in (43:35)	M2_D
Test_in (44)	M1_WEN
Test_in (45)	M2_WEN
Test_out(60:52)	M1_Q
Test_out(69:61)	M2_Q

＊ SIPO&FIFO Pin Map Table

Table A- 6 FIFO&SIPO Pin Map Table

Test_out port	**Pin Map**
For FIFO in Test Mode	
Test_out(52)	tP1_Empty
Test_out(53)	P1_Full
Test_out(69:54)	tP1_DataOut(15: 0)
Test_out(70)	tP2_Empty
Test_out(71)	P2_Full
Test_out(87:72)	tP2_DataOut(15: 0)
Test_out(88)	tP3_Empty
Test_out(89)	P3_Full
Test_out(105:90)	tP3_DataOut(15: 0)
For SIPO in Test Mode	
Test_out(52)	P1_A_EN
Test_out(53)	P1_done
Test_out(69:54)	Pass1_Out(15: 0)
Test_out(70)	P2_A_EN
Test_out(71)	P2_done

Test_out(87:72)	Pass2_Out(15: 0)
Test_out(88)	P3_A_EN
Test_out(89)	P3_done
Test_out(105:90)	Pass3_Out(15: 0)

∗ JPEG2000 Chip Pin Map Table

Table A- 7 JP2K Top Pin Map Table

Test_out port	**Pin Map**
For JPEG2000 in Normal Mode	
Test_in(14:2)	Img_size_Y
Test_in(27:15)	Img_size_X
Test_in(29:28)	Tile_size_YX
Test_in(45:30)	Dwt_in_Ext
Test_in(61:46)	data_in_TILE_B
Test_out(52)	CEN_Ext
Test_out(53)	WEN_Ext
Test_out(66:54)	address_in_Ext
Test_out(67)	CEN_TILE_A
Test_out(68)	WEN_TILE_A
Test_out(74:69)	address_in_TILE_A
Test_out(90:75)	data_in_TILE_A
Test_out(91)	CEN_TILE_B
Test_out(92)	WEN_TILE_B
Test_out(98:93)	address_in_TILE_B
Test_out(111:99)	Address
Test_out(143:112)	OutBus

Test_out(144)	RAM_EN
Test_out(145)	RAM_WEN