國立交通大學

資訊科學與工程研究所

碩士論文

支援 MPEG 可重組視訊編碼運作模式之系統單晶片架構設計

SoC architecture for MPEG Reconfigurable Video Coding Framework

研 究 生：蕭哲民

指導教授：蔡淳仁 教授

中華民國九十六年六月

支援 MPEG 可重組視訊編碼運作模式之系統單晶片架構設計

SoC architecture for MPEG Reconfigurable Video coding Framework

研 究 生 : 蕭哲民　　Student : Jer-Min Hsiao

指導教授 : 蔡淳仁 博士　　Advisor : Chun-Jen Tsai

國立交通大學

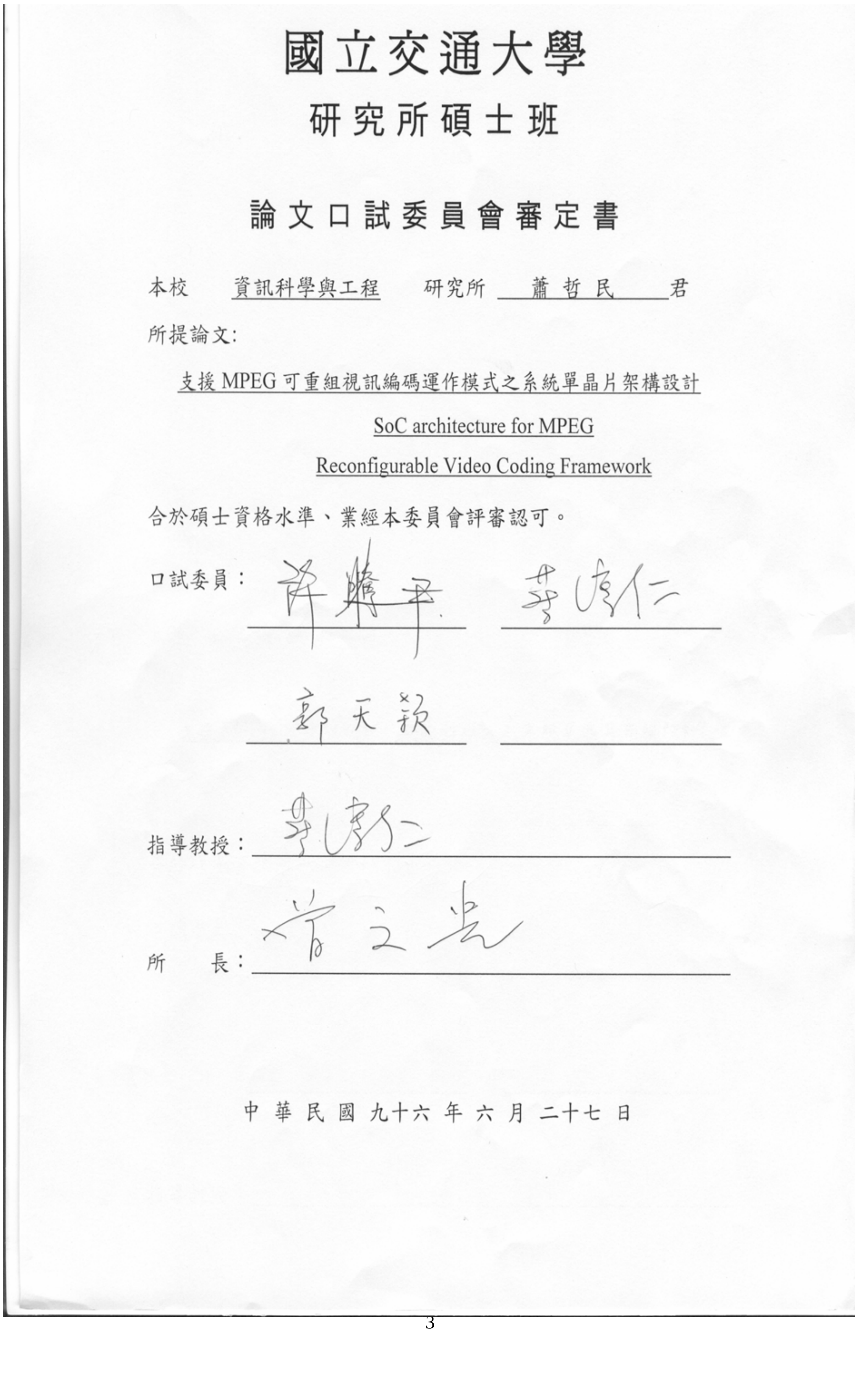
資訊科學與工程研究所

碩士論文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of Master
in

Computer Science
June 2007
Hsinchu, Taiwan, Republic of China

中華民國九十六年六月



國立交通大學
研究所碩士班

論文口試委員會審定書

本校 資訊科學與工程 研究所 蕭 哲 民 君

所提論文:

支援 MPEG 可重組視訊編碼運作模式之系統單晶片架構設計

SoC architecture for MPEG

Reconfigurable Video Coding Framework

合於碩士資格水準、業經本委員會評審認可。

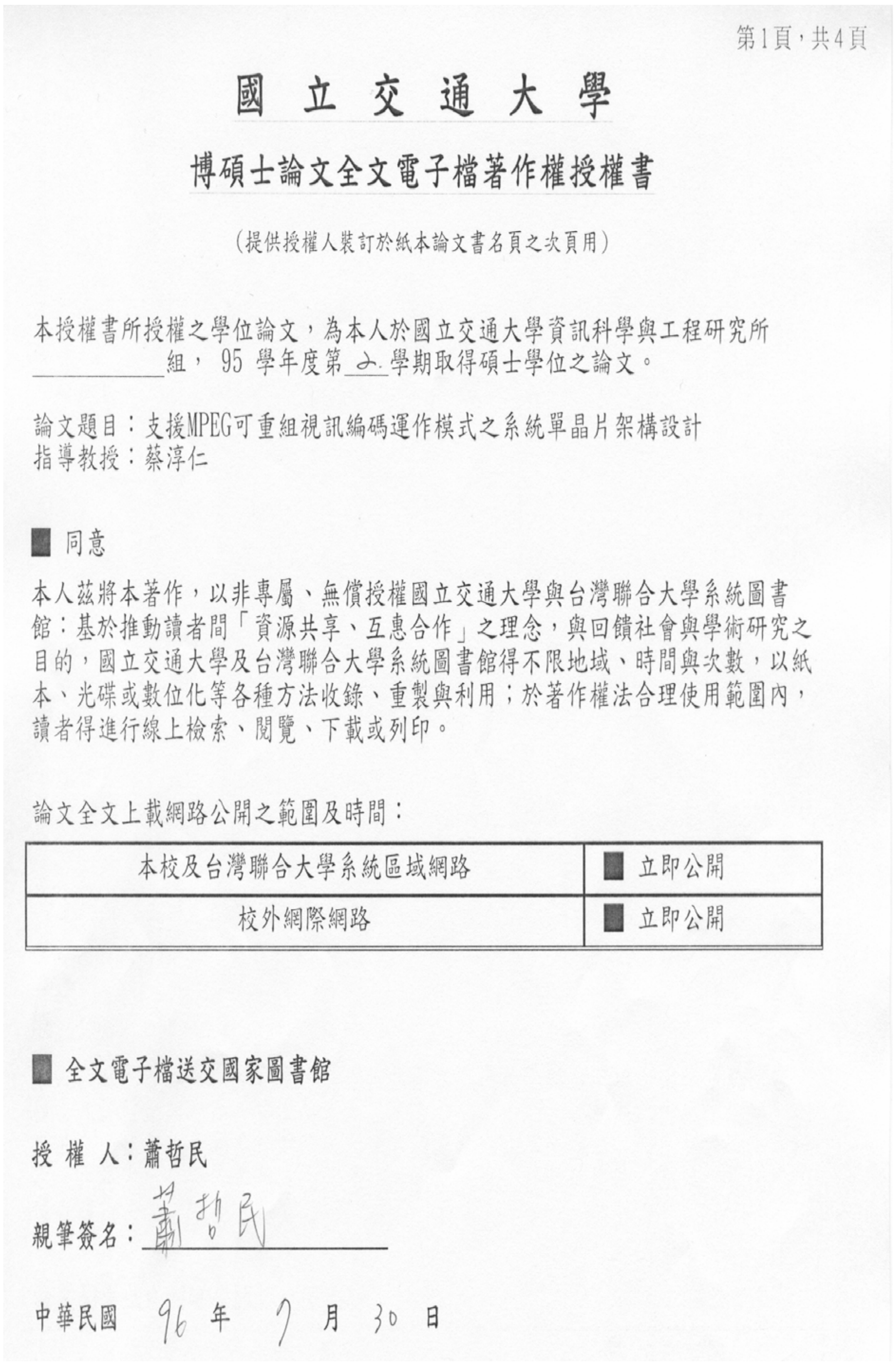
口試委員： [illegible] [illegible]

郭天穎

指導教授： [illegible]

所　　長： [illegible]

中 華 民 國 九十六 年 六 月 二十七 日



第1頁，共4頁

國 立 交 通 大 學

博碩士論文全文電子檔著作權授權書

（提供授權人裝訂於紙本論文書名頁之次頁用）

本授權書所授權之學位論文，為本人於國立交通大學資訊科學與工程研究所__________組， 95 學年度第 2 學期取得碩士學位之論文。

論文題目：支援MPEG可重組視訊編碼運作模式之系統單晶片架構設計
指導教授：蔡淳仁

■ 同意

本人茲將本著作，以非專屬、無償授權國立交通大學與台灣聯合大學系統圖書館：基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學及台灣聯合大學系統圖書館得不限地域、時間與次數，以紙本、光碟或數位化等各種方法收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行線上檢索、閱覽、下載或列印。

論文全文上載網路公開之範圍及時間：

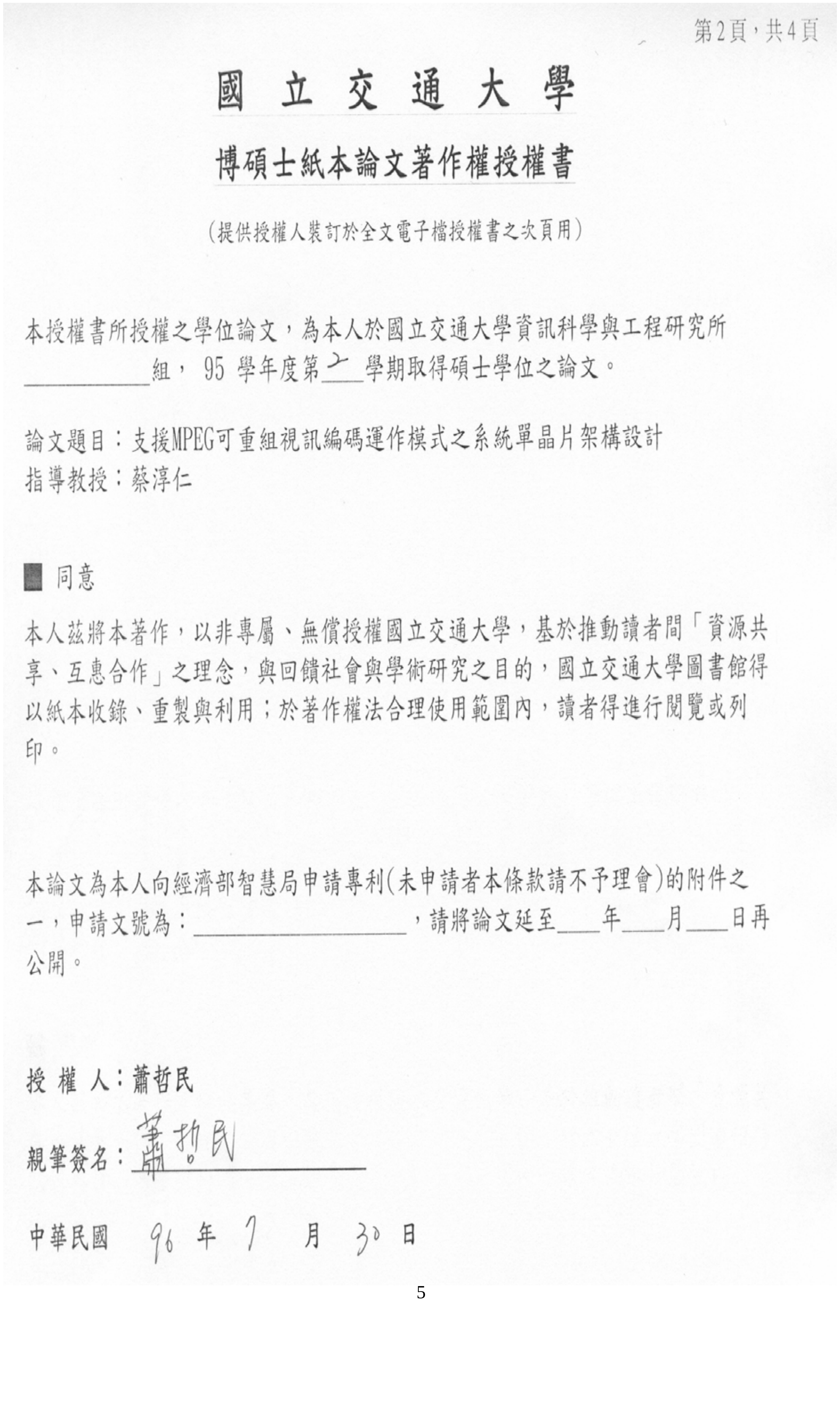
本校及台灣聯合大學系統區域網路	■ 立即公開
校外網際網路	■ 立即公開

■ **全文電子檔送交國家圖書館**

授 權 人：蕭哲民

親筆簽名： 蕭哲民

中華民國 96 年 7 月 30 日



國 立 交 通 大 學

博碩士紙本論文著作權授權書

(提供授權人裝訂於全文電子檔授權書之次頁用)

本授權書所授權之學位論文，為本人於國立交通大學資訊科學與工程研究所__________組， 95 學年度第 2 學期取得碩士學位之論文。

論文題目：支援MPEG可重組視訊編碼運作模式之系統單晶片架構設計
指導教授：蔡淳仁

■ 同意

本人茲將本著作，以非專屬、無償授權國立交通大學，基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立交通大學圖書館得以紙本收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行閱覽或列印。

本論文為本人向經濟部智慧局申請專利(未申請者本條款請不予理會)的附件之一，申請文號為：____________________，請將論文延至____年____月____日再公開。

授 權 人：蕭哲民

親筆簽名：蕭哲民

中華民國 96 年 7 月 30 日

國家圖書館
博碩士論文電子檔案上網授權書

（提供授權人裝訂於紙本論文本校授權書之後）

ID:GT009455585

本授權書所授權之論文為授權人在國立交通大學資訊科學與工程研究所 95 學年度第 二 學期取得碩士學位之論文。

論文題目：支援MPEG可重組視訊編碼運作模式之系統單晶片架構設計
指導教授：蔡淳仁

茲同意將授權人擁有著作權之上列論文全文（含摘要），非專屬、無償授權國家圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

※ 讀者基於非營利性質之線上檢索、閱覽、下載或列印上列論文，應依著作權法相關規定辦理。

授權人：蕭哲民

親筆簽名：蕭哲民

民國 96 年 7 月 30 日

SoC architecture for MPEG Reconfigurable Video Coding Framework

Abstract

Due to the variety of popular video coding standards, many efforts have been put into the design of a single video decoder chip that supports multiple formats. In 2004, ISO/IEC MPEG started a new work item to facilitate multi-format video codec design and to enable more flexible usage of coding tools. The work item has turned into the MPEG Reconfigurable Video Coding (RVC) framework. The key concept of the RVC framework is to allow flexible reconfiguration of coding tools to create different codec solutions on-the-fly. In this thesis, flexible SoC architecture is proposed to support the RVC framework. Some analysis has been conducted to show the extra costs required for this platform compared to hard-wired codec architecture. In conclusion, the RVC framework can be mapped to an SoC platform to provide flexibility and scalability for dynamic application environment with reasonable cost in hardware design.

Acknowledge

在研究所兩年學習的時間裡,很感謝我的指導教授蔡淳仁博士,在老師的身上我學習到作研究英有的嚴謹態度,以及對各種小細節的注意,很感謝老師的耐心指導,讓我不管是在學術上,生活上都學習到作任何事情都應該全力以赴認真去執行。也謝謝同實驗室的厚任,雅婷,大家一起努力寫論文的革命感情。實驗室的學長和學弟妹們大家平時一起吃飯玩樂,讓我們研究生活不至於過於苦悶,最後我想謝謝我的家人,因爲有家人的支持以及鼓勵,我才有辦法無憂地讀書,感謝內湖教會朋友的鼓勵,以及梅竹團契的學弟妹的加油打氣,當然更要感謝上帝讓我這兩年來都很順利,最後很高興成爲 MMESLAB 的一員,真的很謝謝大家。

Content

Lists of Figures

E S A 1896

Lists of Tables

1. Introduction

Most multimedia devices today have to support multiple codec standards. Take video codecs for example, a portable multimedia player usually supports the playback of the MPEG-1/2, MPEG-4 SP, WMV, and H.264/MPEG-4 Part 10 video contents. In order to reduce system cost, a single-chip SoC solution that supports all these standards is a sensible approach. From IC designers' point of view this is not a serious problem since most (if not all) popular video codecs share the same block-based motion compensated transform coding data flow. In addition, many coding tools have similar architecture. However, there are some application issues that makes traditional codec design approaches unsatisfactory [1].

A major problem with existing approach of defining a codec standard is the lack of flexibility when new applications emerge. A video codec is composed of several coding tools (e.g. DCT/IDCT, MC, VLC/VLD, etc.). However, for a codec standard, the conformance point is defined at codec-level, instead of tool-level. Different profiles/levels are created for each codec to address the need of different classes of applications. This approach works fine in the past since the application scenarios were quite simple (e.g. DVD, DTV). However, with the exponential growth of new multimedia applications, the old approach of defining conformance point at codec-level becomes awkward. Quite often, a new application designer finds it impossible to find a reasonable codec profile@level to fit the target application well. For example, the FMO tool of H.264 is useless for many applications but a decoder may still need to support it simply because it is included in AVC baseline profile. In general, application environment is changing faster than an international standard can catch up that there should be a more efficient way of allowing a codec to adapt to new applications while maintaining interoperability among different solutions.

MPEG has recognized this issue and started a new work item called Video Coding Tools Repository (VCTR) in 2004. After some investigations, the direction and benefit of VCTR is becoming clear [2]. Later, this effort becomes the Reconfigurable Video Coding (RVC) framework in 2006 [3]. This new framework defines the conformance point at tool-level. Therefore, in principle, an RVC-enabled codec can negotiate on-the-fly with the video bitstream encoder/sender about which coding tools is required and how the data path can be wired among these coding tools in order to decode the video bitstream. After the setup stage, the decoder can decode the bitstream correctly. With this approach, an SoC can support multiple codec standards as well as creating customized codecs in real time as long as it contains all the standard-conforming tools that is necessary to decode bitstreams from different encoders.

1.1 Introduction to MPEG RVC Framework

The concept of MPEG RVC framework can be illustrated by Fig. 1. The key difference between RVC and the old MPEG codec standards is that the interface of each coding tools is defined precisely so that they can be used (like LEGO blocks) to build various codecs. The decoder configuration describes how input bitstream can be parsed so that the raw input data to each coding tools can be extracted. A decoder description language is under development so that the configuration of a specific codec (such as H.264) can be described using a (small) configuration bitstream. The decoder configuration bitstream will be processed by an RVC decoder before decoding of a video bitstream conforming to the described standard. Note that after processing a configuration bitstream, the RVC decoder will generate a Global Control Unit (GCU) that governs the operation of the coding tools.

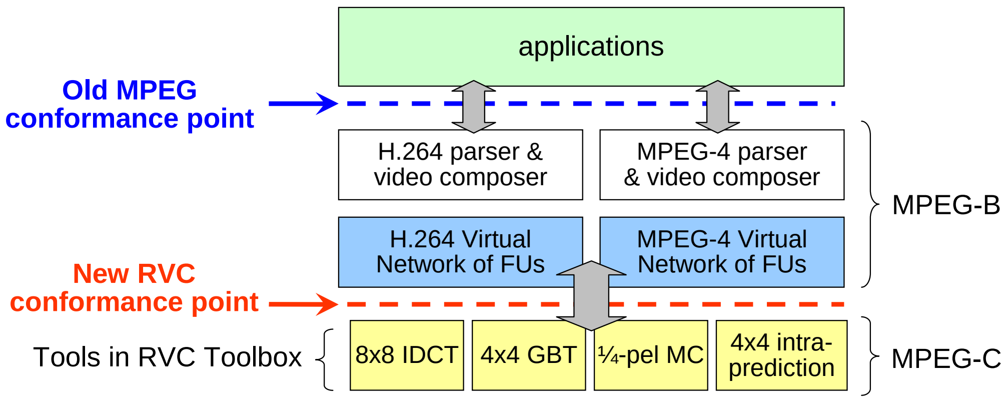

applications
Old MPEG conformance point
H.264 parser & video composer
MPEG-4 parser & video composer
MPEG-B
H.264 Virtual Network of FUs
MPEG-4 Virtual Network of FUs
New RVC conformance point
Tools in RVC Toolbox
8x8 IDCT
4x4 GBT
¼-pel MC
4x4 intra-prediction
MPEG-C

Fig. 1. Concept of MPEG RVC framework

In principle, the configuration description tells the RVC decoder how to wire the coding tools to form a data path. In the RVC framework, each coding tools is called a functional unit (FU) and is specified in Fig. 2 [1]. In Fig. 2, a control signal is a signal embedded in the video bitstream (for example, the width and height of the video frame). A context signal is a signal generated from the processing of bitstream data (for example, the AC prediction direction in the MPEG-4 Part 2 video standard). The context-control unit reads in the context and control signals generated by previous FU's and generates (or passes on) some context and control signals to the next FU's based on the result of the processing unit.

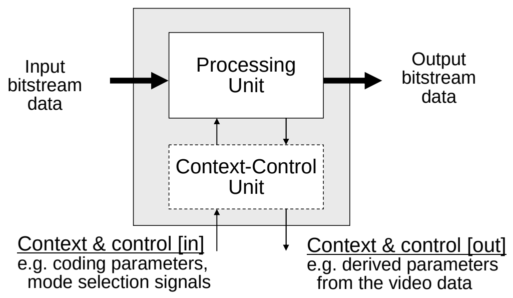


Fig. 2. Definition of a functional unit in RVC

The overall architecture of RVC is shown in Fig. 3. There is a syntax paring FU to parse the bitstream to context and control information and MB-based data. The content and control information is fed to Global Controller Unit(GCU) and data is fed to other MB-based FU to processing. The GCU is responsible for controlling the data path between network of functional units and receiving and passing content and control information to each function units.

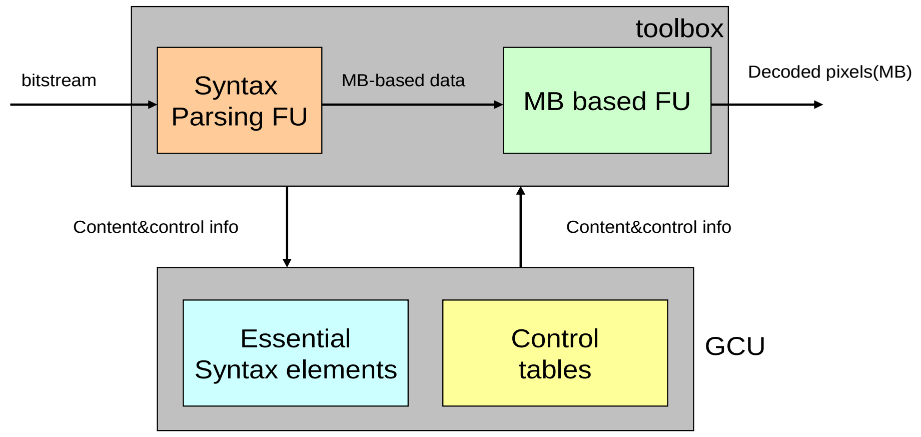


Fig. 3. Overall architecture of RVC

A partial example of a configured RVC codec that behaves like an H.264 baseline decoder is shown in Fig. 4. In Fig. 4, the functional blocks encircled in the dashed rectangles will be implemented using the proposed architecture in next two sections.

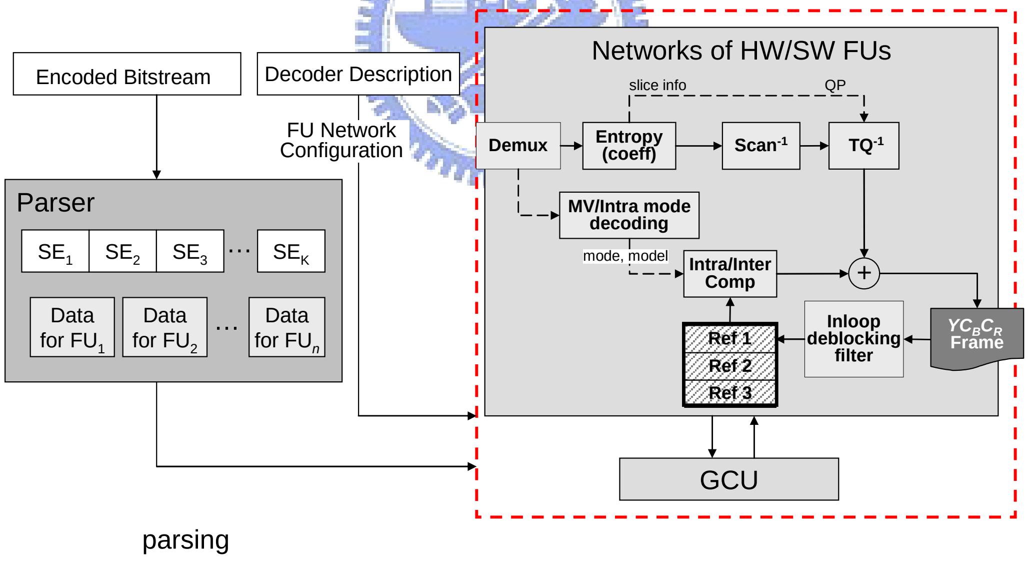


Fig. 4. Example of RVC configuration

Currently, the two parts of RVC is defined in two different MPEG internal documents. Namely,

- MPEG-B part 4, Codec Configuration Representation.
- MPEG-C part 4, Video Tool Library.

The MPEG-B is composed of two elements, the first element is the generic video bitstream description language that can be used to instantiate a parser, and the second element is the language that can be used to specify the network of functional units of the decoder data path. The tools in RVC toolbox is defined in MPEG-C. So far, the RVC framework is still under development in MPEG. Most of the investigations are done using C models and behavioral model simulators such as Moses [4]. In this thesis, we propose a HW platform that can support the RVC framework for construction of a virtual network of FUs (MPEG-B) from a collection of tools (MPEG-C).

The thesis is organized as follows. Some related works are introduced in chapter 2. And the proposed SoC architecture for supporting the RVC framework is presented in chapter 3. Some comparisons of the RVC architecture with a common hard-wired solution are also given in this chapter. The designs of some HW functional units are shown in chapter 4. The experimental results and analysis are discussed in chapter 5. Finally, the conclusions and discussions are given in chapter 6.

2. Previous Works

Nowadays, multimedia applications for embedded systems are growing quickly. Media players or cell phones have to support multi-standard video/audio codecs to fulfill consumer’s need. There are many papers on the design of video codecs, multi-standard codecs, or reconfigurable codecs. We will discuss some of these designs in this chapter.

2.1 Traditional ASIC design approach

In traditional video ASIC design, profiling of computation load and memory requirements of the coding tools is first performed [11][12]. And the coding tool that is suitable for hardwired logic is implemented as an IP. Take H.264 for example, many papers presents the implementation of integer transform/quantizer[6][13], motion estimation and motion compensation[14][15], deblocking filter[16], and entropy coding[17][18].

Lin et al. [5] proposed a fully pipelined architecture of an H.264 video decoder. The architecture is shown in Fig. 5. Two different pipeline processing schemes are used, including 4x4 block level pipeline processing and hybrid block level pipeline processing. Since the size of processing block unit for I/T is 4x4, a 4x4 pipeline flow is used and for other modules, MB size pipeline is adopted.

For traditional ASIC video codec design, they are pursuing designs with smaller area, less internal memory, and higher clock rate.

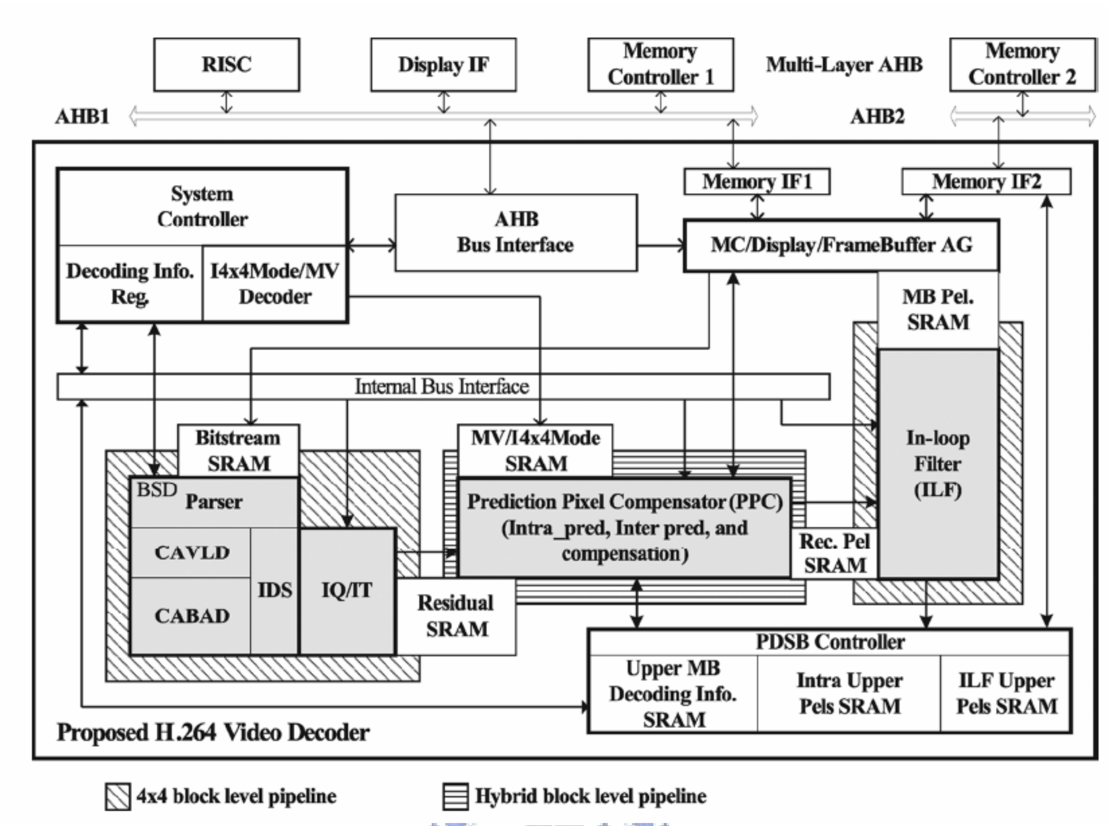


Fig. 5. Architecture of the h.264 decoder in [5].

2.2 SoC architecture for multimedia system

It is not cost-effective to implement a complicated system completely in hardware. Therefore, hardware and software co-design is becoming popular for complex system design. Yang et al. [8] shows an example of prototyping a system on an SoC platform. The example is a JPEG codec. First, they profiled the JPEG encoder (Fig. 6) and found that VLC and DCT cost most computation time. Therefore, DCT and VLC are partitioned into hardware (FPGA) and other modules are in software. With HW/SW co-design, the performance of the system is better than a pure software system and more flexible than a pure hardware approach. The whole architecture is shown in Fig. 7. This paper shows how to prototype a system and how to partition functionality in hardware or software. Besides, it also illustrates how to design the hardware and software interface using a bus protocol (AHB) [19].

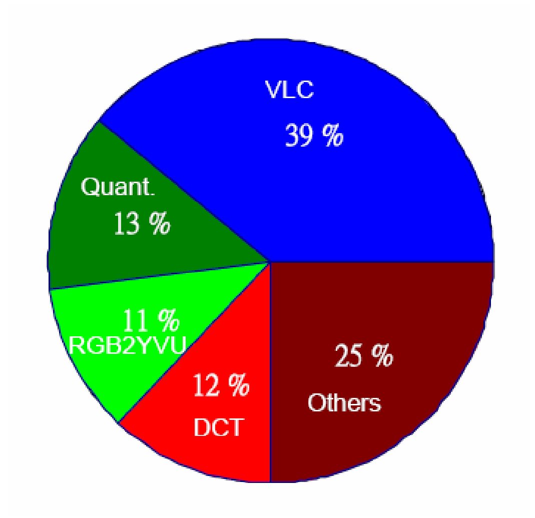


Fig. 6. Profiling of JPEG encoder

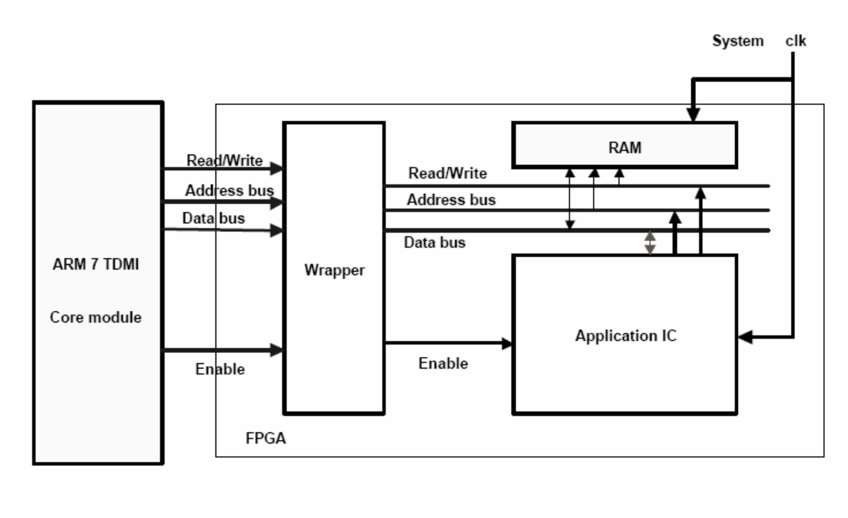


Fig. 7. Architecture for JPEG encoder

2.3 Reconfigurable video coding

Zhang and Kittler [9] propose a dynamically reconfigurable video (DRV) codec. The idea is quite simple; they add a extra DRV head on existing codec head. The DRV head configure the DRV codec to decode. And there are many coding tools in a DRV codec. The idea is similar with MPEG RVC. However, the network of functional units in MPEG RVC can be either in hardware or in software. In this paper, they only allow software coding tools. Besides, in many multi-standard hardware approaches codec is implemented by merging coding tools in different codecs since coding tools between different coding standard are similar [20].

2.3.1 RVC Proposal from Hanyang University

Lee and Kim [10] have proposed a draft of RVC framework in MPEG (Fig. 8). They define a decoder description which is composed of seven tables. The decoder is configurable by changing the tables. The detail is presented in this section.

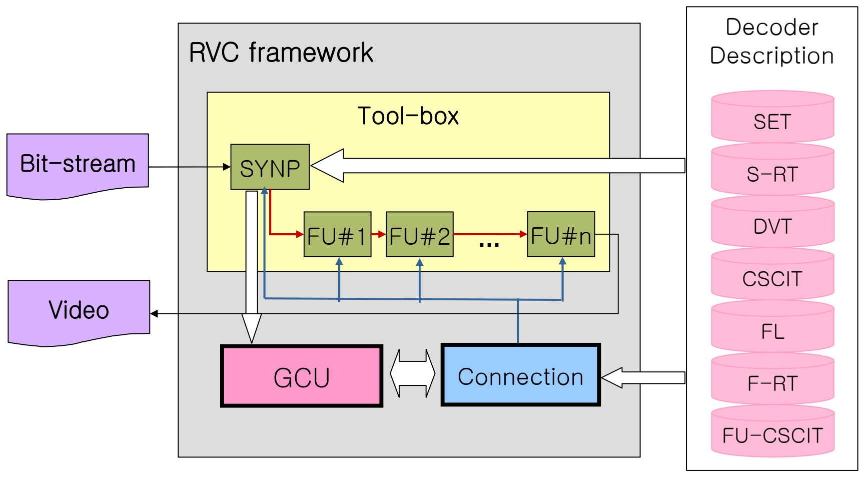


Fig. 8. Architecture of RVC proposal from Hanyang University [10]

2.3.1.1 Decoder description (DD)

The decoder description describes how to configure the decoder. In this proposal, the description is composed of seven tables, including CSCI, DVT, FL, F-RT, FU-CSCIT, SET, and S-RT (these acronyms are explained in Table 1). The proposal uses a DD parser to parse a decoder description bitstream and set up seven tables (Fig. 9). By using different tables, it is able to configure the decoder to support different specs such as MPEG-4, MPEG-2, H.264…etc.

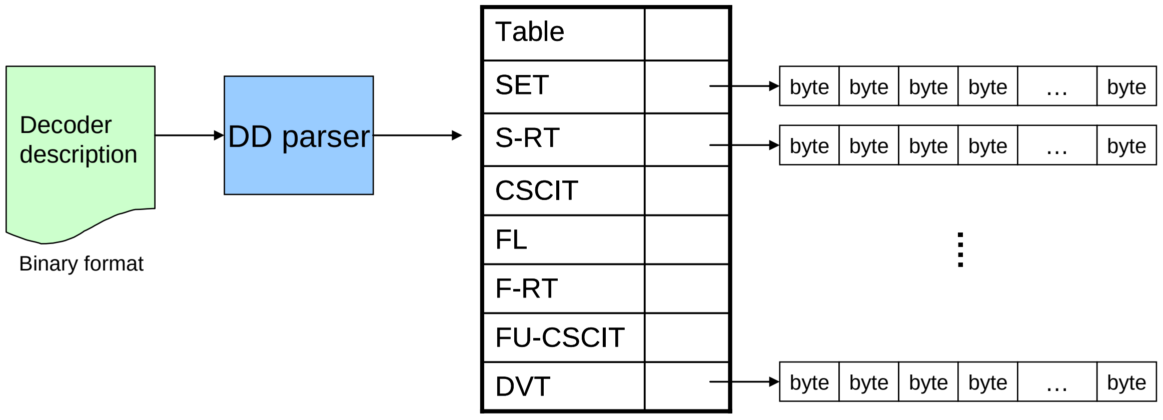

Decoder description
Binary format
DD parser
Table
SET
S-RT
CSCIT
FL
F-RT
FU-CSCIT
DVT
byte byte byte byte … byte
byte byte byte byte … byte
byte byte byte byte … byte

Fig. 9. Mechanism of parsing decode description

CSCIT	CSCI Table
DVT	Default Value Table
FL	FU List
F-RT	FU Rule Table
FU-CSCIT	FU CSCI Table
SET	Syntax Element Table
S-RT	Syntax Rule Table

Table 1. seven decoder description table

2.3.1.2 Mechanism of decoding the bitstream

The decoding procedure is as follows. First, we start at R0 state in F-RT. According to the rules in F-RT , we do syntax-parsing job if F0 is found in the table. Otherwise, we look up the FL table. The table records the corresponding CSCI buffer with each FU and execute that FU in F-RT. By doing so, we store the output of other-FU or syntax-parsing FU to the corresponding CSCI buffer. After that, we look up the F-RT again with the number of CSCI buffer as our index. This way, we are able to branch to next stare R# according to F-RT. If the state is not end state,we continue the whole procedure again (Fig. 10).

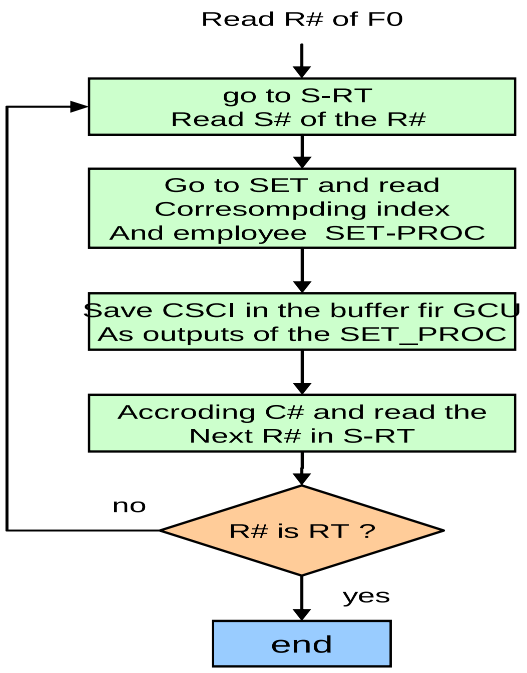


Fig. 10. Decoding procedure in [10]

The detail syntax parsing procedure is described as follows. First, we read the corresponding index of syntax-parsing FU. According to the index, we go to S-RT table and lookup another index for SET. Then, we go to SET, execute the specific SET-PROC by the index. The SET_PROC is kind of program and it retrieves the syntax element and store them into corresponding CSCI buffer. And by the number of CSCI buffer, we check S-RT again and branch to the next state in S-RT. If the next state is RT (end), we continue the whole procedure for syntax parsing. Otherwise, sytax parsing is completed (Fig. 11).

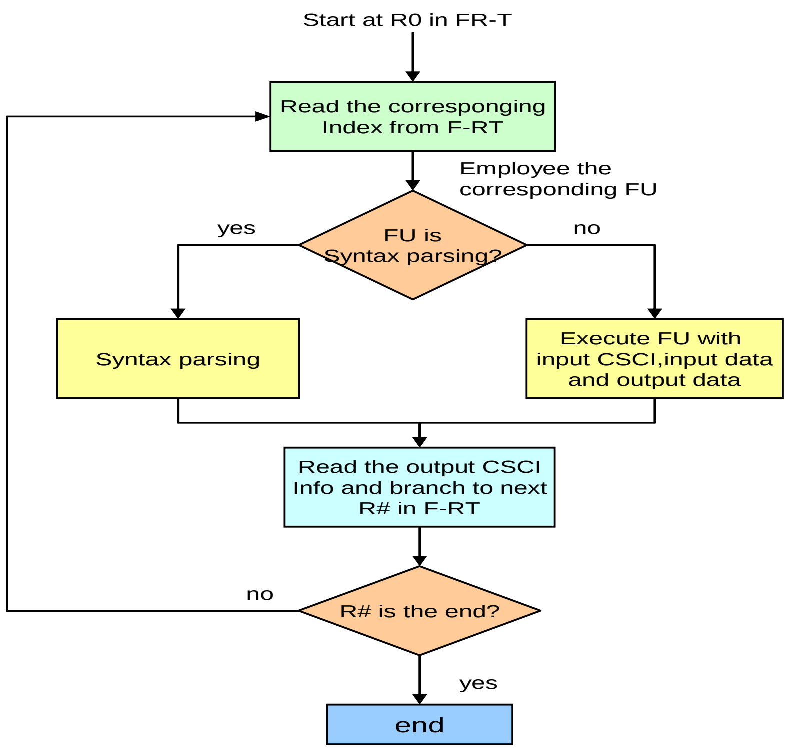

Start at R0 in FR-T
Read the corresponging Index from F-RT
Employee the corresponding FU
FU is Syntax parsing?
yes
no
Syntax parsing
Execute FU with input CSCI,input data and output data
Read the output CSCI Info and branch to next R# in F-RT
R# is the end?
no
yes
end

Fig. 11. Parsing procedure in [10]

3. Proposed RVC Framework

3.1 SoC architecture of the propose RVC framework

Since the specification of the bitstream syntax description language is still under development at MPEG, only the operation of composition of a network of functional units and the specification of functional units are defined at this time. This thesis proposes a VLSI architecture that supports the RVC framework. The RVC framework actually fits the platform-based design principle of SoC quite well. For maximal flexibility, the global control unit (GCU) will be implemented in software and running on the processor core of an SoC. Each coding tool can be implemented as an IP on the bus with limited configurability via a private register file. The proposed architecture is show in Fig. 12.

It is important to note that in the proposed architecture, hardware FUs and software FUs (stored in SDRAM) can be mixed to compose a network of functional units for a specific codec. We use multiple banks of SRAM as the “virtual wire” to connect each functional unit. With this virtual wire approach, the system can be scale up easily. New functional units (either hardware or software) can be integrated into the system to support new codec with very little design effort.

It is also important to note that the hardware coding tools are not attached to the main system bus (AMBA AHB) directly. A local bus, Multi-Media Bus (MMB), is used to off-load the bandwidth from the main system bus. In our implementation, MMB is a simplified version of AHB, however, a more flexible bus protocol or the concept of network-on-chip (NoC) can be used here to increase the throughput of the bus. A two-way DMA is used to transfer data between external SDRAM and internal

SRAM banks. The DMA can be invoked from either the ARM core or the coding tool IPs. The reason for multiple SRAM's on the MMB is to reduce the memory bandwidth requirement for parallel operations of the coding tools.

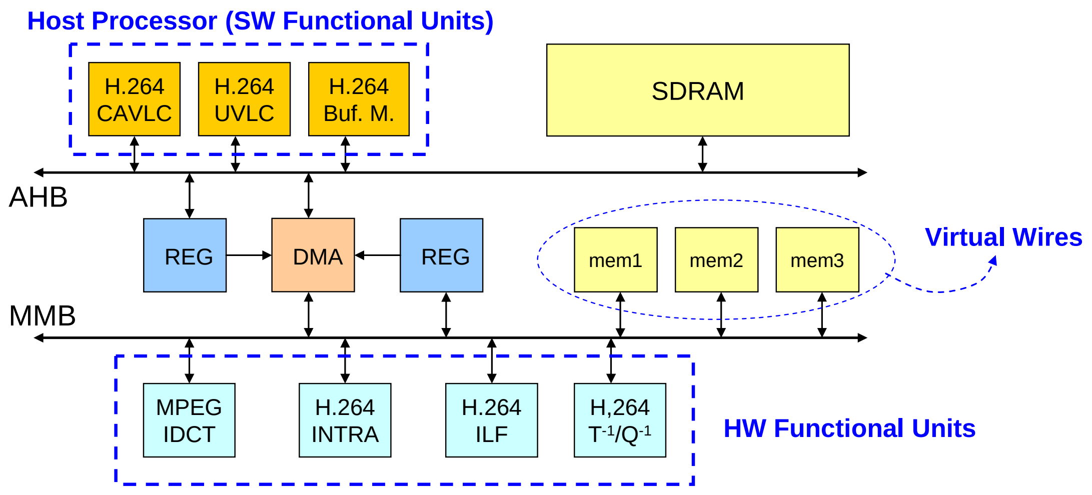


Fig. 12. Our proposed SoC architecture for MPEG RVC

Although local bus and multiple SRAM banks are used to alleviate the bandwidth issue, the performance of this architecture still cannot match that of a hard-wired architecture. For example, a hard-wired H.264 baseline decoder may have a tighter MB decoding pipeline as shown in Fig. 13. There are two main advantages of the architecture in Fig. 13. First of all, the decoding pipeline is controlled by a hard-wired FSM with cycle-based synchronization. On the other hand, for the RVC framework, the GCU controller will be implemented in software, and hence, cannot guarantee cycle-based operation of the pipeline. Another advantage of the hard-wired approach is that it does not require excessive accesses to external memory.

It is important to point out that the purpose of the RVC framework is not to obtain the most efficient design of a single codec, but to allow a flexible and extensible design of codec systems. Multi-standard codec support (or even generating customized codecs on-the-fly) can be achieved by configuring a new GCU via decoder description bitstreams. In the next section, we will study an actual implementation of the proposed architecture in Fig. 12 to get an idea about the cost one has to pay for such flexibility.

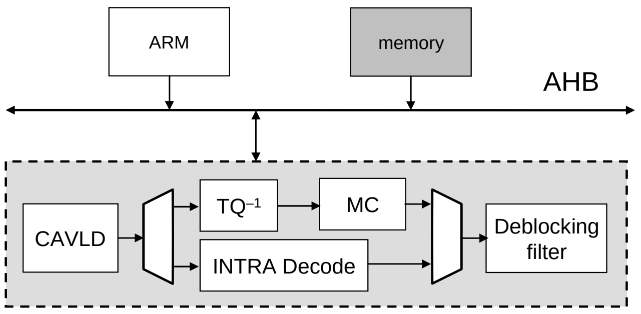


Fig. 13. Hard-wired decoder example

3.2 Propose RVC framwrok

In this section, an implementation of the proposed system architecture (Fig. 12) is investigated. The implementation is based on an SoC emulation platform, the ARM Integrator [21]. The platform is composed of a main board, an ARM 9 processor core module, and a Xilinx VirtexE XCV2000E FPGA logic module. The platform adopts the AMBA bus protocol. The RVC coding toolbox logic of the proposed system is implemented in the FPGA. The local bus protocol, MMB, of the toolbox logic is a reduced version of AHB with much less wires and a minimal implementation of bus arbiter and decoder. The detail architecture of emulation platform will be presented in chapter 5.

In the proposed system architecture, the controller that drives the operation of the network of FUs is implemented in software. As a result, the codec pipeline is not executed in a lock step fashion but instead driven by the software controller via signals triggered by read/write of register files. Each coding tool FU (please refer to

Fig. 2) is implemented so that the input bitstream data is coming from a SRAM bank on the MMB and the output bitstream data will be stored in another SRAM bank on the MMB. Block RAMs of the Virtex II FPGA and the ZBT SRAM of the ARM Integrator are used for this purpose. Note that in the proposed architecture, the input/output SRAM banks for a FU (either software or hardware) are dynamically controlled by a memory allocator module (see Fig. 14). It is obvious that such implementation is not as efficient as a tightly-coupled pipeline [21] where different pipeline stages are connected via registers or FIFO.

3.2.1 Design of Global Control Unit

3.2.1.1 architecture of GCU

The Global Control Unit of the MPEG RVC framework can be dynamically implemented as in Fig. 14. In Fig. 14, tool state table is a runtime table that record the states of each running FUs. The table is used by the controller to synchronize the operation of the codec pipelines. The network description table is extracted from the decoder description which is attached to the encoded video bitstream. It basically describes the input/output connections (i.e. SRAM banks) of the networks of the functional units. Note that this table can be modified by the memory allocator to allow optimal use of the available SRAM banks. An example of the tool state table and the network description table is shown in Fig. 15.

Also note that in Fig. 15, the “Pointer” field of the network description table stores a pointer to the entry point of software FU and the address of the control register for hardware FU. For hardware FUs, the implementation of the processing unit and context-control unit follows traditional hard-wired IP design methodology where the processing unit is implemented as a data path and the context-control unit is a hard-wired FSM with register files for memory-mapped I/O configuration and signaling.

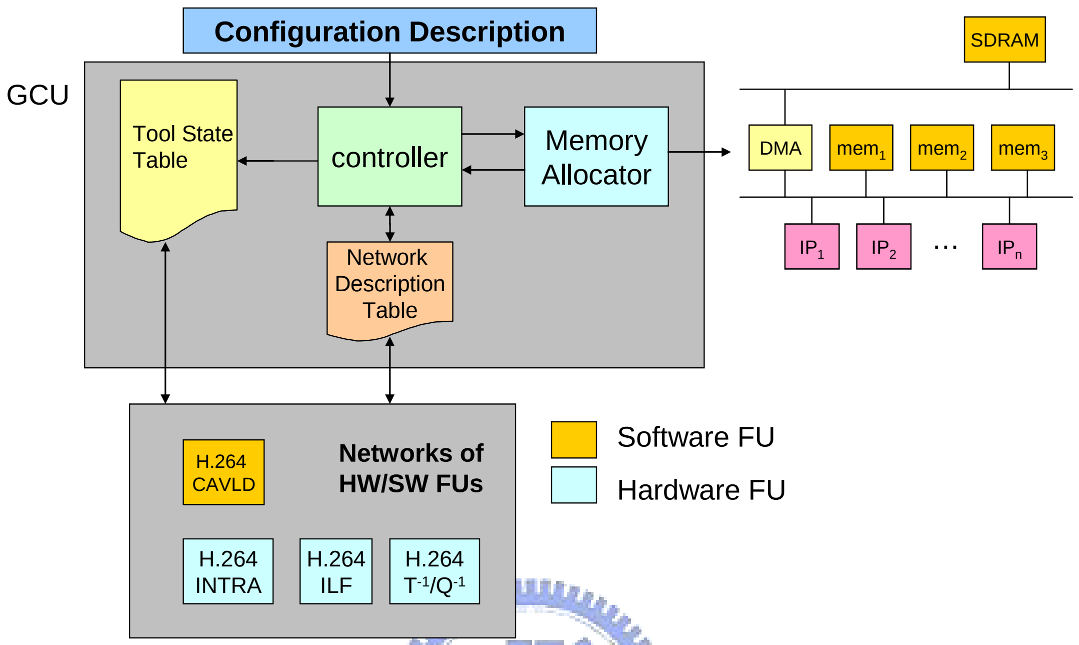
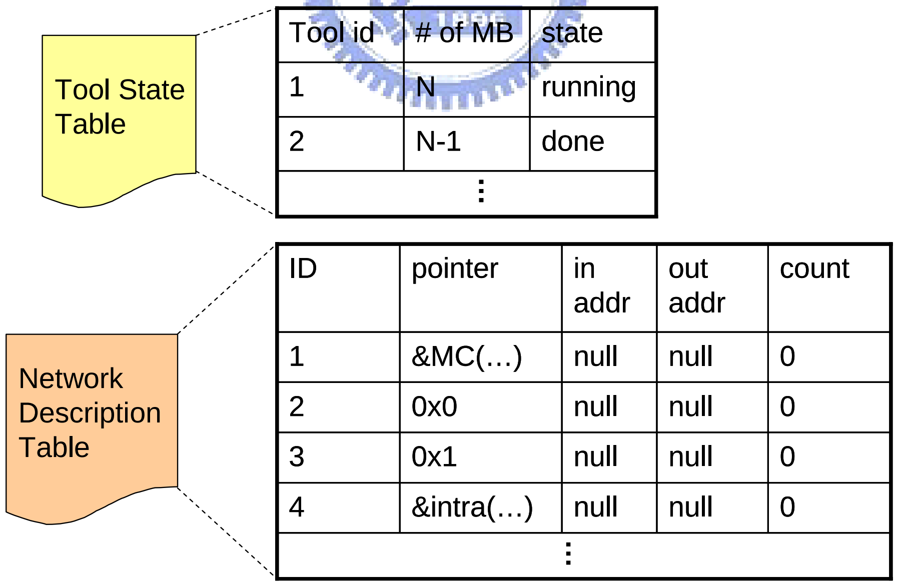


Fig. 14. GCU architecture



Tool State Table
Network Description Table

Tool id	# of MB	state
1	N	running
2	N-1	done
⋮		

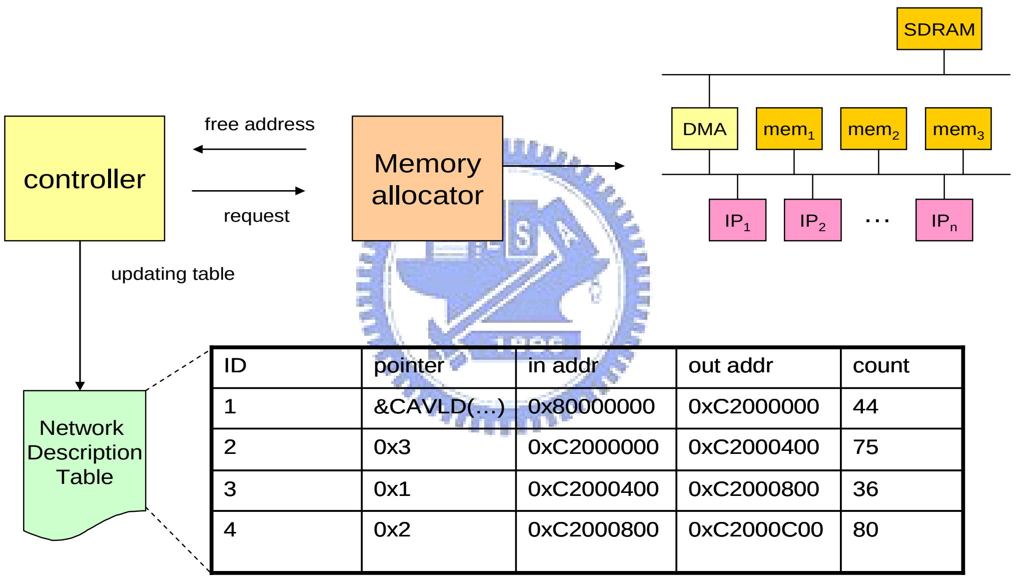
ID	pointer	in addr	out addr	count
1	&MC(…)	null	null	0
2	0x0	null	null	0
3	0x1	null	null	0
4	&intra(…)	null	null	0
⋮				

Fig. 15. example of tables in GCU

3.2.1.2 Mechanism of GCU

When the system is first initialized, the content of the network description table is empty. When parsing unit pass data and FU description into GCU, the controller invokes memory allocator to allocate a piece of free address for this FU and update the data in the tool box table.

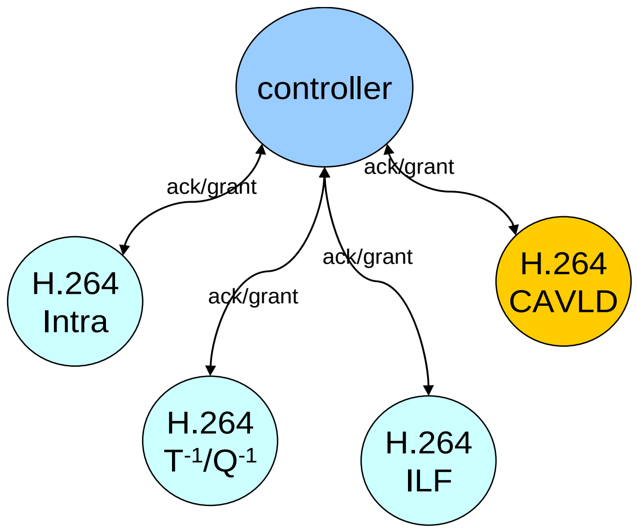
Until every FU described in the network description table is updated, the GCU can use this table to decide the data path easily according to the configuration description. It’s illustrated in Fig. 16.



ID	pointer	in addr	out addr	count
1	&CAVLD(…)	0x80000000	0xC2000000	44
2	0x3	0xC2000000	0xC2000400	75
3	0x1	0xC2000400	0xC2000800	36
4	0x2	0xC2000800	0xC2000C00	80

Fig. 16. Construction of virtual wires based on the NDT

When information in network description table is ready, we can invoke each FU to decode and control the whole decoding process. Controller use tool state table to schedule FU performing its job. Since the order of FU is listed sequentially in the tool state table, we can use handshaking protocol to schedule the whole decoding process. The scheduling method is illustrated in Fig. 17.



Tool id	# of MB	state
1*	N	running
2	N-1	done
3	N-2	done
4	N-3	done

* Mean the first FU

Fig. 17. Scheduling mechanism of GCU

3.2.2 Memory allocator

In our emulation platform, the arm integrator, there is a 1MB SRAM connected to the custom logic unit via soft-IP memory controller. However, since we want to use multiple small memory banks as the virtual wires to connect network of functional units. Therefore, we use block rams in system FPGA to construct our scratch-pad memory and size of each memory bank is 20KB. Moreover, we modify the bus decoder to change the system memory map to replace original SSRAM to our scratch-pad memory. It's shown in Fig. 18. And more details in our emulation platform will be presented in chapter 5.

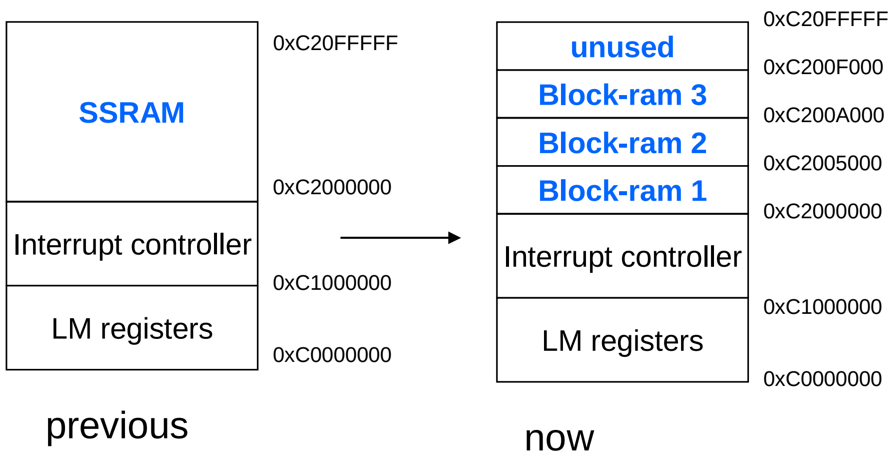


Fig. 18. New system memory map

How we manage scratch-pad memory is illustrated in Fig. 19. In our implementation, we use three memory blocks. Each memory block is further divided into small pages of 1KB each. Each memory block has an array of flags to tag the availability of each page within the memory block. We tag the array of corresponding index with bit ‘1’ if the page is used and vise versa. When the system controller request memory allocator to allocate one section of memory to him, the memory allocator search the array linearly to find out which pages are available and report it to the system controller until each network of functional units are assigned input and output data addresses.

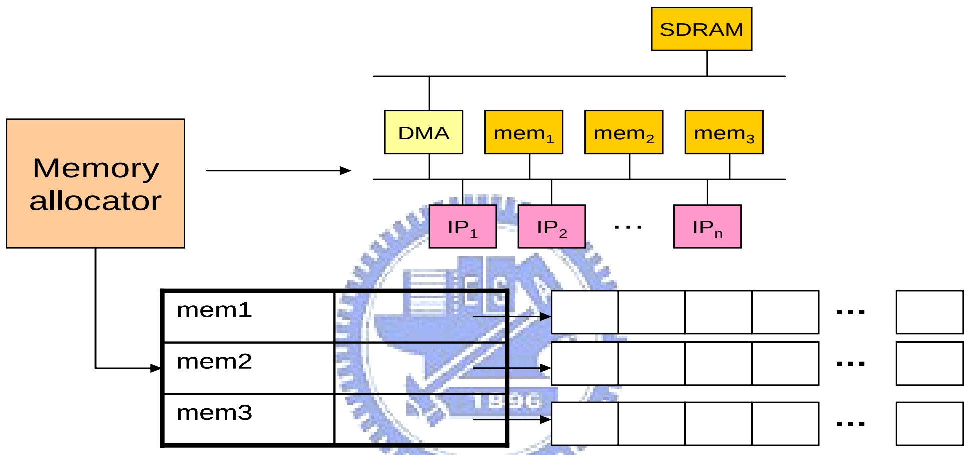


Fig. 19. Management of the on-chip scratch-pad memory

The proposed platform adopts round-robin policy to allocate memory banks. It’s shown in Fig. 20. For example, for the first operation the memory allocator picks mem1 and for the next operation the memory allocator picks mem2, and so on. The round-robin method is easy to implement and it also helps us to reduce the memory bandwidth for parallel operations of the coding tools.

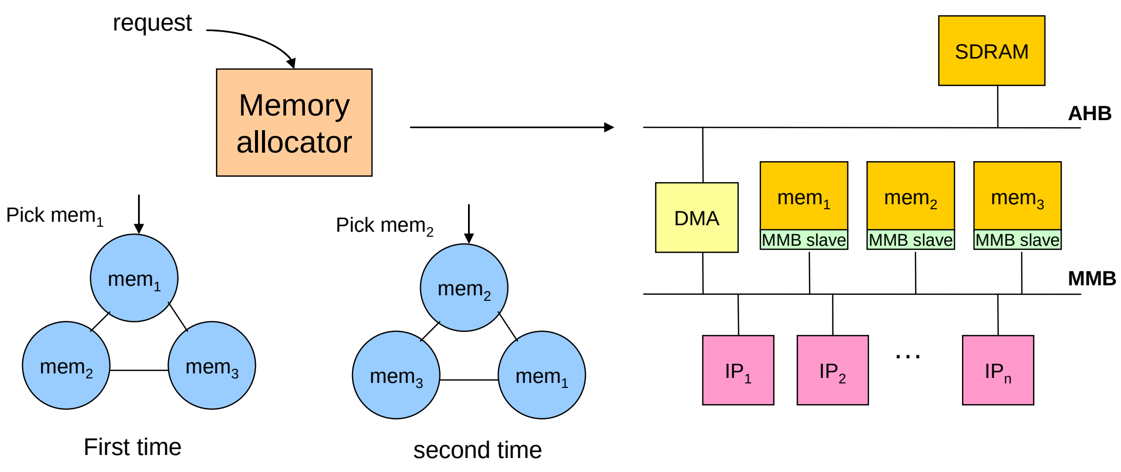


Fig. 20. Mechanism of how memory allocator choose memory banks

3.3 Implementation of FUs in the propose RVC framework

In software functional unit, we have to extract every function in traditional codec and modify it. We use function pointers to invoke software FUs. Therefore, we have to let the parameters of each function to be the same and add with the input/output data address within parameters. As for hardware FU, we have developed some H.264 and MPEG FUs and it is discussed in the next chapter.

4. Design of hardware functional units

4.1 Introduction to H.264/AVC decoding

The ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving Picture Experts Group (MPEG) developed a new video coding standard, Advanced Video Coding (AVC), also known as MPEG4 part 10 or H.264 [24][25]. It provides better compression efficiency than previous standards such as MPEG2, MPEG4 part2, H.263…etc. The improvements comes from adopting some new methods in inter and intra prediction. For example, the motion compensation block sizes for luma samples are 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 or 4x4. Smaller blocks enable a more accurate motion model. The intra prediction is enhanced with supporting 13 spatial prediction modes for luma samples.

H.264 defines four Profiles, baseline profile, main profile, extended profile and high profile, each supporting a particular set of coding tools for specific application. The baseline profile supports intra and inter coding (I-slice and P-slice) and entropy coding with context-adaptive variable-length codes (CAVLC). The main profile supports for interlaced video and inter coding with B-slice, inter coding using weighted prediction and entropy coding using context-based arithmetic coding(CABAC). The extended Profile doesn’t support interlaced video and CABAC but adds modes to enable efficient switching between coded bitstreams (SP-slice and SI-slice) and improve error resilience. The high profile includes not only B-slice and interlaced video but also 8x8 integer transform, 422, 444 color spaces and Q matrix. Fig. 21 shows the relationship between the four profiles.

The overall decoding path of h.264 is illustrated in Fig. 22. And we will show hardware design of inverse integer transform, inverse quantizer, intra compensation of H.264/AVC in the next sections.

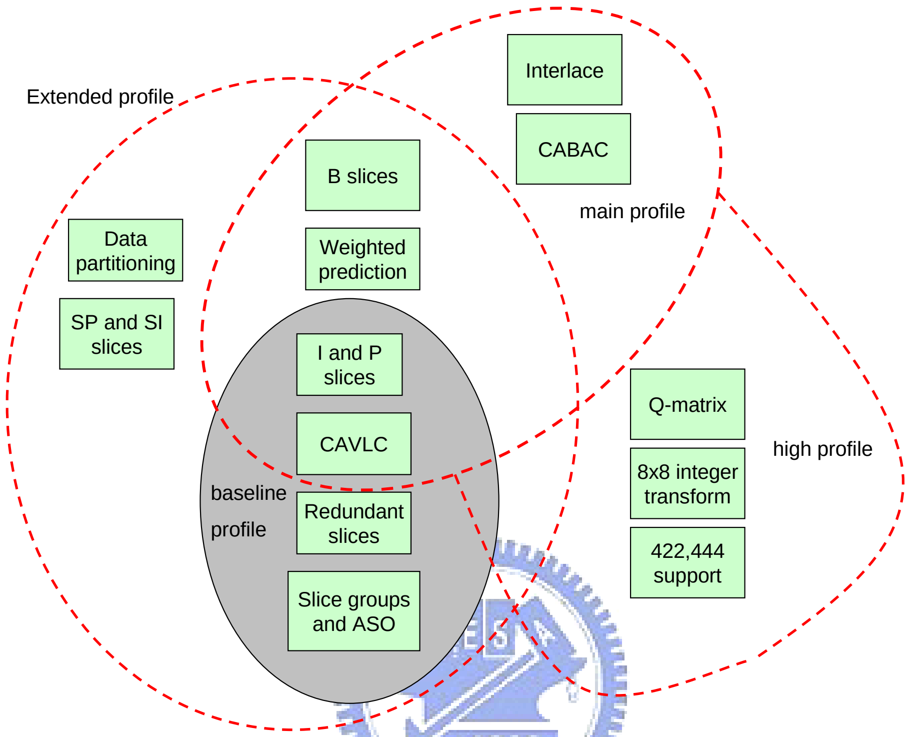


Fig. 21. H.264 profiles

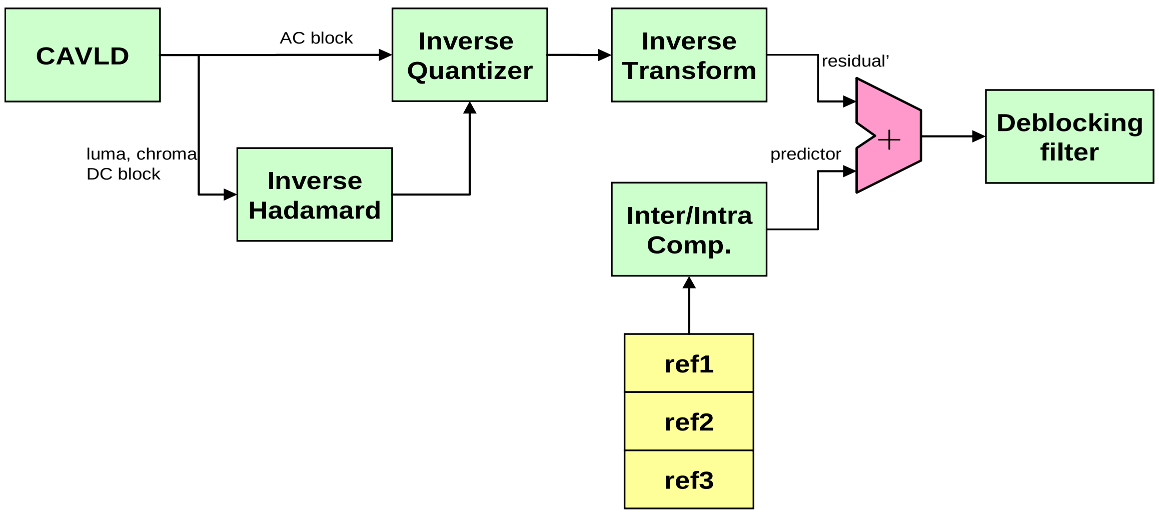


Fig. 22. Architecture of h.264 decoding

4.2 Design of inverse transform and inverse quantizer

In most video coding system, every residual macroblock is transformed, quantized and then entropy coded. The 8x8 DCT is the most commonly used. However, there are three transforms depending on the type of residual data in H.264 and every luma and chroma macro block is organized in 26 sub-blocks to be transmitted to corresponding transform [Fig2]. The order is the same with the labels of sub blocks. If a luma macro block is coeded in 16x16 intra mode, a sub block labeled -1 consists of DC coefficients of each 4x4 luma block is transmitted to a 4x4 hadamard transform first. Then, sub blocks 0-15 are transmitted to a 4x4 integer transform. Next sub block 16-17 are transmitted to a 2x2 hadamard transform. Finally, sub block 18-25 are also transmitted to a 4x4 DCT-based integer transform.

Consequently, A 4x4 hadamard transform is applied for DC coefficients of luma macroblock coded in 16x16 intra mode. A 2x2 hadamard transform is used for DC coefficients of chroma macroblock. A 4x4 integer transform is used for others. The decoding process is in the inverse order.

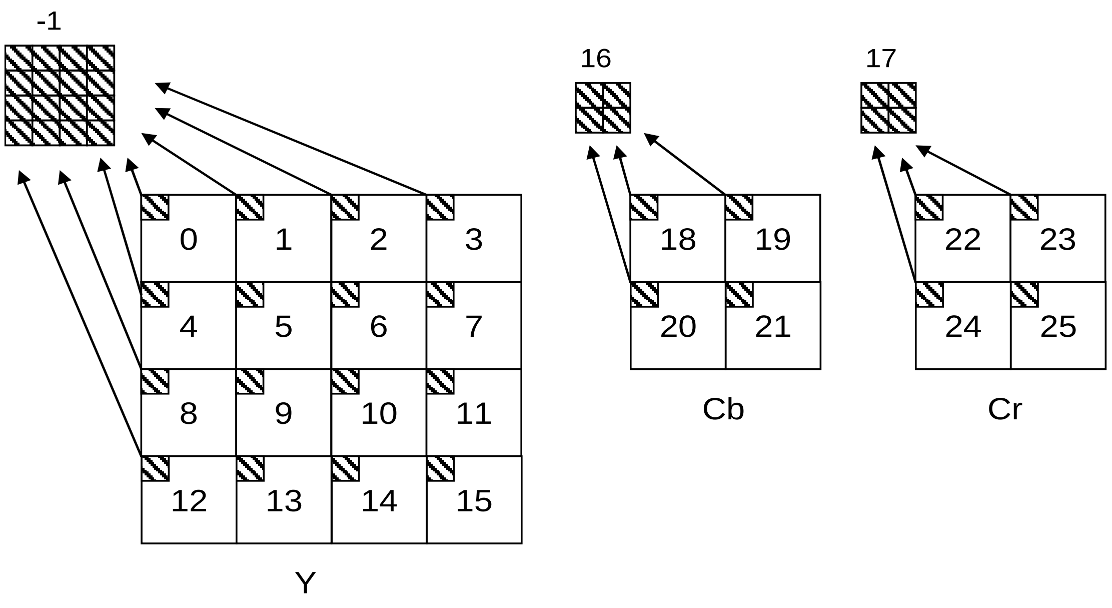


Fig. 23. input order of transmitting blocks

4.2.1 Architecture of inverse integer transform

The equation of 1-D inverse integer transform is illustrated in Eq.1. We can find that the coefficients of Eq.1 only contains {1, 1/2, -1/2, -1}. Therefore, we are able to use several adders and shifters instead of multipliers to implement it. Base on this idea, our 1-D inverse integer transform is shown in Fig. 24. And the architecture of 2-D inverse integer transform is illustrated in Fig. 25. We use one 1-D inverse integer transform and two block-rams in FPGA as transpose memory to construct the 2-D inverse integer transform. The Finite State Machine (FSM) is used to generate the corresponding addresses of transpose memory. The reason why we use two block ram in the design is that the bandwidth of block ram is 32 bits (16 bits dual port) since we have to feed 4 pixels into our 2-D inverse integer transform. Only one block ram is not sufficient to fulfill the bandwidth requirement.

The data is transmitted to the 1-D transform unit row-wise in the first pass, and the output is stored in the transposed memory column-wise (Fig. 26). Then, the logic fetches the data from the transpose buffer and feed them into the 1-D inverse transform unit again in the second pass. The output of the 1-D inverse transform unit is directly passed to the next functional unit in the decoding loop. The whole process costs 9 cycles and output 16 pixels of data. The timing diagram is showed in Fig. 27.

$$\begin{bmatrix} x_0' \\ x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \bullet \begin{bmatrix} y_0' \\ y_1' \\ y_2' \\ y_3' \end{bmatrix}$$

Eq. 1.Equation of 1-D inverse integer transform

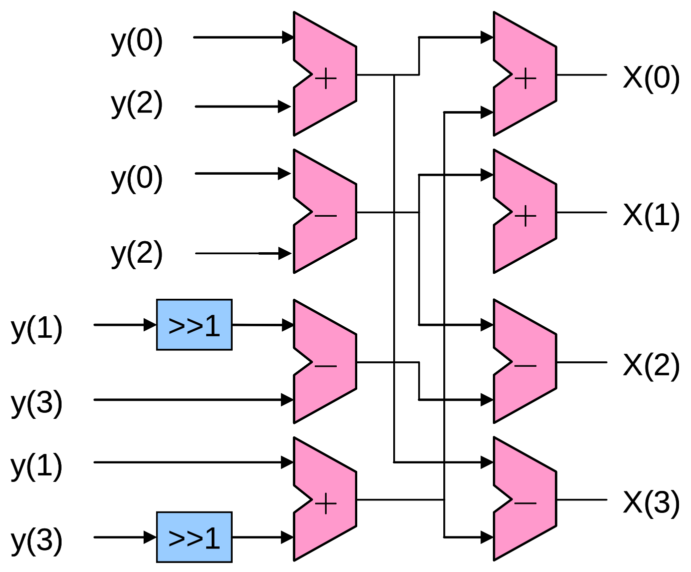


Fig. 24. Architecture of 1-D inverse transform

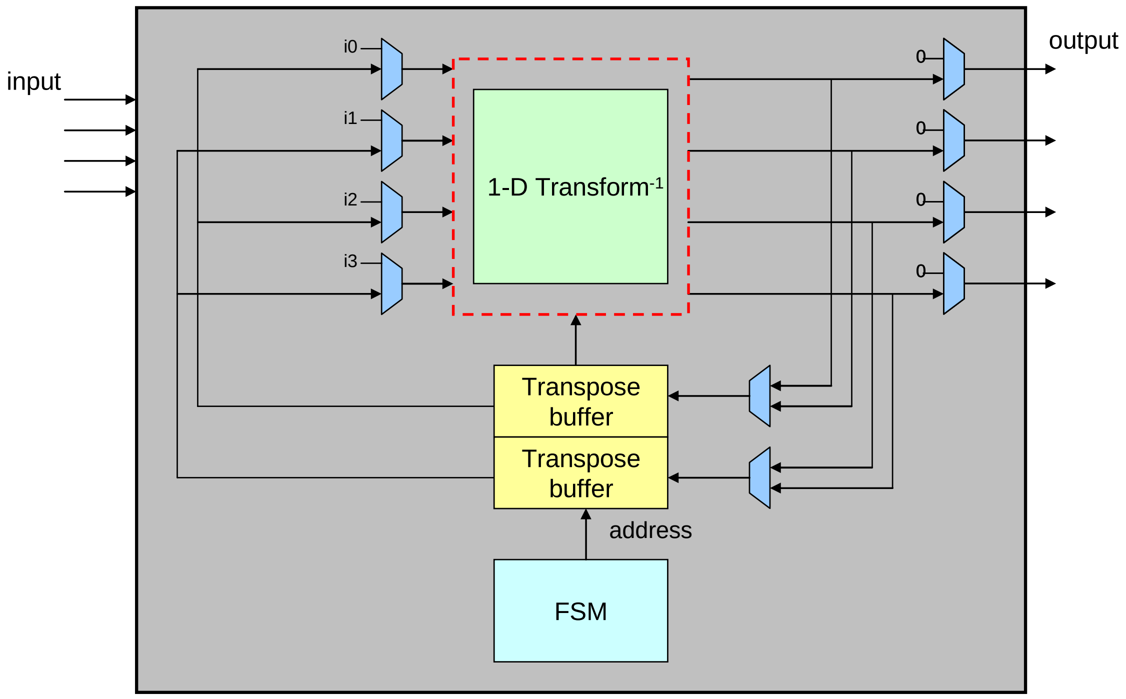


Fig. 25. Architecture of 2-D inverse transform

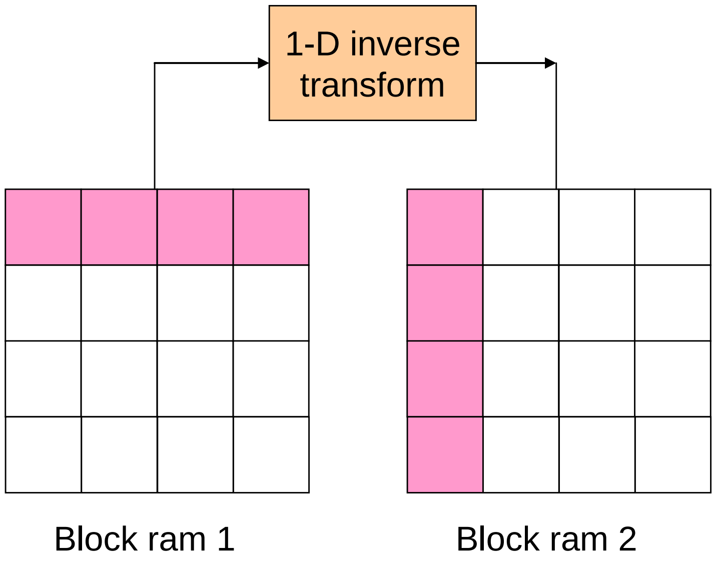


Fig. 26. Transposed memory of 2-D inverse transform

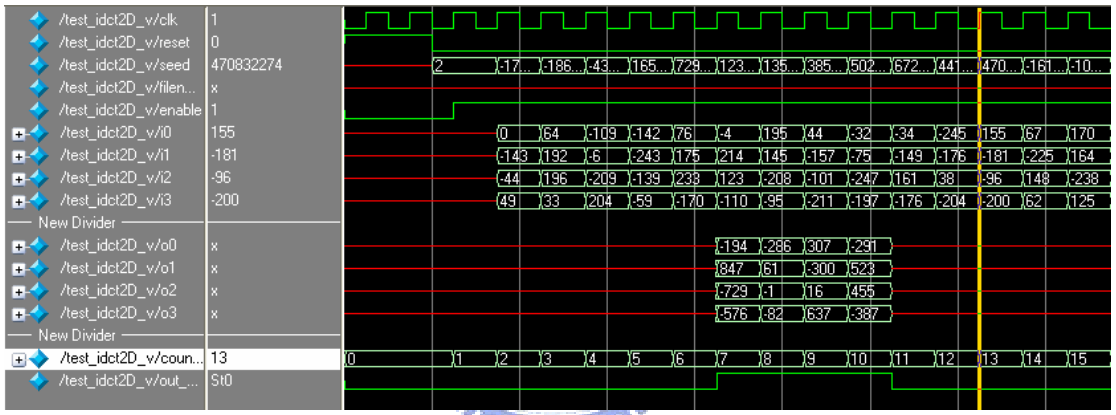


Fig. 27. Timing diagram of 2-D inverse transform

4.2.2 Architecture of the 4x4 inverse hadamard transform

The equation of 1-D inverse 4x4 hadamard transform is illustrated in Eq. 2. We can find that the coefficients of Eq.2 only contains {1,-1}. Therefore, the architecture of 1-D 4x4 inverse hadamard transform is similar with inverse transform and shown in Fig. 28. The difference between 2-D hadamard and 2-D inverse integer transform is the 1-D transform matrix. We can add 1-D inverse hadamard into 2-D inverse integer transform module and add four muxes to choose the corresponding output of both transforms. The idea is illustrated in Fig. 29. The timing diagram is the same in Fig. 27.

$$\begin{bmatrix} x_0' \\ x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} y_0' \\ y_1' \\ y_2' \\ y_3' \end{bmatrix}$$

Eq. 2. Equation of inverse hadamard transform

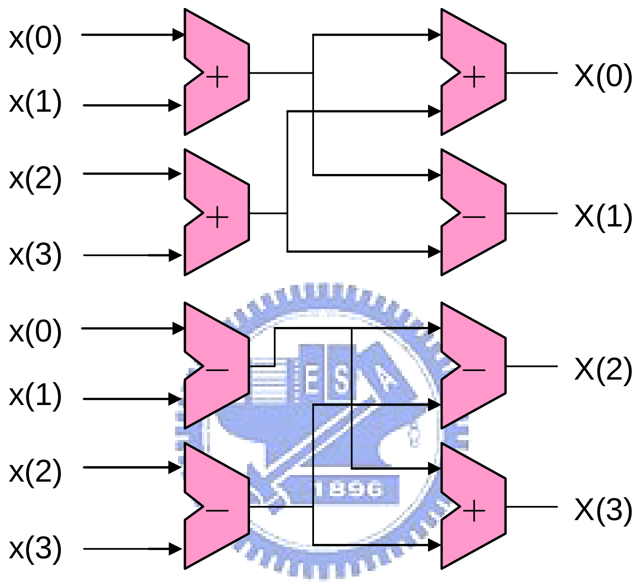


Fig. 28. Architecture of 1-D inverse hadamard

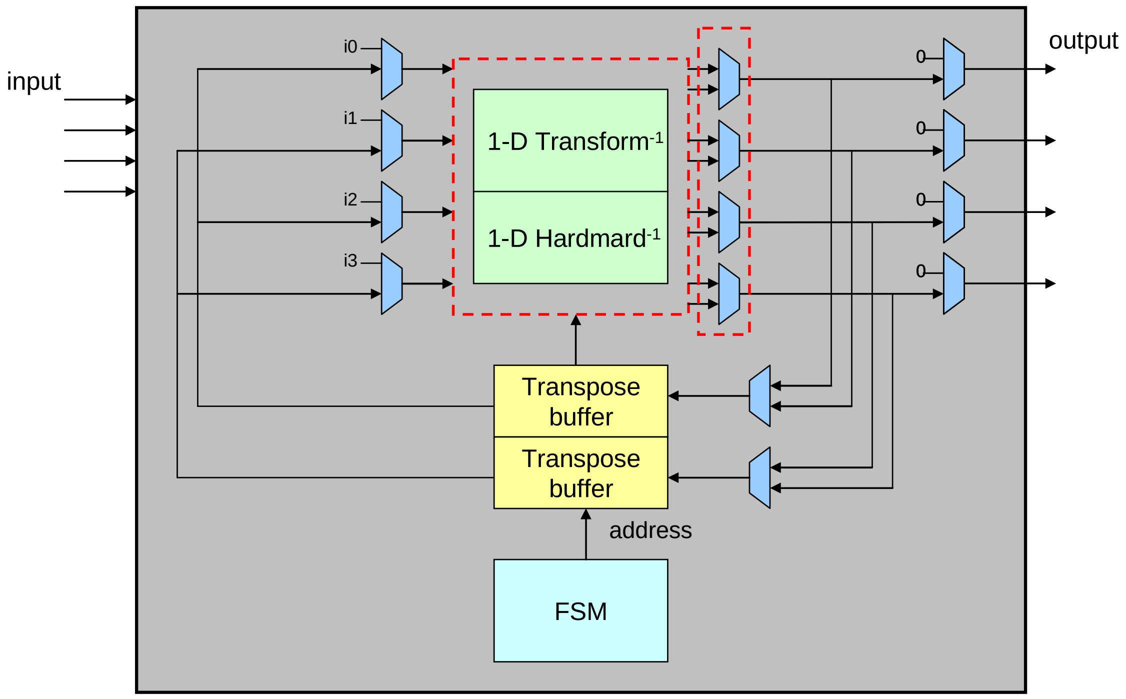


Fig. 29. inverse integer/hadamard transform module

4.2.3 Architecture of the inverse quantizer

The equation of inverse quantizer is shown in Eq. 3, where Vij is showed in Table1 and QP is the parameter of quantization. Base on the equation, the architecture is illustrated in Fig. 30. Instead of using divider, we use block-rom to record the value of QP/6 . And the information of Vij is also stored in block-rom. Besides, we use shifter to compute the operation of $2^{\text{floor(QP/6)}}$. The gate count of multiplier is huge and it reduces the cost of gate counts effectively. It's important to notice that there are two modes in inverse quantizer. When input is DC and it's transformed in I16MB mode, the exponent of 2 is (QP/6)-2 instead of QP/6. Therefore, parameters of inverse quantizer is whether the input is DC and I16MB or not.

The design is a combinational logic. When input is transmitted into inverse quantizer , it outputs immediately. Fig. 31 is test bench simulation with modelsim.

$$W'_{ij} = Z_{ij} \bullet V_{ij} \bullet 2^{floor(QP/6)-2}, \text{if } Z_{ij} = \text{DC and intra16}$$
$$W'_{ij} = Z_{ij} \bullet V_{ij} \bullet 2^{floor(QP/6)}, \text{others}$$

Eq. 3. Equation of inverse quantizer

V_{ij} values			
QP_{step}	Position (0,0),(2,0),(2,2),(0,2)	Position (1,1),(1,3),(3,1),(3,3)	Other position
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Table 2. table of inverse quantizer

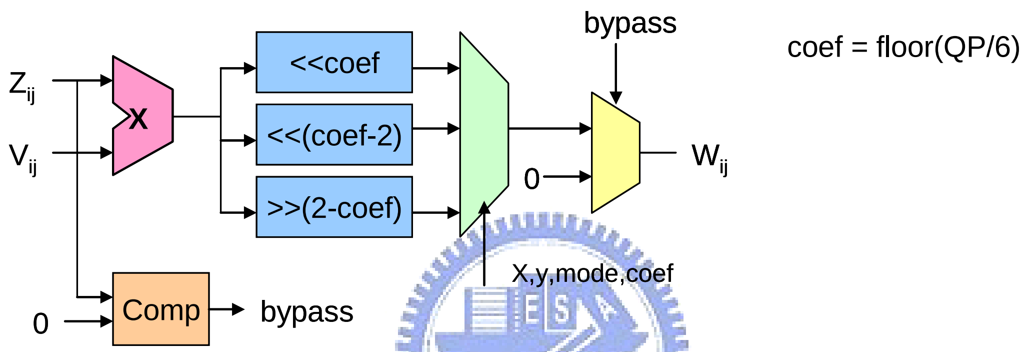
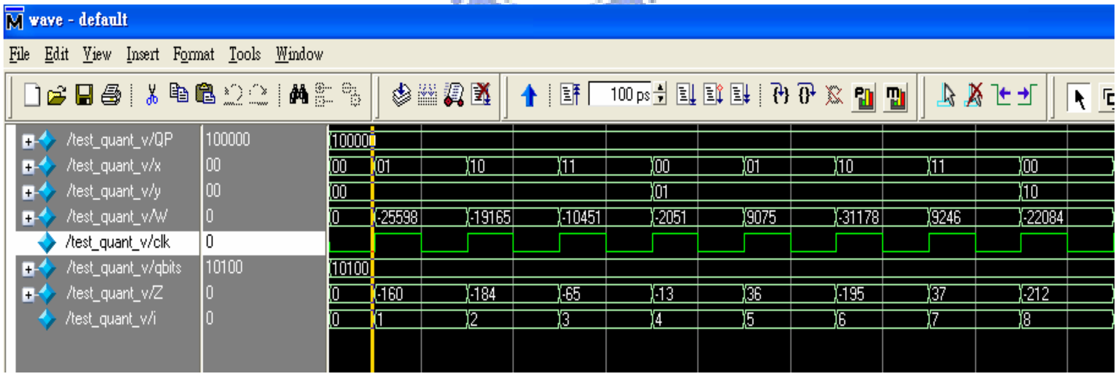


Fig. 30. architecture of inverse quantizer



```
/test_quant_v/QP     100000  10000
/test_quant_v/x      00      00  01      10      11      00     01     10      11    00
/test_quant_v/y      00      00                          01                          10
/test_quant_v/W      0       0   -25598  -19165  -10451  -2051  9075   -31178  9246  -22084
/test_quant_v/clk    0
/test_quant_v/qbits  10100   10100
/test_quant_v/Z      0       0   -160    -184    -65     -13    36     -195    37    -212
/test_quant_v/i      0       0   1       2       3       4      5      6       7     8
```

Fig. 31. Simulation of inverse quantizer

4.3 Design of intra compensation

4.3.1 Intra prediction

In h.264/AVC intra coding (Fig. 32), two types of luma intra macro block is supported. One is for 4x4 prediction mode ,called I4MB and the other is for 16x16 prediction mode, called I16MB. There are nine prediction types in I4MB (Fig. 33) and four in I16MB (Fig. 34). The nine prediction types in I4MB are dc mode, and eight direction modes. The four prediction types in I16MB are vertical mode, horizontal mode, dc mode, plane mode. The plane mode is a bilinear approaching method to calculate the predictor. The predictor is calculated by 13 reconstructed boundary pixels.

For every luma intra macro blocks choose I4MB or I16MB type generating minimal SAD for encoding. The intra prediction of two chroma components is similar to luma I16MB. The only difference between it is block size. The block size of intra prediction for chroma components is 8x8. And they also choose one intra prediction mode from 4 modes.

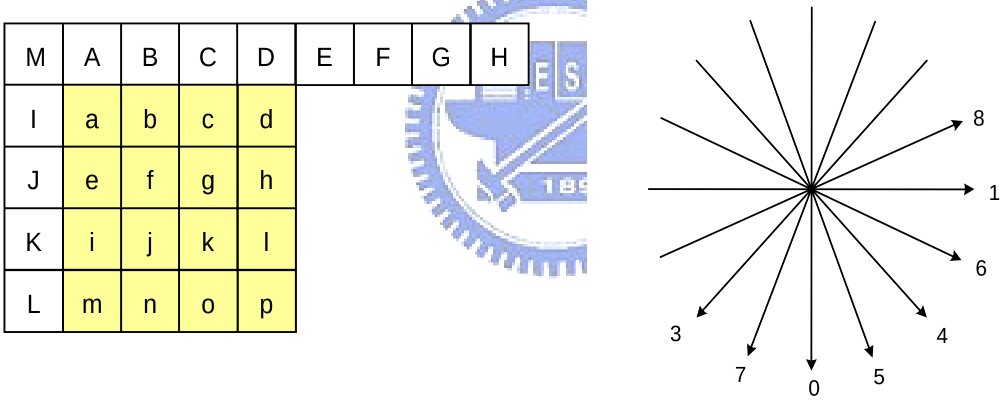


Fig. 32. idea of intra prediction

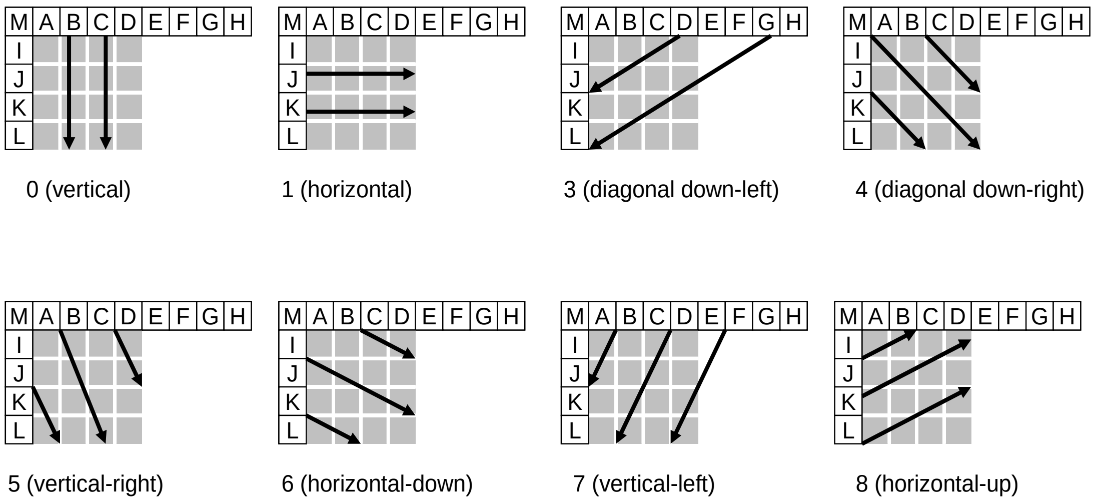


Fig. 33. I4MB prediction mode besides dc prediction

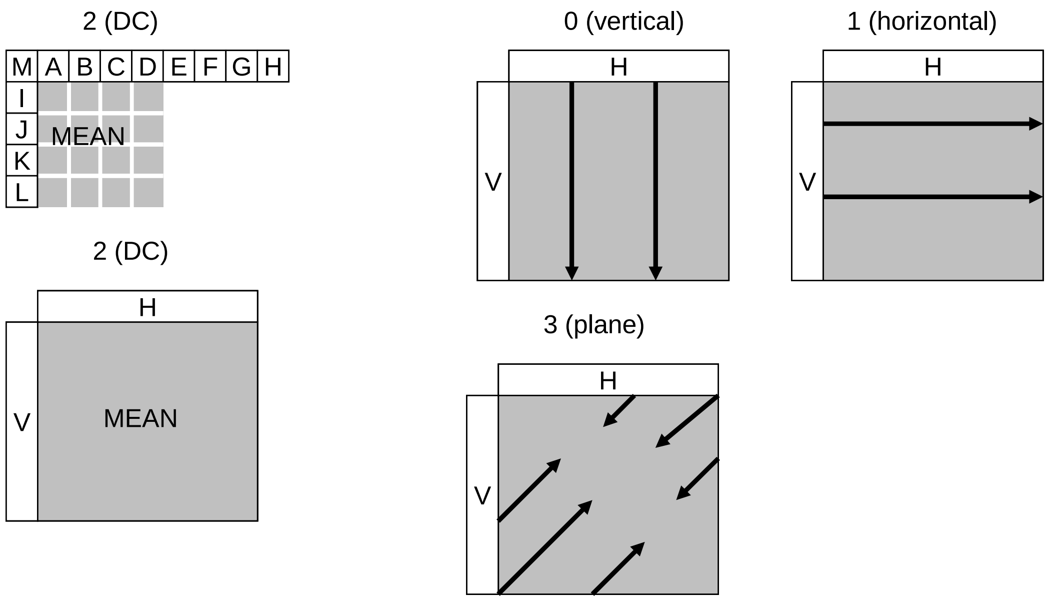


Fig. 34. I16MB predictions mode and one I4MB dc prediction

4.3.2 Proposed architecture of intra compensation

In the proposed architecture, we have two hardware components to support all I4MB modes and DC prediction mode in I16MB. The remaining modes, I16MB vertical mode, I16MB horizontal mode, I16MB plane mode and the chroma prediction modes are implemented in software. To design a hardware module to support vertical and horizontal prediction mode is meaningless since it just choose the top or lest boundary pixels. In addition, designing plane mode in hardware is too complicated and the cost is huge. Therefore, we have decided to put them in software.

4.3.3 Eight directions of I4MB mode

The equation of I4MB except vertical and horizontal mode is shown from Eq.2 to equation 9. We can summary up the equation to Table 3. With the observation, the operation of equations all sum up with at most three digits from neighboring pixels (upper row of left column) and multiply with 2. Since multiplication with 2 is easy to implement, we can add 1 bit zero to the data bus and achieve the goal. However, the critical problem is to choose corresponding boundary pixels according to the address of predictor in 4x4 array and modes. Therefore, we use one complicated neighbor pixels selector, three adders and one shifter to calculate the predictor. The task of the selector, which is composed of lots of multiplexers, is to choose the corresponding boundary pixels with specific address and mode. We use a table to record the state for each address and mode to implement this complicated selector. A counter is used to count the x, y coordinate transmited to the selector for input information. The architecture is illustrated in Fig. 35. Each instance of the module is able to output one predictor in one clock. Four instances of the module is used to generate 4 outputs in one clock. It means that we need 4 clocks to generate all 16 predictors for I4MB.

```
If x==3 && y==3
Pred4x4[3,3] = (p[6,-1] + 3*p[7,-1] +2 ) >>2
else
Pred4x4[x,y] = ( p[x+y,-1] + 2*p[x+y+1,-1] + p[x+y+2,-1]+2) >>2
```

Eq. 4. equation of mode 3 in I4MB

```
If x > y
Pred4x4[x,y] = ( p[x-y-2,-1] + 2*p[x-y-1,-1] + p[x-y,-1]+2) >>2
else if x < y
Pred4x4[x,y] = ( p[-1,y-x-2] + 2*p[-1,y-x-1] + p[-1,y-x]+2 ) >>2
else
Pred4x4[x,y] = (p[0,-1] + 2*p[-1,-1] + p[-1,0] +2 ) >>2
```

Eq. 5. equation of mode 4 in I4MB

```
zVR=2*x-y
If zVR == 0,2,4,6
Pred4x4[x,y] = ( p[x-(y>>1),-1] + p[x-y-1,-1] + 1) >>1
else if zVR== 1,3,5
Pred4x4[x,y] = (p[x-(y>>1)-2,-1] + 2*p[x-(y>>1)-1,-1] + p[x-(y>>1),-1] +2) >>2
Else if zVR == -1
Pred4x4[x,y] = (p[-1,0] + 2*p[-1,-1] + p[0,-1] +2 ) >>2
Else if zVR == -2 or -3
Pred4x4[x,y] =(p[-1,y-1])+2*p[-1,y-2]+p[-1,y-3]+2)>>2
```

Eq. 6. equation of mode 5 in I4MB

```
zHD=2*y-x
If zHD == 0,2,4,6
Pred4x4[x,y] = ( p[-1,y-(x>>1)-1] + p[-1,y-(x>>1)] + 1) >>1
else if zHD== 1,3,5
Pred4x4[x,y] = (p[-1,y-(x>>1)-2] + 2*p[-1,y-(x>>1)-1] + p[-1,y-(x>>1)] +2) >>2
Else if zHD == -1
Pred4x4[x,y] = (p[-1,0] + 2*p[-1,-1] + p[0,-1] +2 ) >>2
Else if zHD == -2 or -3
Pred4x4[x,y] =(p[x-1,-1])+2*p[x-2,-1]+p[x-3,-1]+2)>>2
```

Eq. 7. equation of mode 6 in I4MB

```
If y == 0,2
Pred4x4[x,y] = ( p[x+(y>>1),-1] + p[x+(y>>1)+1,-1] + 1) >>1
else
Pred4x4[x,y] = (p[x+(y>>1),-1] + p[x+(y>>1)+1,-1] + p[x+(y>>1)+2,-1] +2) >>2
```

Eq. 8. equation of mode 7 in I4MB

zHU=x+2*y

If zHU == 0,2,4

Pred4x4[x,y] = (p[-1,y+(x>>1)] + p[-1,y+(x>>1)] +1) >>1

else if zHU== 1,3

Pred4x4[x,y] = (p[-1,y+(x>>1)] + 2*p[-1,y+(x>>1)+1] + p[-1,y+(x>>1)] +2) >>2

Else if zHD == 5

Pred4x4[x,y] = (p[-1,2] + 3*p[-1,3] +2) >>2

Else if zHD >5

Pred4x4[x,y] =p[-1,3]

Eq. 9. equation of mode 8 in I4MB

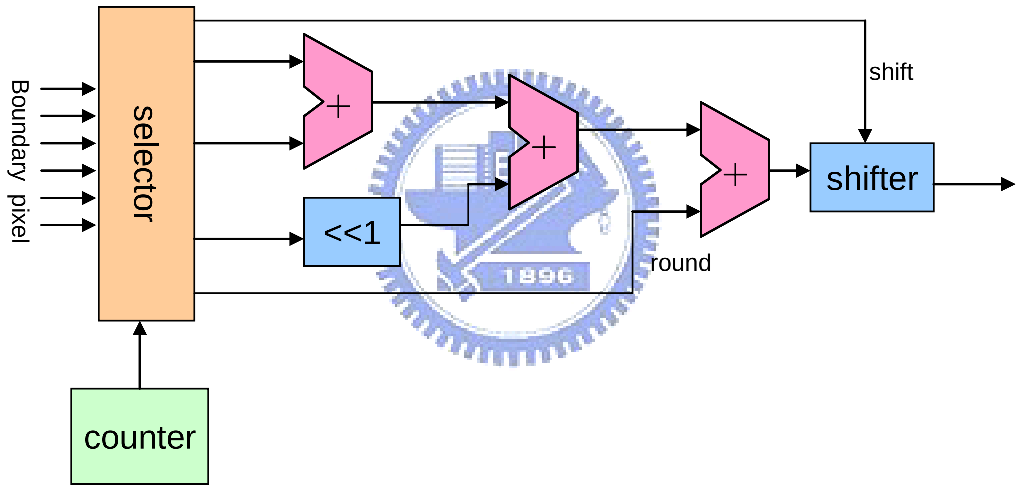


Fig. 35. The architecture of intra compensation.

mode	X	Y	equation	mode	X	Y	equation
3	0	0	A+2*B+C	4	0	0	A+2*M+I
3	0	1	B+2*C+D	4	0	1	M+2*I+J
3	0	2	C+2*D+E	4	0	2	I+2*J+K
3	0	3	D+2*E+F	4	0	3	J+2*K+L
3	1	0	B+2*C+D	4	1	0	M+2*A+B
3	1	1	C+2*D+E	4	1	1	A+2*M+I
3	1	2	D+2*E+F	4	1	2	M+2*I+J
3	1	3	E+2*F+G	4	1	3	I+2*J+K
3	2	0	C+2*D+E	4	2	0	A+2*B+C
3	2	1	D+2*E+F	4	2	1	M+2*A+B
3	2	2	E+2*F+G	4	2	2	A+2*M+I
3	2	3	F+2*G+H	4	2	3	M+2*I+J
3	3	0	D+2*E+F	4	3	0	B+2*C+D
3	3	1	E+2*F+G	4	3	1	A+2*B+C
3	3	2	F+2*G+H	4	3	2	M+2*A+B
3	3	3	G+3H	4	3	3	A+2*M+I
mode	X	Y	equation	mode	X	Y	equation
5	0	0	A+M	6	0	0	M+I
5	0	1	A+2*M+I	6	0	1	I+J
5	0	2	J+2*I+M	6	0	2	J+K
5	0	3	K+2*J+I	6	0	3	K+L
5	1	0	A+B	6	1	0	A+2*M+I
5	1	1	M+2*A+B	6	1	1	M+2*I+J
5	1	2	M+A	6	1	2	I+2*J+K
5	1	3	A+2*M+I	6	1	3	J+2*K+L
5	2	0	B+C	6	2	0	B+2*A+M
5	2	1	A+2*B+C	6	2	1	M+I
5	2	2	A+B	6	2	2	I+J
5	2	3	M+2*A+B	6	2	3	J+K
5	3	0	C+D	6	3	0	A+2*B+C
5	3	1	B+2*C+D	6	3	1	A+2*M+I
5	3	2	B+C	6	3	2	M+2*I+J
5	3	3	A+2*B+C	6	3	3	I+2*J+K
mode	X	Y	equation	mode	X	Y	equation
7	0	0	A+B	8	0	0	I+J
7	0	1	A+2*B+C	8	0	1	J+K
7	0	2	B+C	8	0	2	K+L
7	0	3	B+2*C+D	8	0	3	L
7	1	0	B+C	8	1	0	I+2*J+K
7	1	1	B+2*C+D	8	1	1	J+2*K+L
7	1	2	C+D	8	1	2	K+3*L
7	1	3	C+2*D+E	8	1	3	L
7	2	0	C+D	8	2	0	J+K
7	2	1	C+2*D+E	8	2	1	K+L
7	2	2	D+E	8	2	2	L
7	2	3	D+2*E+F	8	2	3	L
7	3	0	D+E	8	3	0	J+2*K+L
7	3	1	D+2*E+F	8	3	1	K+3*L
7	3	2	E+F	8	3	2	L
7	3	3	E+2*F+G	8	3	3	L

Table 3. Equation of I4MB except for vertical and horizontal modes

4.3.4 DC mode for I4MB mode and I16MB

The dc mode uses the average of boundary pixels as the predictor. Since the computation of the I4MB dc mode and the I16MB dc mode are the same, we use one hardware module with six adders and one accumulator to calculate it (Fig. 36). It takes 4 cycles to finish the computation for I16MB and one cycle for I4MB. Before outputting the result, rounding and clipping the result is a necessary procedure since the range of predictor lies within [0,255]. Therefore, for dc mode in I4MB, it cost 1 cycle to output the result and 4 cycles for I16MB. The timing diagram is shown in Fig. 37 and Fig. 38

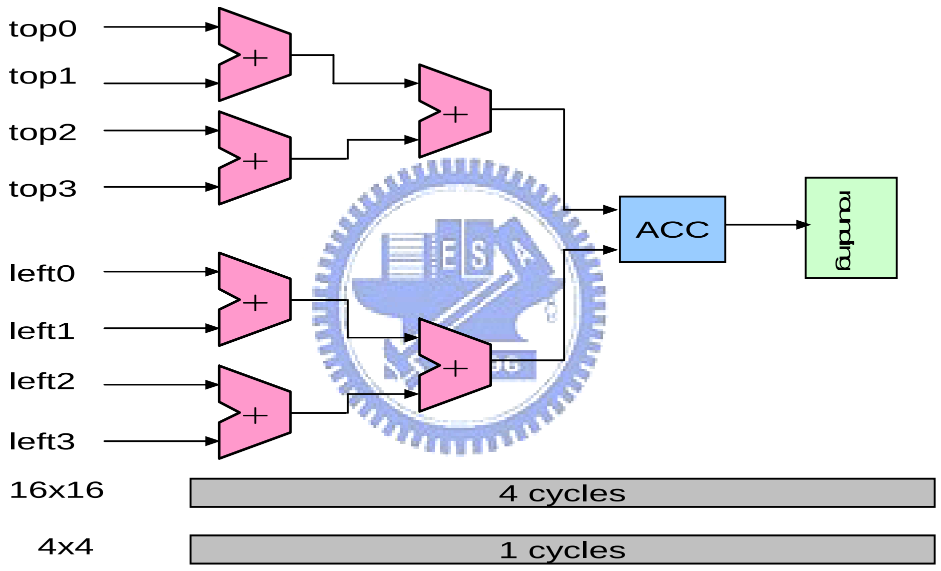
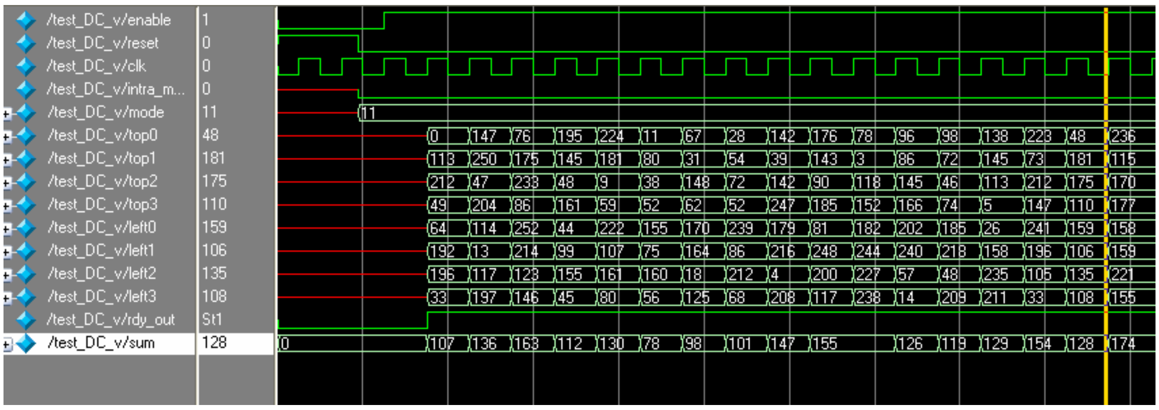


Fig. 36. architecture of dc mode for I4MB and I16MB



```
/test_DC_v/enable      1
/test_DC_v/reset       0
/test_DC_v/clk         0
/test_DC_v/intra_m...  0
/test_DC_v/mode        11    11
/test_DC_v/top0        48    0   147 76  195 224 11  67  28  142 176 78  96  98  138 223 48  236
/test_DC_v/top1        181   113 250 175 145 181 80  31  54  39  143 3   86  72  145 73  181 115
/test_DC_v/top2        175   212 47  233 48  9   38  148 72  142 90  118 145 46  113 212 175 170
/test_DC_v/top3        110   49  204 86  161 59  52  62  52  247 185 152 166 74  5   147 110 177
/test_DC_v/left0       159   64  114 252 44  222 155 170 239 179 81  182 202 185 26  241 159 158
/test_DC_v/left1       106   192 13  214 99  107 75  164 86  216 248 244 240 218 158 196 106 159
/test_DC_v/left2       135   196 117 123 155 161 160 18  212 4   200 227 57  48  235 105 135 221
/test_DC_v/left3       108   33  197 146 45  80  56  125 68  208 117 238 14  209 211 33  108 155
/test_DC_v/rdy_out     St1
/test_DC_v/sum         128   0   107 136 163 112 130 78  98  101 147 155 126 119 129 154 128 174
```

Fig. 37. timing diagram for I4MB dc mode

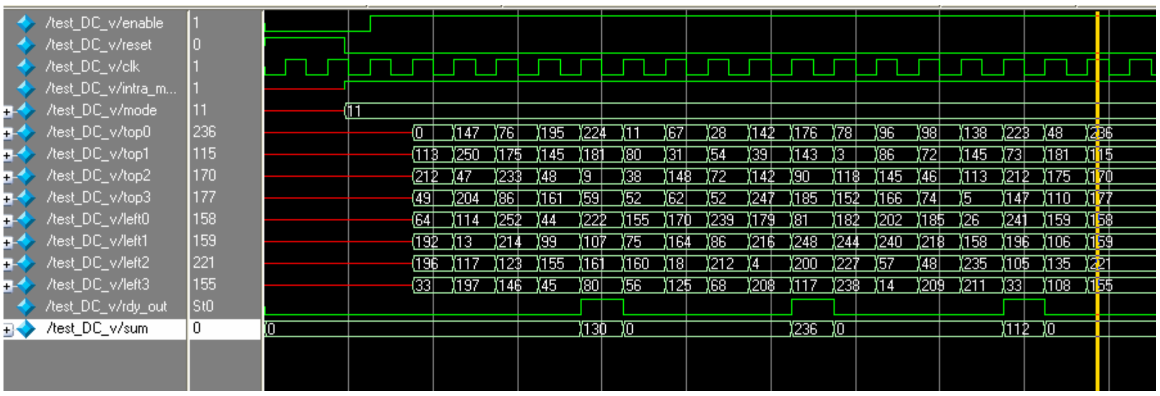


Fig. 38. timing diagram for I16MB dc mode

5. Experimental results

In this section, some experiments are conducted on an SoC emulation platform for an H.264 intra-only decoder (with both software FUs and hardware FUs). The test bitstream is the FOREMAN sequence in QCIF resolution coded at 64K bps. This chapter is organized as follows. First, the emulation platform, ARM integrator, is introduced. Secondly, the synthesis report of hardware functional units is presented. Finally, the performance analysis will be discussed.

5.1 Emulation platform

Design of Soc is much complicated than traditional ASIC design. Therefore, a complete development environment is necessary. We choose ARM INTEGRATOR [22] for our emulation platform. The architecture of arm integrator is shown in Fig. 39. and it is composed of three parts. ASIC Platform (AP)[28], Core Module (CM)[27], and Logic Module (LM)[29]. Each of them will be presented in the followings.

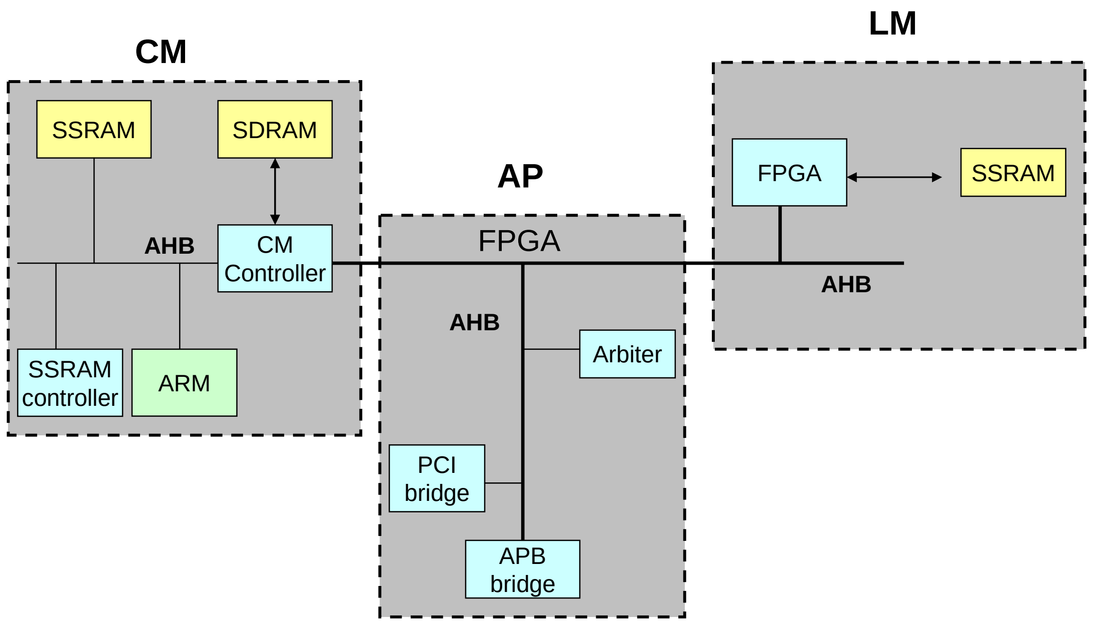


Fig. 39. architecture of arm integrator

5.1.1 Integration board (Arm integrator/AP)

The AP is responsible for connecting between CM and LM. It's shown in Fig. 40. The main component of AP is the system controller FPGA. The system controller FPGA contains several components as followings

- Connector between CM and LM
- AHB bus arbiter
- AHB bus decoder
- Interrupt controller
- Peripheral controller
- System state and control register

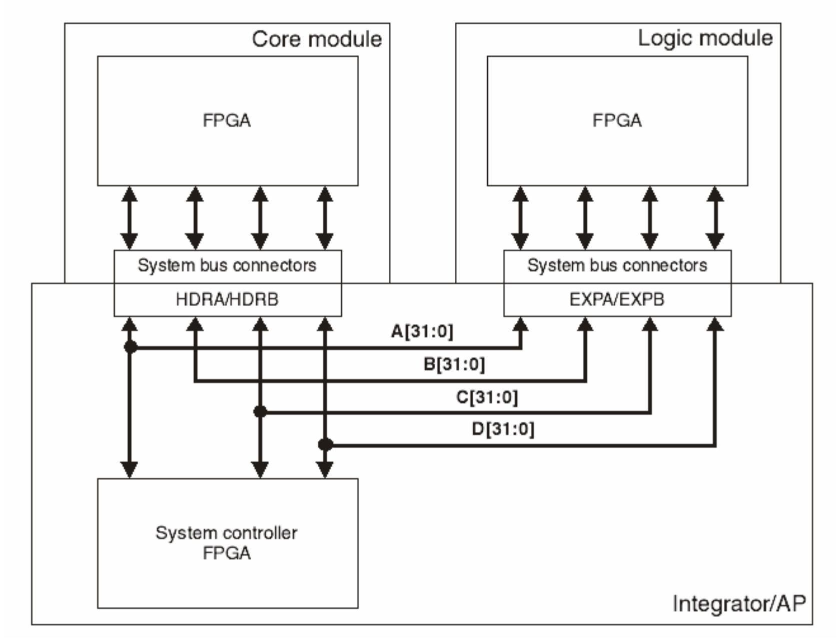


Fig. 40. connection between CM and LM

The AP support the connection between core module and logic module. Our HW functional units are implemented in LM and our GCU, software functional units are running on CM contains a ARM9 processor.

5.1.2 Core module (CM)

Our software functional units and GCU are running on core module. It is composed with followings (Fig. 41).

- ARM9 core

- Core FPGA
    i. SDRAM controller
    ii. System bridge
    iii. Reset controller
- 256KB SSRAM
- Clock generator
- System bus connector
- Multi-ICE interface

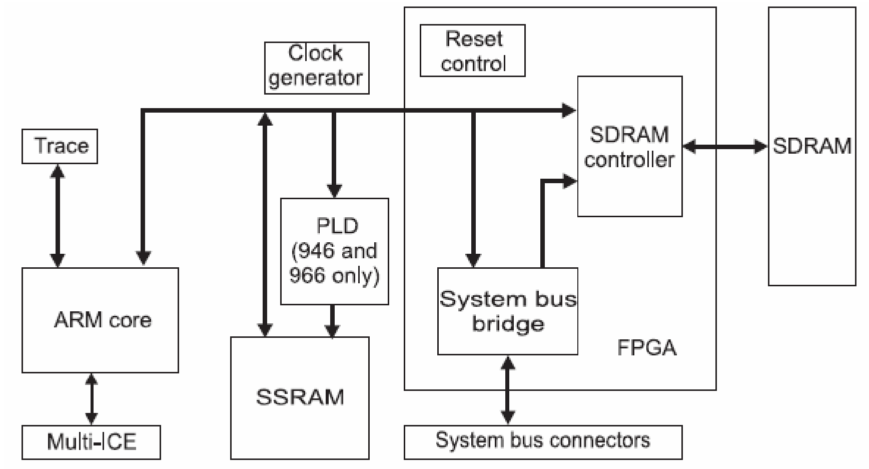


Fig. 41. Block diagram of CM

Core module is a master device in AHB bus and communicates with each master logic (HW functional unit) in logic module.

5.1.3 Logic module (LM)

The version of our logic module is Xilinx virtex 2000E. The block diagram is shown in Fig. 42. It is composed with followings.

- Xilinx FPGA virtex 2000E
- 1 MB ZBT SSRAM
- LED
- System bus connector
- LA connector

- MULTI-ICE interface

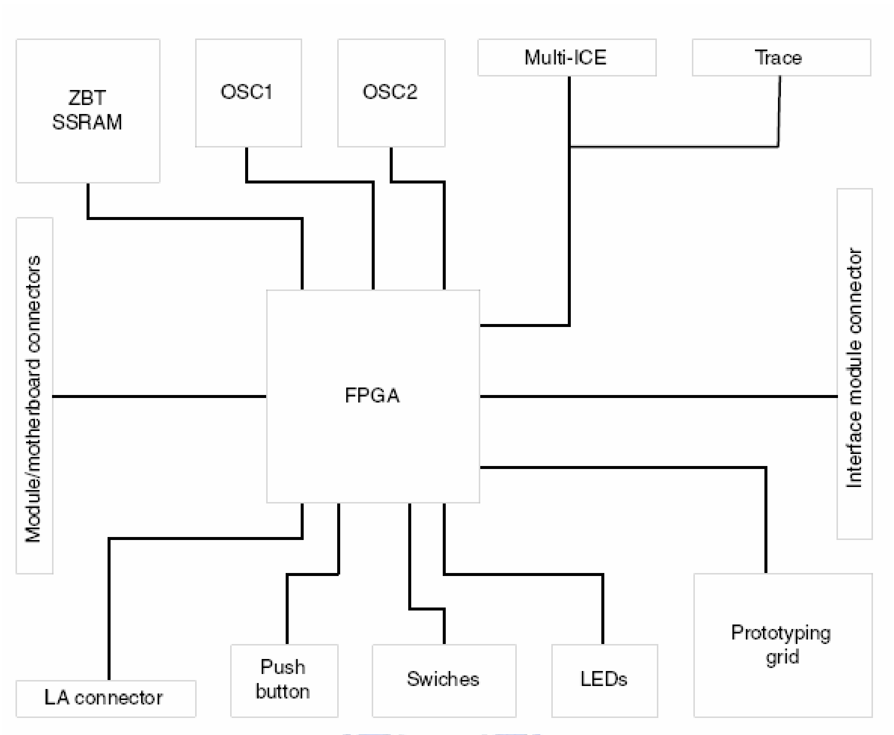


Fig. 42. Block diagram of LM

Our hardware functional units are implemented in LM. All HW functional units are master devices.

5.1.4 memory map for arm integrator

The system memory map on arm integrator is shown in Fig. 43. The original SSRAM is substituted with our scratch-pad memory in LM (Fig. 18). And we organize the LM register in Fig. 44. The register file is composed of enabling register responsible for enabling/disable our hardware functional units, DMA parameters which are passing source address destination address and size of data transfer, inverse transform and inverse quantizer register which are setting these modules, such as type of transform(inverse integer transform or inverse hadamard transform), value of QP…etc.

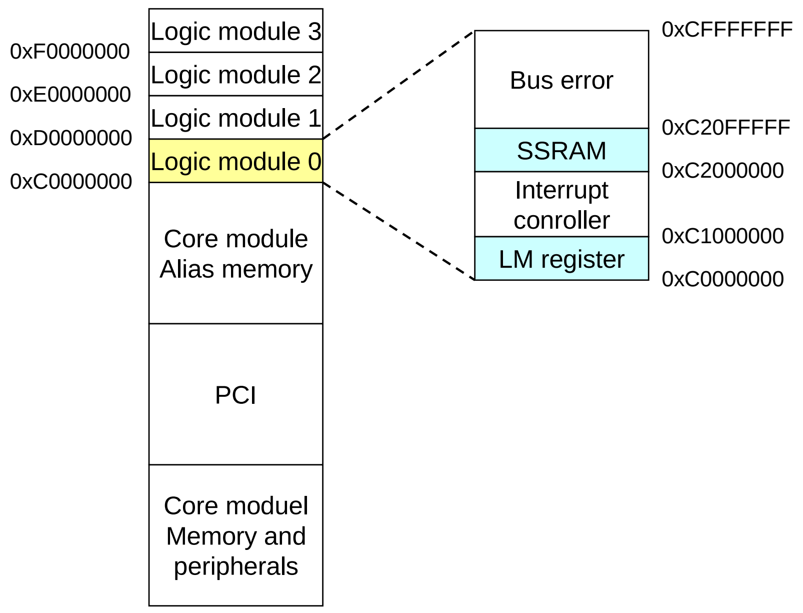


Fig. 43. memory map for arm integrator

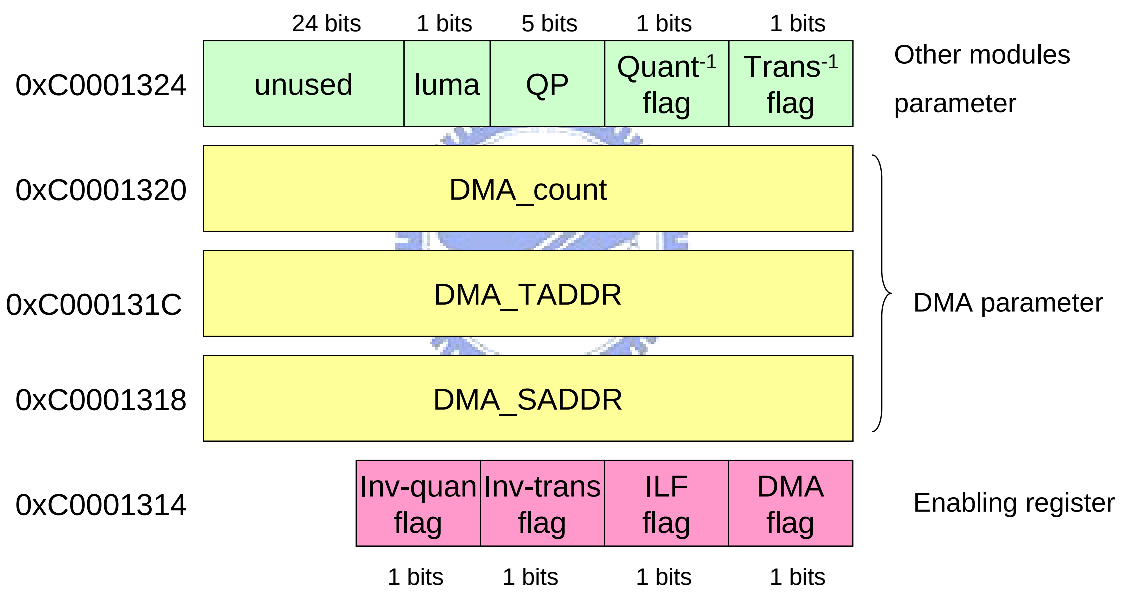


Fig. 44. Parameter register for hardware functional unit

5.2 Synthesis report for hardware functional units

Our target version of FPGA is Xilinx virtex2000E,FG 680, and the synthesis tool is Synplify Pro 8.6. Placing and routing are used Xilinx ISE 7.1. The overall report is shown in Table 4.

Module name	Integer transform^{-1}/ Hadamard^{-1}	Inv quantizer	Intra comp. 1 (other modes)	Intra comp. 2 (DC mode)	Deblocking filter
Clock rate	90 MHZ	Combinational (N/A)	185 MHZ	134 MHZ	76 MHZ
Gates count	476 LUTS	274 LUTS(0%) without MULT 18x18 blocks	1102 LUTS	335 LUTS	868 LUTS
Bandwidth	16/9 (output/clk)	1/1 (output/clk)	4/1 output/clk	1/1 output/clk (I4MB) 1/5 output/clk (I16MB)	4/2 (output/clk)
Block ram usage	2x16x16 bit	96 words by 14 bits 52 words by 5 bits 52 words by 3 bits	NA	NA	NA

Table 4. synthesis report of HW functional units

5.3 Performance analysis

MPEG test bitstream FOREMAN is decoded using the proposed RVC framework. Both ARM 9 in the core module and LM are running at 25 MHZ. However the communication overhead between the CM and LM is very high. It is illustrated in Fig. 45. Network of functional units are composed of software and hardware functional units. Therefore, communication between CM and LM ca not be avoided. However, every data transfer between each other has to go through bus controller in AP which is implemented in FPGA. Besides, the board-level connections between CM, AP, and LM also limit the speed of the bus and data transfer. According to our experiments, it takes 20 to 30 cycles to transfer 32 bits of data. The evaluation time of functional units are shown in Table 5. Inverse integer transform, inverse hadamard transform, inverse quantizer, and intra compensation are discussed in chapter4. The deblocking filter is based on Pens's design [26]. It is obvious that the computation time of HW FUs are much faster then the SW FUs. However, the communication overhead is too high due to the limitation of the ARM Integrator.

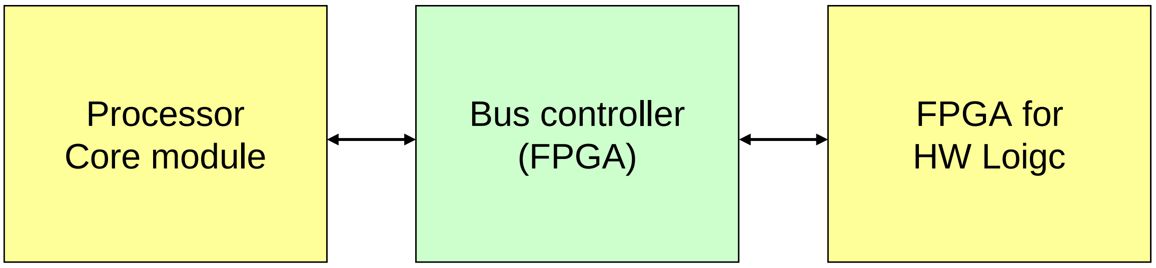


Fig. 45. Communication between CM and LM

	RVC (SW FU's only)	RVC (with HW FU's)	
		Communication Overhead	Computation time
Inv-Transform	6309 ms	3447 ms	247 ms
Inv-qnantize	4224 ms	2687 ms	435 ms
Deblocking filter	7326 ms	2320 ms	65 ms
Intra compensation	1858 ms	972 ms	317 ms

Table 5. evaluation time of functional units

The performance comparison between RVC with pure SW functional units and optimized H.264 SW decoder is shown in Table 6. The optimized H.264 SW decoder used in this experiment is not the H.264 JM reference software [30]. It is an H.264 decoder developed in MMES LAB, NCTU and it is much faster than the JM software. We can see that the overhead of our proposed RVC framework is very small and it cost 2% more than a traditional SW decoder.

	Overall execution time
RVC (pure SW)	102%
Optimized SW decoder	100%

Table 6. Performance comparison

6. Conclusion and Future work

In this thesis, we propose an SoC architecture that supports MPEG Reconfigurable Video Coding framework. Based on the experimental results, the overhead of the proposed RVC architecture can be very small with proper bus bandwidth.

It is important to point out that the purpose of the RVC framework is not to obtain the most efficient design of a single codec, but to allow a flexible and extensible design of codec systems. Multi-standard codec support (or even generating customized codecs on-the-fly) can be achieved by configuring a new GCU via a new decoder description.

An example is shown in Fig. 46. When a new codec is constructed by replacing the MPEG-4 DC/AC prediction tool with the AVC Intra prediction tool, we only need to change the data path between FUs. Of course, here we assume both tools are in the codec toolbox of the chip. In traditional hard-wired HW approach, in order to decode a bitstream of a new codec, one has to redesign the whole HW. The main advantage of RVC is flexibility and quick adaptation to new multimedia standards.

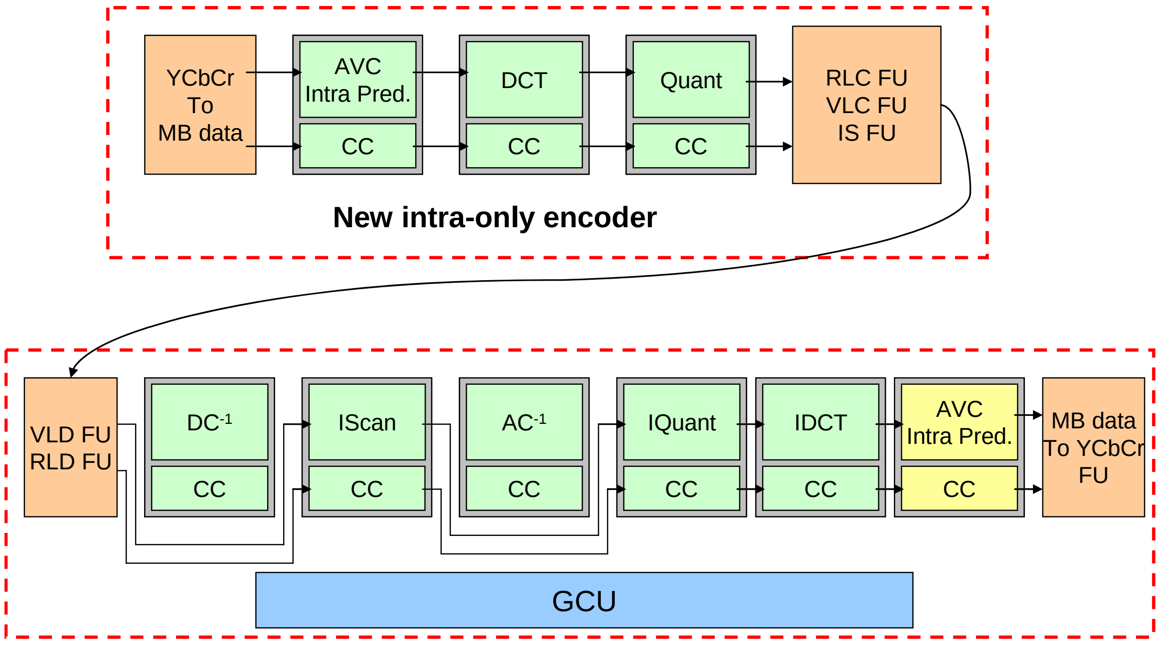


Fig. 46. Construct a new decode on-the-fly

There are still quite some improvements that can be made to the proposed RVC framework. For example, the memory allocator only manages scratch-pad memory in current implementation. However, in a complicated SoC platform, there are different kinds of memory, such as flash, SDRAM, and SRAM. One improvement is that memory allocator can manage several kinds of memory according to their characteristics. In addition, we can add other video coding tools into the toolbox so that other codecs such as MPEG1/2/4 and VC-1 can be supported.

7. Reference

[1] E. S. Jang, K. Asai, and C.-J. Tsai, Study of Video Coding Tool Repository v5.0, MPEG Meeting Document N7329, Poznan, July 2005.

[2] C.-J. Tsai, Suggestions on the Direction of VCTR, MPEG Input Document M12074, Busan, April, 2005.

[3] ISO/IEC MPEG Video Group, Final Call for Proposals on Reconfigurable Video Coding, MPEG Meeting Document N8070, Montreux, April 2006.

[4] J. Janneck et al., Moses Tool Suite, https://sourceforge.net/projects/mosestoolsuite/.

[5] Lin, C.-C. ,Chen, J.-W. ,Chang, H.-C. ,Yang, Y.-C. ,Yang, Y.-H. O. ,Tsai, M.-C., Guo, J.-I., Wang, J.-S. ,A 160K Gates/4.5 KB SRAM H.264 Video Decoder for HDTV Applications, ISSCC 2007

[6] Kuan-Hung Chen ,Jiun-In Guo ,Jinn-Shyan Wang ,A high-performance direct 2-D transform coding IP design for MPEG-4AVC/H.264, IEEE Transactions on Circuits and Systems for Video Technology 2006

[7] Yu-Wen Huang,Bing-Yu Hsieh,Tung-Chien Chen,Liang-Gee Chen “Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder”, IEEE Transactions on Circuits and Systems for Video Technology,2005

[8] Chin-Jen Yang, Bin-Da Liu,Jar-Ferr Yang, “Implementation of JPEG Multimedia system with HW/SW co-design on SoC Development Platform,” NCKU 2002

[9] Kui Zhang and Josef Kittler, “Framework for dynamically reconfigurable video codec using multiple coding tools,” Broadband European Networks and Multimedia Services 1998

[10] Sunyoung Lee, Hyungyu Kim, “Proposed Updates of RVC Working Draft 1.0,” MPEG input document 2006.

[11] Horowitz, M., Joch, A. , Kossentini, F. ,Hallapuro, A. ,”H.264/AVC baseline profile decoder complexity analysis”, IEEE Transactions on Circuits and Systems for Video Technology 2003.

[12] Denolf, K., De Vleeschouwer, C., Turney, R., Lafruit, G., Bormans, J. ,”Memory centric design of an MPEG-4 video encoder”, IEEE Transactions on Circuits and Systems for Video Technology 2003.

[13] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, “Low-complexity transform and quantization in H.264/AVC,” IEEE Trans. Circuits Syst. Video Techno.., vol. 13, no. 7, pp. 598–603, Jul. 2003.

[14] T. Wedi, “Motion Compensation in H. 264/AVC”, IEEE Trans. Circuits System Video Technology 2003

[15] Yueh-Yi Wang ,Yan-Tsung Peng ,Chun-Jen Tsai “VLSI architecture design of motion estimator and in-loop filter for MPEG-4 AVC/H.264 encoders”, ISCAS 2004

[16] Miao Sima, Yuanhua Zhou, Wei Zhang ,“An efficient architecture for adaptive deblocking filter of H.264/AVC video coding”, IEEE Transactions on Consumer Electronics,2004

[17] Wu Di, Gao Wen, Hu Mingzeng, Ji Zhenzhou, “A VLSI architecture design of CAVLC decoder”, ASIC, 2003. Proceedings. 5th International Conference on, IEEE Transactions on

[18] Yao-Chang Yang, Chien-Chang Lin, Hsui-Cheng Chang, Ching-Lung Su, Jiun-In Guo, “A High Throughput VLSI Architecture Design for H.264 Context-Based Adaptive Binary Arithmetic Decoding with Look Ahead Parsing”,IEEE International Conference on Multimedia and Expo,2006

[19] *AMBA Specification 2.0*, ARM Limited, 1999.

[20] R. Peset Llopis , R. Sethuraman “A Low-Cost and Low-Power Multi-Standard Video Encoder”, Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis,2003

[21] T.-C Chen, Y.-W. Huang, and L.-G. Chen, “Analysis and design of macroblock pipelining for H.264/AVC VLSI architecture,” Proc. of IEEE ISCAS 2004, Kobe, 2004.

[22] http://www.arm.com/products/DevTools/IntegratorAP.html

[23] S. Lee, E. S. Jang, M. Matavelli, C. –J. Tsai, Working Draft of ISO/IEC 23001-4: Codec Configuration Representation, MPEG Meting Document N8762, Marrakech, Jan. 2007.

[24] Joint Video Team, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, April 2005

[25] Wiegand, T. Sullivan, G.J. Bjntegaard, G. Luthra, A., Overview of the H.264/AVC video coding standard, IEEE Transactions on Circuits and Systems for Video Technology 2003.

[26] Y-T Peng, VLSI architecture for the in-loop filter of H.264 Video codec, 2004 NCTU

[27] ARM Integrator Core Module/920-T User Guide, ARM Ltd. April 2001

[28] ARM Integrator AP User Guide, ARM Ltd., April 2001

[29] ARM integrator LM-XCV2000E User Guide, ARM Ltd., 2002

[30] JM reference software http://iphome.hhi.de/suehring/tml/