國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

MHP 視訊及圖形加速器於 Xilinx ML403 平台上之設計與實現

Design and Implementation of MHP Video and Graphics Accelerators on Xilinx ML403 Platform

研 究 生：朱浩廷

指導教授：杭學鳴 博士

中 華 民 國 九 十 六 年 八 月

MHP 視訊及圖形加速器
於 Xilinx ML403 平台上之設計與實現

Design and Implementation of MHP Video and Graphics Accelerators on Xilinx ML310 Platform

研究生：朱浩廷　　Student: Hao-Ting Chu

指導教授：杭學鳴 博士　　Advisor: Dr. Hsueh-Ming Hang

國 立 交 通 大 學

電 子 工 程 學 系　電 子 研 究 所

碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

In

Electronics Engineering

2007

HsinChu, Taiwan, Republic of China

中華民國九十六年八月

MHP 視訊及圖形加速器
於 Xilinx ML403 平台上之設計與實現

研究生：朱浩廷　　指導教授：杭學鳴 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘要

MHP (Multimedia Home Platform)是由 DVB Consortium 於 2003 年所公開訂定的數位電視中介層軟體之標準，它允許數位電視在接收視訊及音訊的同時，接收並且執行互動式的應用程式服務。我們已經基於 Linux 及 X window 系統，在 Xlinx ML403 平台上實現了簡化的 MHP 視訊圖形系統，但是這純軟體的實現方式速度不夠快以提供高品質的影像，此研究的目的便是設計與實現硬體加速邏輯，並替換原本軟體的版本，以增加系統效能。

基於軟體系統的 profiling，我們將運算量大的部份以硬體取代。我們為 scaling 及 composition 這兩個運算設計了硬體邏輯，我們研讀了 MHP 標準，簡化演算法以加快運算速度，合成電路並驗

證其正確性。我們也為每個運算發展了獨立的子系統以評估其效能，結果顯示，composition 硬體速度增加約為 50%，而 scaling 硬體則大約為軟體的 6.5 倍快。

最後我們將此兩個硬體邏輯和軟體 JMF 系統整合在一起，為了達成這一目標，我們重建 Linux 核心及一些工具，並成功達成此目標。由於 FPGA 空間的限制，並沒有加入硬體 DMA 而導致系統效能不如預期中的好。此一計畫也成功驗證軟硬體相互設計流程在 ML403 平台上的可行性。

Design and Implementation of MHP Video and Graphics Accelerators on Xilinx ML403 Platform

Student: Hao-Ting Chu Advisor: Dr. Hsueh-Ming Hang

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

Abstract

MHP is an open DTV middleware standard developed by the DVB Consortium in 2003. It enables the reception and execution of interactive applications with TV programs. We have implemented a simplified MHP (Multimedia Home Platform) video and graphics system on the Xilinx ML403 platform under the Linux and X Window System. This pure software approach is not fast enough to provide high quality video. The purpose of this study is to design and implement hardware IPs to replace their software counterparts and thus improve performance.

Based on the profiling data of the software system, we move the computation-intensive tasks to the hardware. We design two accelerating IPs, the scaling and the composition operations. We study their MHP specifications, simplify the algorithms for speed-up, synthesize the circuits, and verify their correctness. We also implement standalone subsystem for each operation to evaluate their performance. The results show that the composition hardware increase the speed by about 50% and the scaling hardware runs at about 6.5 times faster.

The final step in our implementation is to integrate these two IPs together with the software JMF system. For this purpose, we rebuild the Linux kernel and reconstruct a few tools. At the end, the software and hardware integration has been done successfully. Due to the limited size of FPGA, the hardware DMA is not included, and thus the system performance is improved but not as much as expected.

This project also successfully demonstrates the flow of software and hardware co-design on the ML403 platform.

致謝

感謝杭學鳴老師兩年來的教導，無論在研究上或是其他方面，老師真的很感謝您。謝謝實驗室的全體學長、同學，尤其是峰誠學長，學長總是不時的給予我適當的幫助，在研究上幫助了我很多，遇到瓶頸時和學長的討論，總是讓我有所突破。最後，也要謝謝一路陪著我的家人朋友，謝謝。

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

The transition from analog TV to digital TV (DTV) is on-going now. DTV is an advanced broadcasting technology that will enhance and transform your television viewing experience. DTV enables broadcasters to offer television with movie-quality picture and sound. It can also offer multiple programming choices, called multicasting and interactive capabilities, which is not possible by the analog broadcast systems.

Digital Video Broadcasting (*DVB*) is one of the most widely adopted DTV transmission standards in the world. The *DVB Multimedia Home Platform* (*DVB-MHP*) defines a Java-based platform for developing interactive DTV applications. In addition to providing abstractions for many DVB and MPEG-2 concepts, it defines interfaces for features such as network card control, application download, and layered graphics model, and others.

A full MHP system is very complicated. It is composed of various functional components. We have already implemented a pure software video and graphics subsystem, but its speed needs improved to provide high quality video service. In this thesis, we design and implement two hardware IPs to replace their software counterparts. We also need to integrate these IPs with the existing software system. These are the two main tasks in this study.

As described in the following chapters, we design two accelerating IPs, the scaling and the composition operations. We study their MHP specifications, simplify the algorithms for speed-up, synthesize the circuits, and verify their correctness. We also implement standalone subsystem for each operation to evaluate their performance. The results show that the composition hardware increase the speed by about 50% and

the scaling hardware runs at about 6.5 times faster. We also integrate these two IPs together with the software JMF system successfully.

This thesis is organized as follows. In Chapter 2, a brief overview of the MHP graphics system is provided, which includes the DVB standards, the MHP architecture, the MHP graphics model and the CoreConnect bus. Chapter 3 describes the objectives of the proposed design, as well as the FPGA tools, and the target environment to develop and deploy the implementation. Chapter 4 describes the design flow of the hardware IPs, the standalone subsystems, and the integration with the existing JMF software system. Also, we evaluate the designed circuit performance. Chapter 5 includes the conclusion remarks and a few possible future work items.

Chapter 2
MHP Graphics

2.1 Digital Video Broadcasting

Digital Video Broadcasting (*DVB*) is a suite of internationally accepted open standard for digital television. DVB systems distribute data using a variety approaches, including satellite (*DVB-S*), cable (*DVB-C*), terrestrial television (*DVB-T*), terrestrial television for handhelds (*DVB-H*), and others. These standards define the physical layer and the data link layer of a distribution system. All data of these standards is transmitted in MPEG-2 transport streams with some additional constraints (*DVB-MPEG*). In most of the European, Asian and South American countries, the DVB systems have been adopted.

There are two major competitive standards for DVB: the *Advanced Television Systems Committee* (*ATSC*) and the *Integrated Services Digital Broadcasting* (*ISDB*). ATSC has been adopted in some countries like the U.S., Canada, Mexico and South Korea. ISDB is developed by Japan for digital television and digital audio broadcasting. However, the DVB standards have been adopted by most countries in the world.

ATSC has been considered by the Ministry of Transportation and Communication (MOTC) originally. But, over-the-air TV broadcasters and manufactures tended to adopt DVB, because of its superiorities in mobile reception, indoor reception and less interference by obstacles in metropolises. In June 2000, the DVB-T standard has been adopted by the MOTC of Taiwan for DTV broadcasting.

As described earlier, a digital TV signal is transmitted as a stream of MPEG-2

data known as a *transport stream*. Each transport stream has a data rate of up to 40 megabits/second for a cable or satellite network, which is enough for seven or eight separate TV channels, or approximately 25 megabits/second for a terrestrial network. In DTV-speak, each transport stream is also known as a *multiplex*. Within a multiplex, groups of elementary streams that make up a single TV channel are called *services* or *programs*. TV shows in a program are known as *events*. The transport stream physically groups a set of services together, but services in DVB systems can also be logically grouped as well. Logical groups of services are called a *bouquets*. Sets of transport streams that share some common service information are called *networks*.

2.2 Multimedia Home Platform

Multimedia Home Platform (*MHP*) [1] is the open middleware system designed by the DVB Project for interactive digital television.The MHP enables the reception and execution of interactive, Java-based applications on a TV-set. The applications can be delivered over the broadcast channel, together with audio and video streams. For all interactive applications an additional return channel is needed.

The MHP model considers 3 layers: *resources*, *system software*, and *applications*. (Figure 2.1) The resources layer collects hardware devices and basic software, like processor, media decoder, remote control, the tuner, demultiplexer, graphics subsystems, memory subsystem, and so on. The resources will not be addressed by applications directly. The system software brings an abstract view of such resources. The applications have been isolated from the hardware, enabling portability of the applications. How to implement the Resources and System software are not in the specification. The system software includes an *application management* function, also known as a *navigator,* which is responsible for managing the lifecycle of all applications. Applications implement interactive services as software running in one or more hardware entities. The interface for MHP applications is a top view from

application to the system software. [1,pp.42]

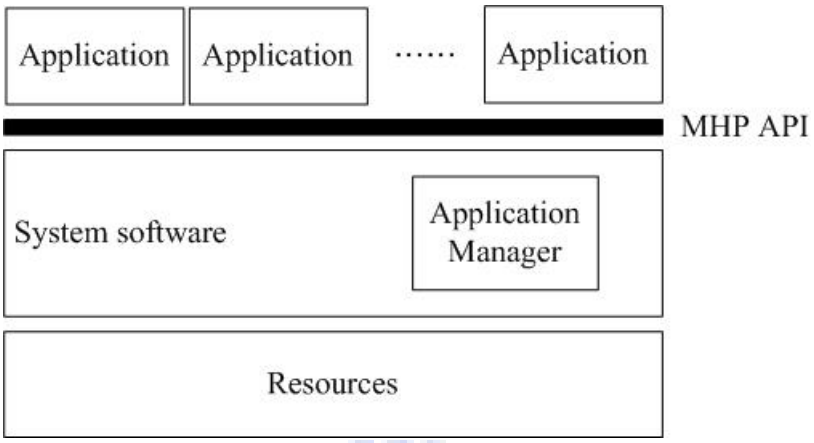


Figure 2.1 MHP architecture

[1,Figure 4,pp.42]

Applications use the MHP API to access the actual resources of the receiver, including: databases, streamed media decoders, static content decoders and communications. The diagrams in Figure 2.2 show the components of an MHP system, and the interface between an MHP system and MHP applications. The MHP API is shown as the bold border enclosing the applications in Figure 2.2.

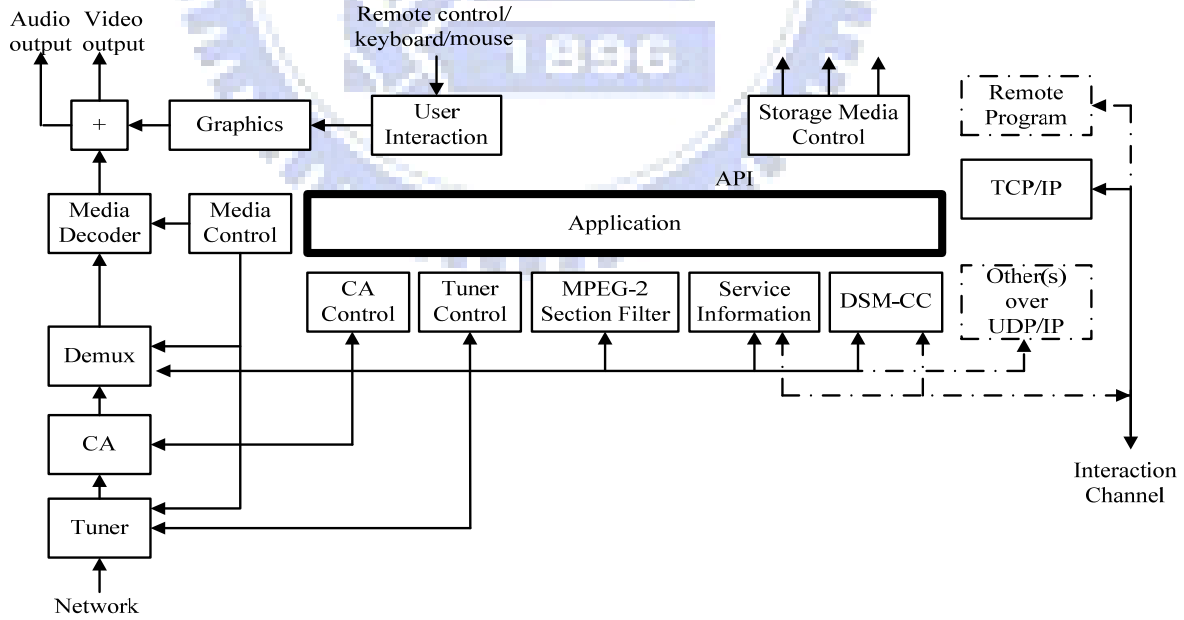


Figure 2.2 MHP system block diagram

[1,Figure 6,pp.45]

2.3 MHP Graphics Model

The MHP graphics model [1, pp.197-217] provides tools to control the poisoning of: video on an output device, interface components, as well as war graphical primitives. In the MHP system, each screen is composed of three planes which are, from front to back, one or more graphics planes, one or more video planes and a background plane. (Figure 2.3)

The background plane has capability of displaying single solid color, a still image, or a video drip, depending on the capability of the MHP receiver.

The MHP specification enables terminals to support multiple applications at any one time, each of which can have a sub area of the screen to which it can draw. The specification enables the areas to overlap.

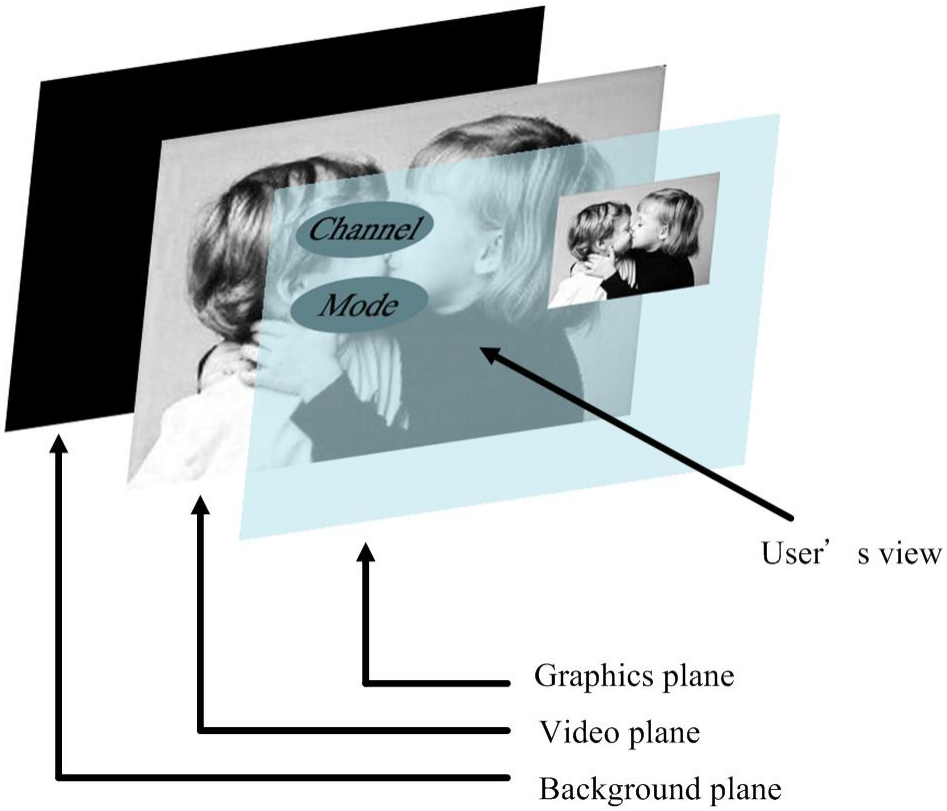


Figure 2.3 An example of MHP display stack

[1, Figure 16, pp.197]

There are three sources of graphical content: background, video and application graphics, and these three sources are mapped in a controllable way onto single screen. The figure shows the conceptual processing functions, the content stages, and application points inside the content pipelines. The figure also shows that applications have full control over the source of the content that enters the different pipelines. Furthermore the application may control clipping, scaling and positioning of the content at different stages in the different pipelines.

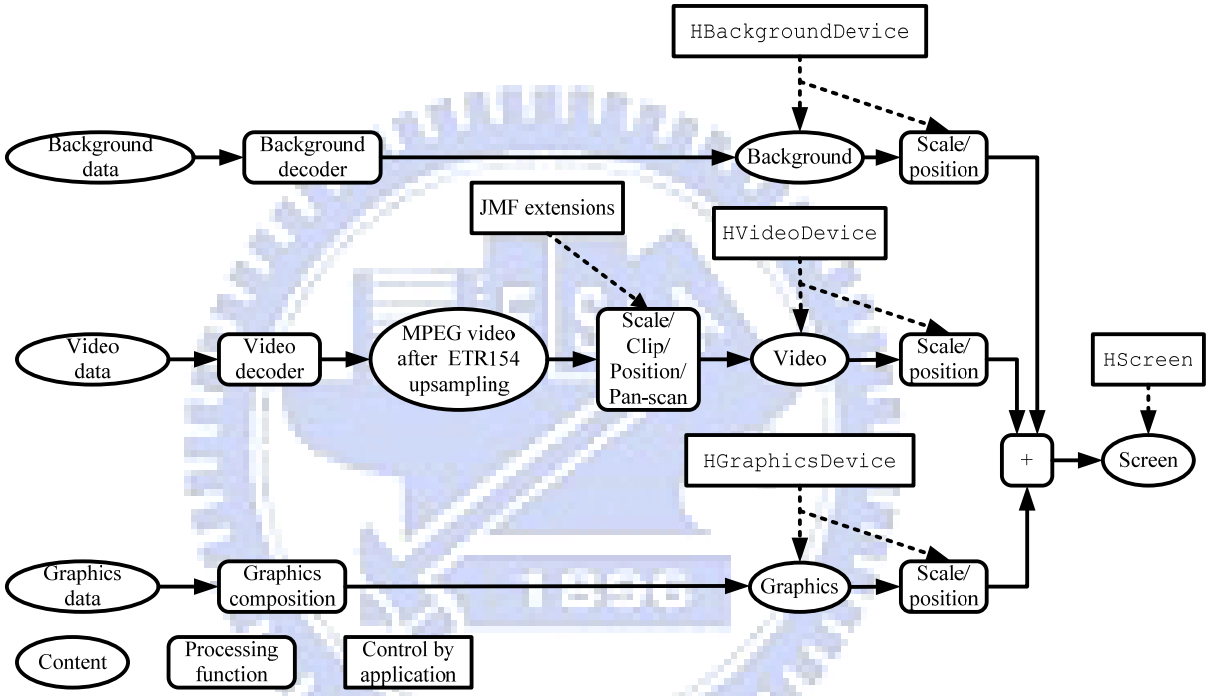


Figure 2.4 Background, video, and graphics pipeline
[1, Figure 22, pp.203]

Figure 2.5 shows that the behavior of the graphics composition process has three elements.

- The graphics components are composed following the traditional AWT graphics model using *Porter-Duff rules*. (The default rule used is the *SRC_OVER* rule.)
- The background and video planes are composed using the Porter-Duff rule *SRC*. The video plane has alpha values of either 0.0 or 1.0

- The results are composed together using the SRC_OVER rule with the graphics results as the source and the results of the background/video composition as the destination

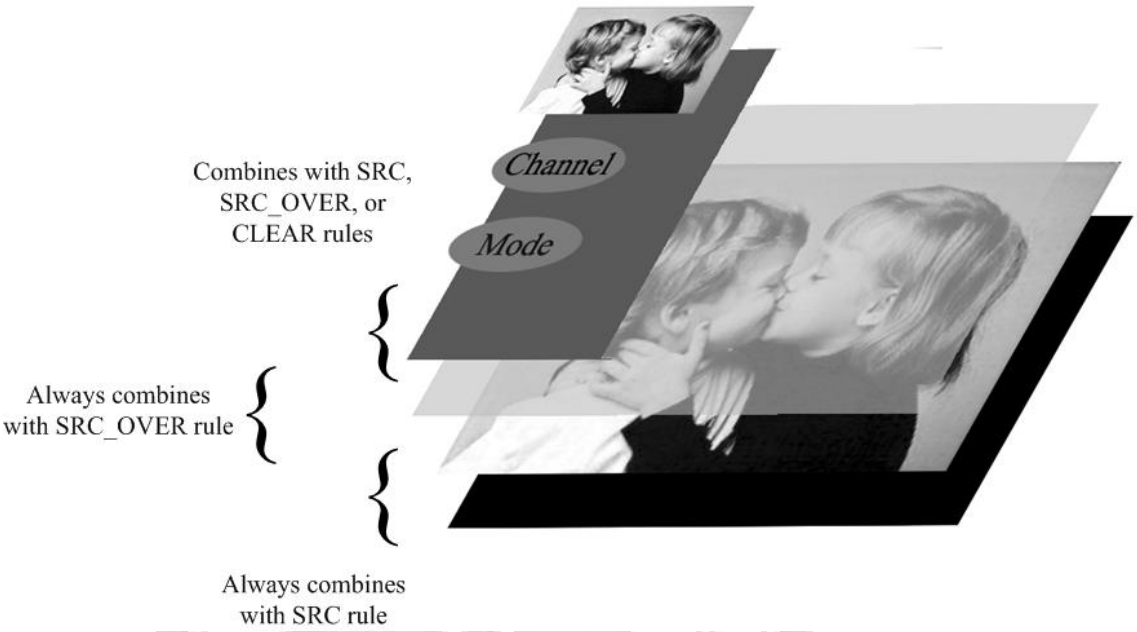


Figure 2.5 MHP composition rules
[1, Figure 24, pp.204]

The Porter-Duff rules are alpha composition rules for combining source and destination pixels to achieve blending and transparency with graphics, image, and video. The composition operations are computed as follows:

$$Cd = Cs \times Fs + Cd \times Fd$$
$$Ad = As \times Fs + Ad \times Fd \quad ,$$

where the abbreviations are used:

- *Cs*: one of the color components of the source pixel.
- *Cd*: one of the color components of the destination pixel.
- *As*: alpha component of the source pixel.
- *Ad*: alpha component of the destination pixel.
- *Fs*: fraction of the source pixel that contributes to the output.
- *Fd*: fraction of the destination pixel that contributes to the output.

Each Porter-Duff composition rule is specified by *Fs* and *Fd*. Figure 2.6 shows the typical implementation of SRC_OVER rule.

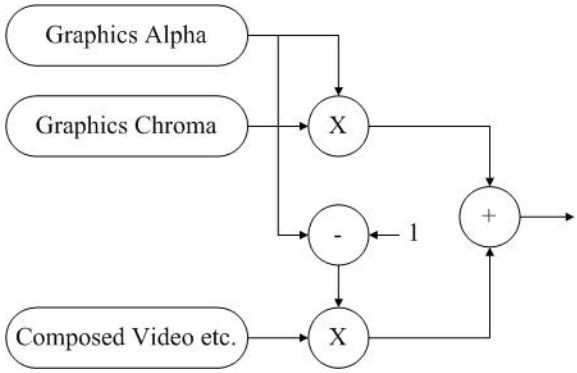


Figure 2.6 Typical implementation of SRC_OVER rule
[1, Figure 35, pp.215]

The SRC rule simply requires that the color and alpha characteristics of the new pixels replace those previously present. (SRC_OVER rule equals to SRC rule when the graphics alpha value is 1.)

The Porter-Duff rules are composed of eight rules, which are: CLEAR, DST_IN, DST_OUT, DST_OVER, SRC, SRC_IN, SRC_OUT and SRC_OVER. MHP terminals are required to implement at least the SRC, SRC_OVER and CLEAR rules.

In the composition of the graphics with background and video planes, MHP terminals are required to implement at least three levels of transparency: 0% (opaque), 100% (completely transparent) and an intermediate value of approximately 30%. Implementation of additional intermediate levels of transparency is optional.

MHP terminals are required to support both displaying MPEG video without any scaling and with 1/2 scaling both vertically and horizontally provided that in this latter case the entire resulting video area is fully on the screen.

2.4 The CoreConnect Bus

CoreConnect [2][3], the IBM PowerPC processor bus, has been licensed at no cost to many intellectual property (IP) developers in the past decade, creating one of

the world's largest I/O core ecosystems. Figure 2.5 show that the elements of the CoreConnect bus architecture include the *processor local bus* (*PLB*), the *on-chip peripheral bus* (*OPB*), a bus bridge, and a *device control register* (*DCR*) bus. High performance peripherals connect to the high-bandwidth, low-latency PLB. Slower peripheral cores connect to the OPB, which reduces traffic on the PLB, resulting in greater overall system performance. Bus model toolkits are available to assist in designing custom logic to the PLB/OPB/DCR bus standards. The CoreConnect bus architecture eases the integration and reuse of processor, system, and peripheral cores within standard product and custom *system-on-chip* (*SOC*) designs.

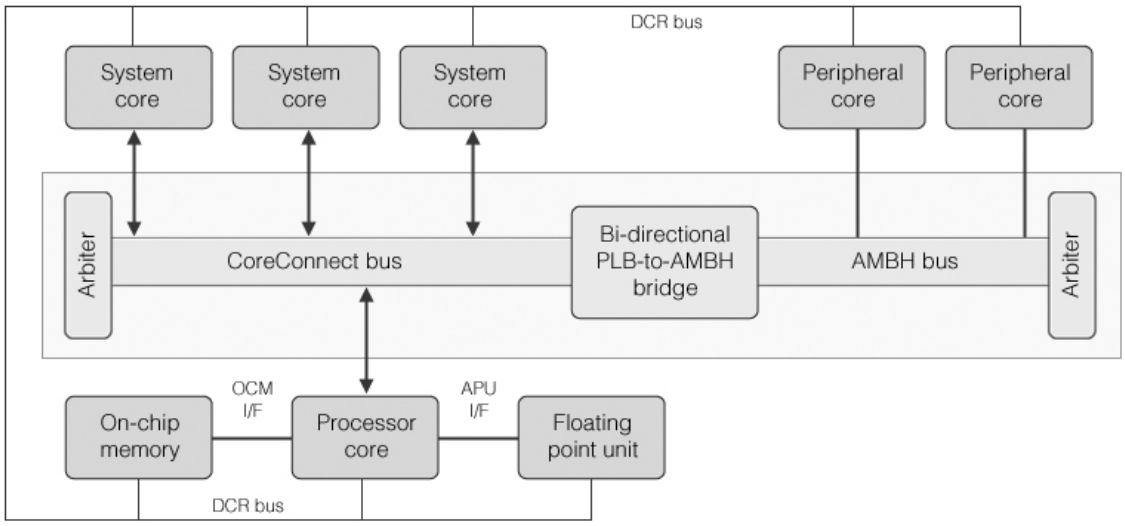


Figure 2.7 The CoreConnect bus architecture

2.4.1 Processor Local Bus (PLB)

The PLB bus is a high performance, synchronous on chip bus. The PLB bus interface provides 32-bit address, 64-bit write and 64-bit read data bus. It also provides read and write transfers between master and slave devices. Each PLB master has separate address, read data, write data, and transfer qualifiers. However, the PLB slaves have shared, but decoupled, address, read data, write data, transfer qualifiers, and status signals. (Figure 2.8) The data accessing is granted through a central arbitration mechanism.

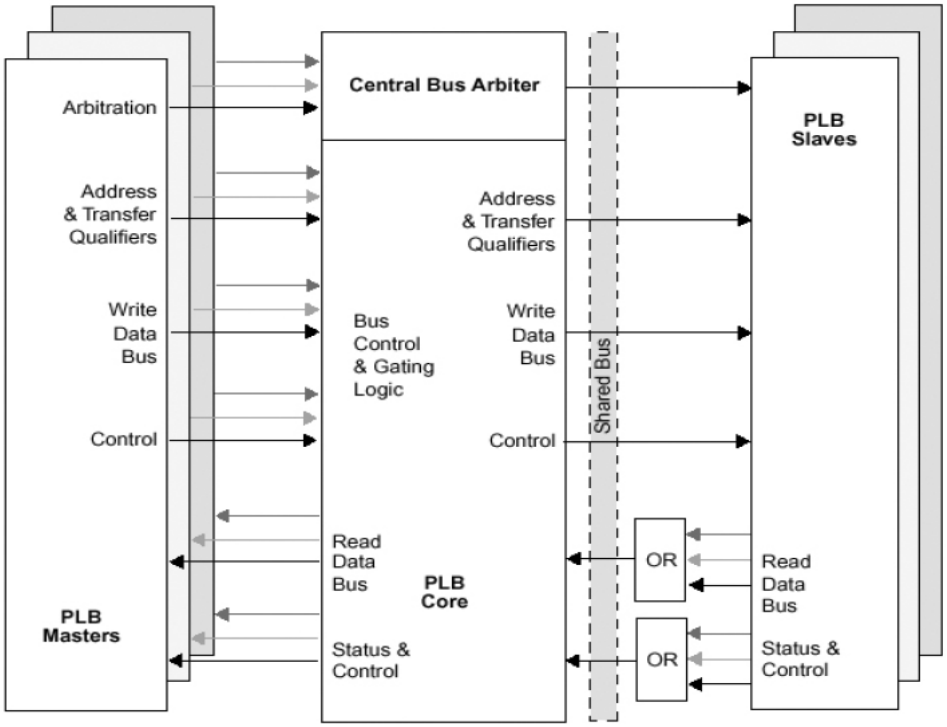


Figure 2.8 The PLB bus diagram

The PLB bus has the following features:

- Architecture supports eight masters.
- Timing is provided by a single, shared clock source.
- Overlaps read and write transfer permits two transfers per clock.
- Four levels of request priority for each master.
- Byte enables for unaligned halfword and three-byte transfers.
- Support for 16-, 32- and 64-bit line data transfers.
- Variable or fixed length burst transfers supported.

Figure 2.9 shows that the PLB transfer protocol: address cycle has three phases (request, transfer, and address acknowledge) and data cycle has two phases (transfer and data acknowledge).

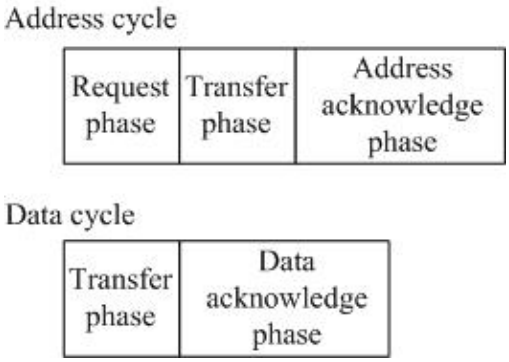


Figure 2.9 The PLB transfer protocol

The PLB bus interface has decoupled address, read data, and write data bus. So that it can support overlapped transfers, including address cycle overlapped with read or write data cycle and read data cycle overlapped with write data cycle. (Figure 2.10) Furthermore, the PLB bus interface support address pipelining to enable next address transfer to begin before current data transfer has completed. Address pipelining and overlapped transfers reduce bus latency.

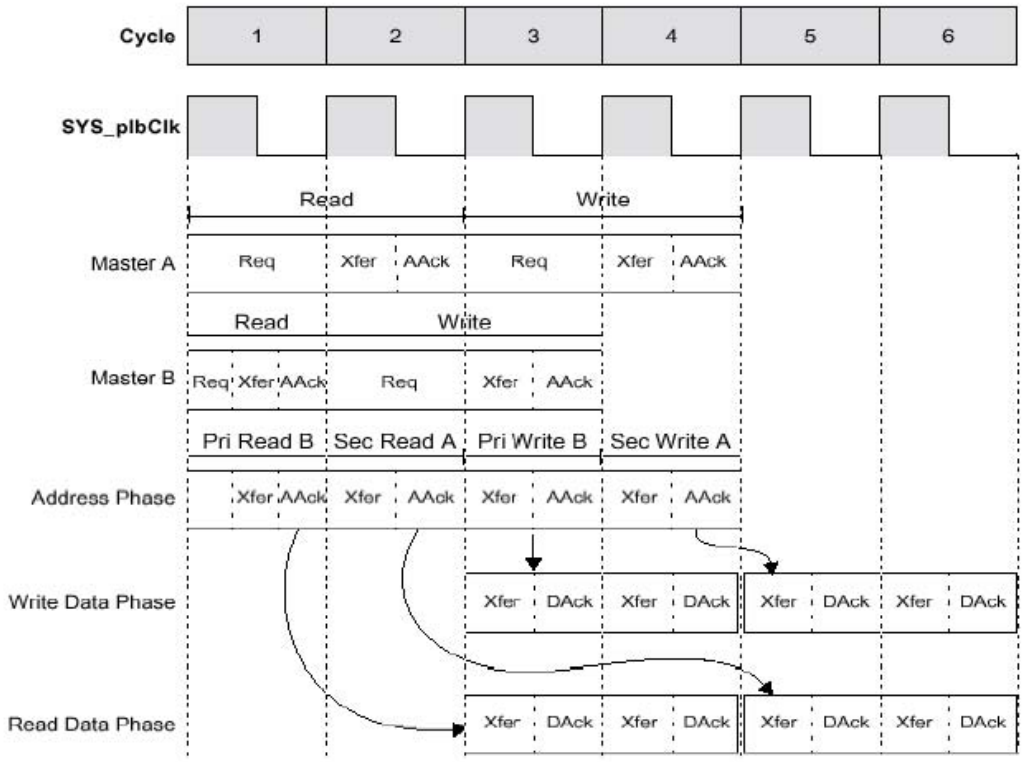


Figure 2.10 Overlap PLB transfers

2.4.2 On-Chip Peripheral Bus (OPB)

The OPB bus is architected to alleviate system performance bottlenecks by reducing capacitive loading on the PLB. And it has the following features:

- Fully synchronous.
- 32-bit address bus, 32-bit data bus.
- Supports single-cycle data transfers between master and slaves.
- Support multiple masters, determined by arbitration implementation.
- Bridge function can be master on PLB or OPB.(PLB to OPB bridge or OPB to PLB bridge)
- No tri-state drivers required.

2.4.3 Device Control Register (DCR) Bus

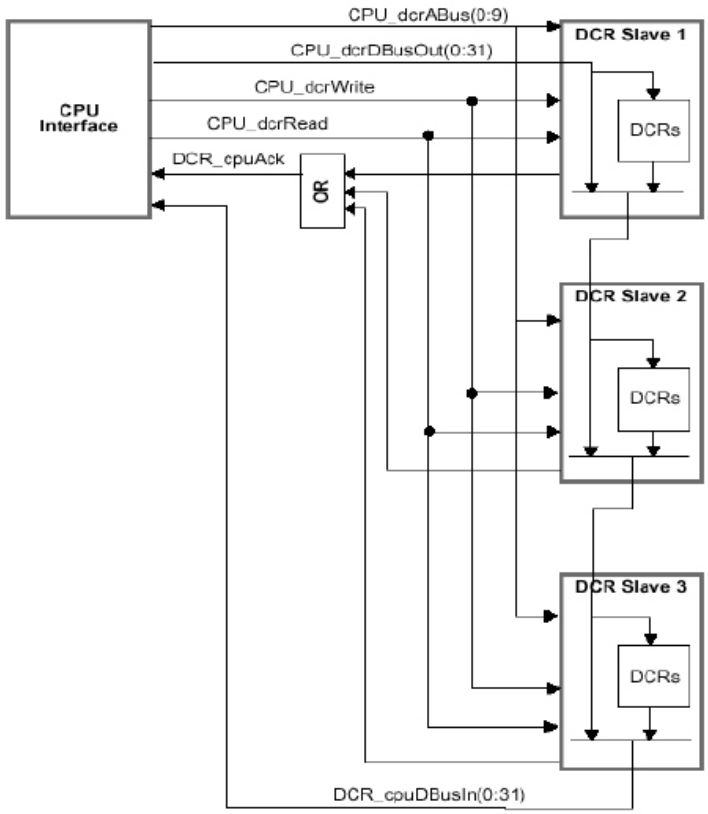


Figure 2.11 The DCR bus

Figure 2.11 shows that the DCR bus interface architecture. The DCR bus composed of a 32-bit data bus and a 10-bit address bus. The CPU interface is the only

master for the DCR bus, and all other attached devices are slaves. The DCR bus is used for on-chip device configuration purposes and it doesn't slow down high performance PLB bus. The DCR bus transfers data between CPU general purpose registers and DCR slave registers. When the DCR logic received a request, it must return DCR_cpuAck within sixty-four clock cycles or processor times out. No error occurred, and the CPU executes next instruction.

2.5 Bresenham Algorithm

Bresenham algorithm [13] can be used for scaling operation. It uses only integer subtraction, bit shifting and addition, all of which are very cheap operations in computer architectures.

The line is drawn between two points (x_0, y_0) and (x_1, y_1), as shown in Figure 2.12. Assume that the slope of the line is restricted between -1 and 0. The goal is, for each column x between x_0 and x_1, to identify the row y in that column which is closest to the line and plot a pixel at (x, y). The general formula for the line between the two points is given by: $y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$

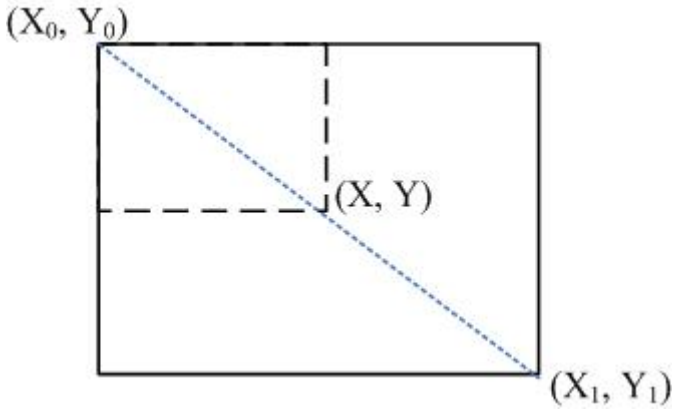


Figure 2.12 Diagram of the line drawing

Note that y starts at y0, and each time we add 1 to x, we add the fixed value $\frac{y_1 - y_0}{x_1 - x_0}$ to the exact y. Moreover, since this is the slope of the line, by assumption it is between 0 and 1. In other words, after rounding, in each column we either use the same y as in the previous column, or we add one to it. We can decide which of these to do by keeping track of an error value which denotes the vertical distance between the current y value and the exact y value of the line for the current x. Each time we increment x, we increase the error by the slope value above. Each time the error surpasses 0.5, the line has become closer to the next y value, so we add 1 to y, simultaneously decreasing the error by 1. Figure 2.13 shows the pseudo code of Bresenham algorithm. Due to the limited time, the Bresenham algorithm is not included in our implementation.

```
Bresenham line(x0, x1, y0, y1)
    boolean steep := (y1 - y0) > (x1 - x0)
    if steep then
        swap(x0, y0)
        swap(x1, y1)
    int deltax := x1 - x0
    int deltay := y1 - y0
    int error := -deltax / 2
    int y := y0
    for x from x0 to x1
        if steep then plot(y,x) else plot(x,y)
        error := error + deltay
        if error > 0 then
            y := y + 1
            error := error - deltax
```

Figure 2.13 The pseudo code of Bresenham algorithm

Chapter 3
Development Environment

3.1 Objectives

We already built a pure software video and graphics subsystem conforming to the MHP graphics model in an embedded system environment, but this approach is not fast enough to support high quality video According to the profiling data of the pure software system on the Xilinx ML310 platform, we observe a few bottlenecks, which are video scaling, video plane and graphic plane alpha composing, and screen blitting. Therefore, the objectives of this thesis are to improve the performance by including hardware logic and software-hardware-design.

The embedded system environment used for deployment is described in the following sections.

3.2 Target Platform and Development Tools

The Xilinx ML403 platform is chosen as the target environment to emulate a DTV set-top box. The hardware and software configurations are described in the following sections.

3.2.1 Xilinx ML403 Platform

The Xilinx ML403 evaluation platform [4] [5] enables designers to develop embedded system and system-on-chip applications. The block diagram of the ML403 platform is shown in Figure 3.1.

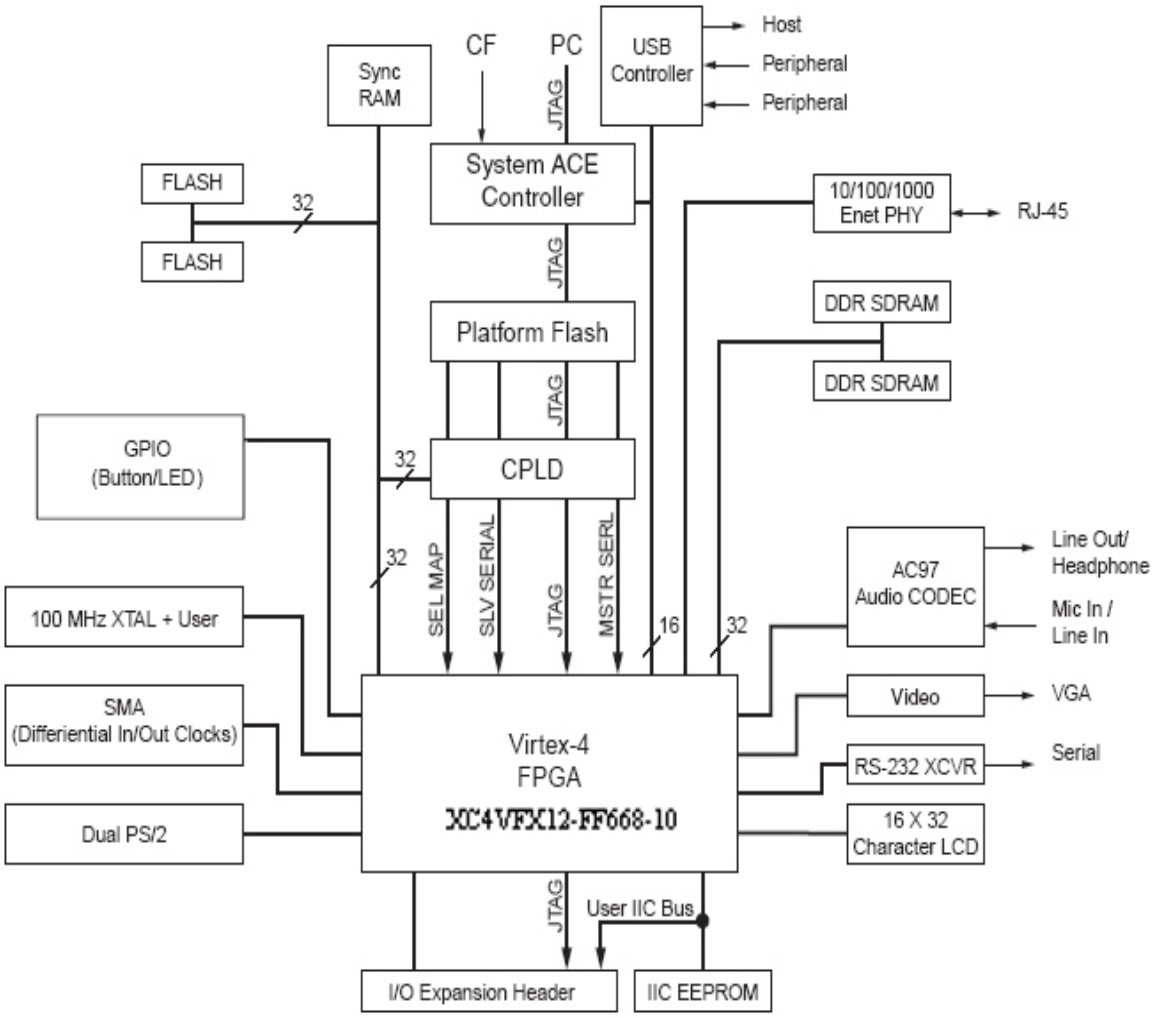


Figure 3.1 Xilinx ML403 platform block diagram
[4, Figure 1, pp.11]

The Virtex-4 XC4VFX12-FF668 FPGA has one embedded PowerPC 405 processor clocked at 300 MHz maximum. There are 648 Kb block RAM on the FPGA. Data accesses between the processor and block RAM have better performance than that between processor and memory on the bus. The ML403 platform has 64 MB DDR SDRAM, 32-bit interface running up to 266-MHz data rate. The ML403 platform also has 8 Mb ZBT synchronous SRAM on 32-bit data bus with no parity bits. The CompactFlash card contains a DOS FAT16 filesystem partition and a Linux EXT3 filesystem partition. The DOS filesystem partition contains a set of ACE files to run board diagnostics, as well as to demonstrate the operation of various operating systems such as MontaVista Linux, VxWorks, and QNX.

3.2.2 Embedded Development Kit (EDK)

Embedded Development Kit (*EDK*) provides a suite of design tools to help us to design a complete embedded processor system in a Xilinx FPGA device. [7][8] Though the *Xilinx Integrated Software Environment* (*ISE*) must be installed along with EDK, it is possible to create our entire design from beginning to end in the EDK environment.

EDK includes several tools, which are:

- Xilinx Platform Studio (XPS): An integrated design environment in which we can create our complete embedded design.
- The Base System Builder (BSB) Wizard: Allows us to create a working embedded design quickly, using any features of a supported development board or using the basic functionality common to most embedded systems.
- Platform Generator (Platgen): Constructs the programmable system on a chip in the form of HDL and implementation netlist files.
- Xilinx Microprocessor Debugger (XMD): Open a shell for software download and debugging.
- System ACE File Generator (GenACE): Generates a Xilinx System ACE configuration file based on the FPGA configuration bitstream and software executable to be stored in a non-volatile device in a production system.

In addition to the aforementioned tools, in order to development a custom system, EDK also offers the other tools such as “EDK Command Line” to allow us to run the embedded design flows or to change tool options by using a command, “Library Generator”, to construct a software platform comprising a customized collection of software libraries, drivers and OS, “GCC” and others.

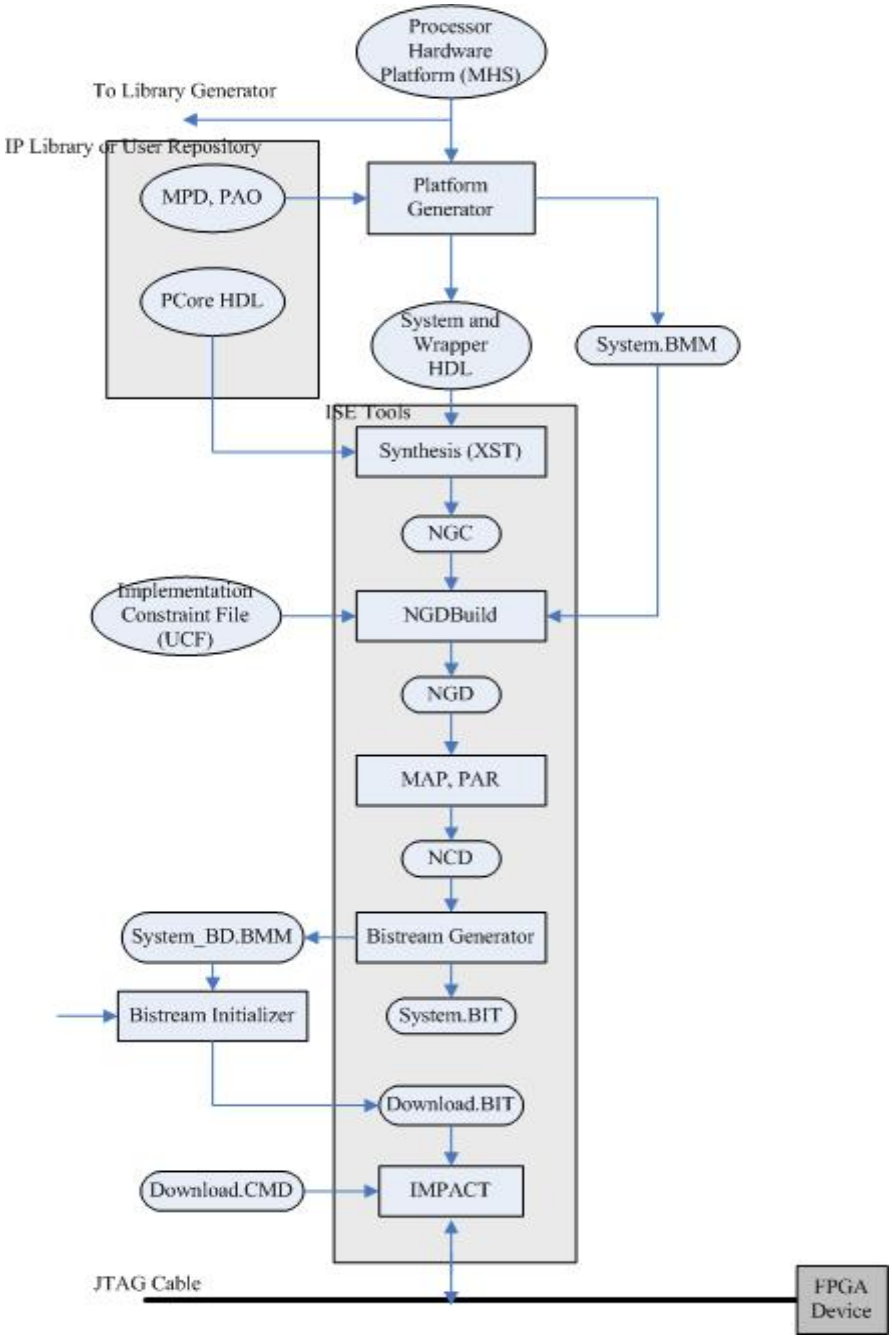


Figure 3.2 EDK Hardware Tools Architecture
[7, Figure 1-2, pp.26]

Figure 3.2 shows that how to create system bitstream and how to download the bitstream to the FPGA device.

When generating the system, the platform generator (Platgen) imports the MHS (Microprocessor Hardware Specification) file, the MPD (Microprocessor Peripheral Definition) and PAO (Peripheral Analyze Order) files of all peripherals to the system. The MHS file defines the configuration of the embedded processor system including

buses, peripherals, processors, connectivity, and address space. The MPD file defines the interface for the peripheral, and the PAO file tells other tools (Platgen, Simgen, for example) which HDL files are required for compilation (synthesis or simulation) for the peripheral and in what order. MPD and PAO files are automatically generated when we create or import a peripheral. Then Platgen will generate the system and wrapper HDL files and the system.BMM file. The BMM (BlockRAM Memory Map) file contains a syntactic description of how individual BlockRAMs constitute a contiguous logical data space. Then the system and wrapper HDL will be fed to the ISE tools such like XST, NGDBuild and so on. Finally, the iMPACT tool will download the bitstream file to the FPGA device by the JTAG cable.

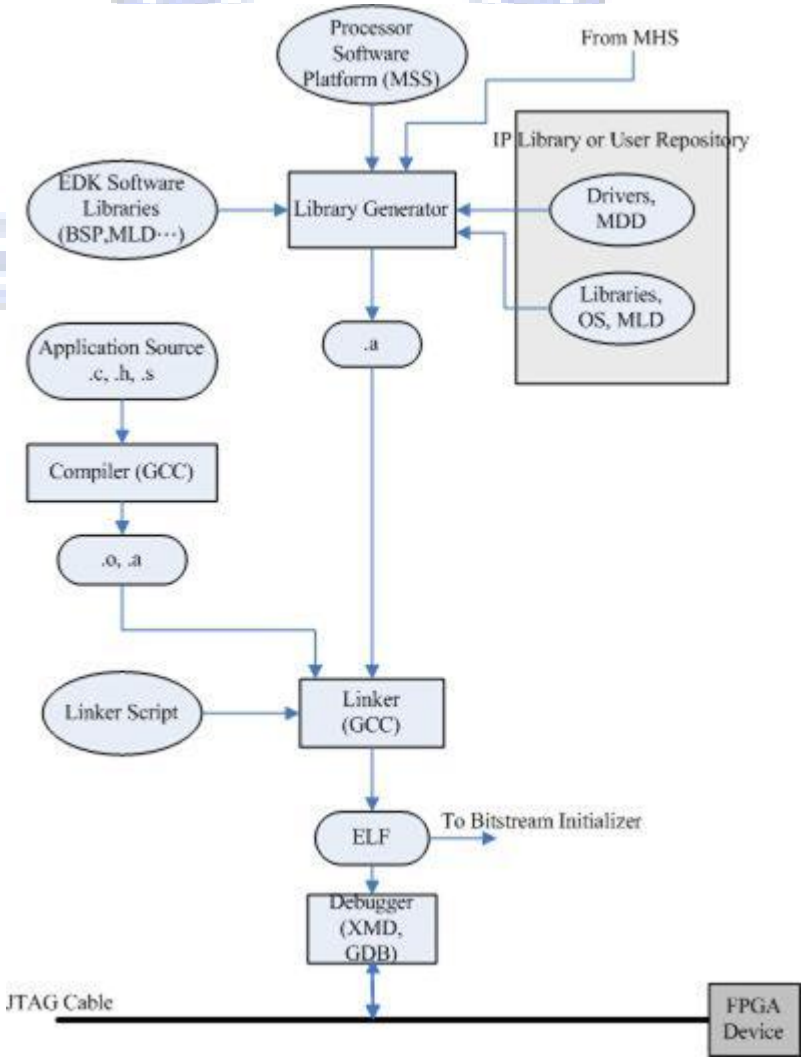


Figure 3.3 EDK Software Tools Architecture
[7, Figure 1-2, pp.26]

Figure 3.7 shows that how to create application *ELF* (*Executable Linked Format file*) file and how to download the ELF file to the FPGA device.

Before creating a application ELF file, we must generate the corresponding library first. The library generator will import the EDK software library file such like *BSP* (*Board Support Package*) and *MLD* (*Microprocessor Library Definition*) files and IP library or user repository (*MDD* (*Microprocessor Driver Description*), MLD) files. EDK provides some software libraries to help us to develop the embedded system easily. These libraries include “LibXil File” which provides block access to file systems and devices using standard calls such as open, close, read and write, “LibXilNet” which includes functions to support the TCP/IP stack and the higher level application programming interface (Socket APIs), “XilFatfs FATfile” which provides read/write access to files stored on a System ACE compact flash, and “LibXil MFS” which provides function for the memory file system and can be accessed directly or through the LinXil File module. IP library or user repository will be created if the hardware IP is set to include some standard function such as DMA or address range when it was created. When the ELF file is created, the debugger tool (XMD or GDB) will download the file to the FPGA device by the JTAG cable.

3.2.3 Block RAM

The Virtex-4 XC4VFX12-FF668 FPGA has 36 pieces of block RAM and each block RAM stores 18K bits of data. [6] Write and Read are synchronous operations; the two ports are symmetrical and totally independent, sharing only the stored data. Each port can be configured in any “aspect ration” from 16Kx1, 8Kx2, to 512x36, and the two ports are independent even in this regard. Figure 3.2 illustrates the dual-port data flow. Table 3.1 lists the port names and descriptions.

A read/write operation uses one clock edge. When reading, the read address is registered on the read port and the stored data is loaded into the output latched after the RAM access time. Similarly, the write address is registered on the write port, and

the data input is stored in memory when writing.

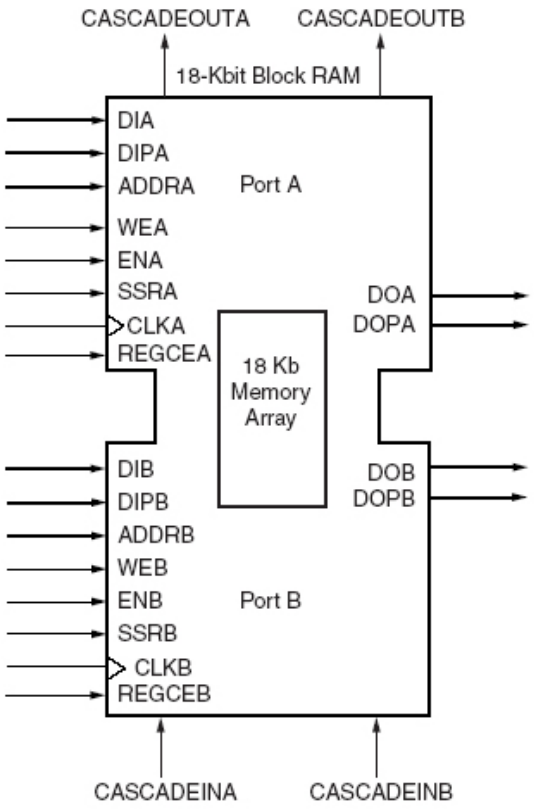


Figure 3.4 Dual-Port Data Flow
[6, Figure 4-1, pp.111]

Port Name	Description
DI[A | B]	Data Input Bus
DIP[A | B]	Data Input Parity Bus
ADDR[A | B]	Address Bus
WE[A | B]	Write Enable
EN[A | B]	When inactive no data is written to the block RAM and the output bus remains in its previous state.
SSR[A | B]	Set/Reset
CLK[A | B]	Clock Input
DO[A | B]	Data Output Bus
DOP[A | B](1)	Data Output Parity Bus
REGCE[A | B]	Output Register Enable
CASCADEIN[A | B]	Cascade input pin for 32K x 1 mode
CASCADEOUT[A | B]	Cascade output pin for 32K x 1 mode

Table 3.1 Port names and descriptions of the BRAM

Besides the basic read and write operations, there are three modes of a write operation, which are WRITE_FIRST, READ_FIRST, and NO_CHANGE. The default mode is WRITE_FIRST mode and mode is set during device configuration. These choices increase the efficiency of block RAM memory at each clock cycle.

These three modes are different on the data available on the output latches after a write clock edge.

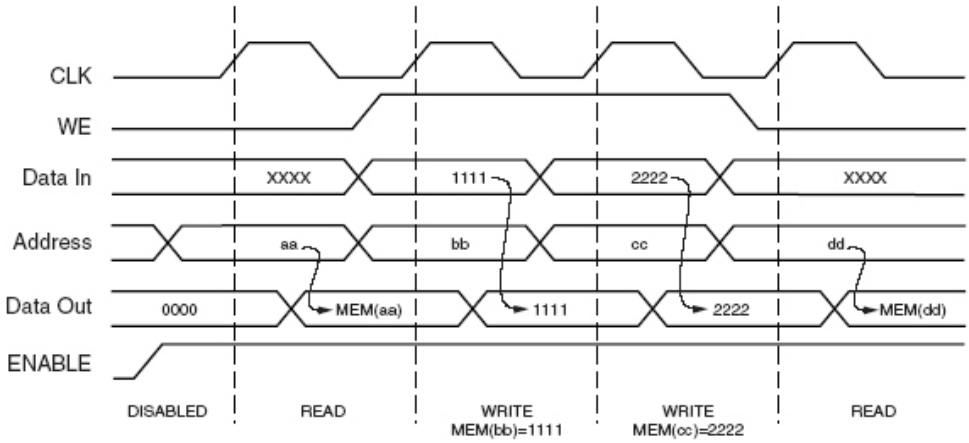


Figure 3.5 WRITE_FIRST mode waveform
[6, Figure 4-2, pp.112]

As shown in Figure 3.3, in WRITE_FIRST mode, the input data is simultaneously written into memory and stored in the data output (transparent write).

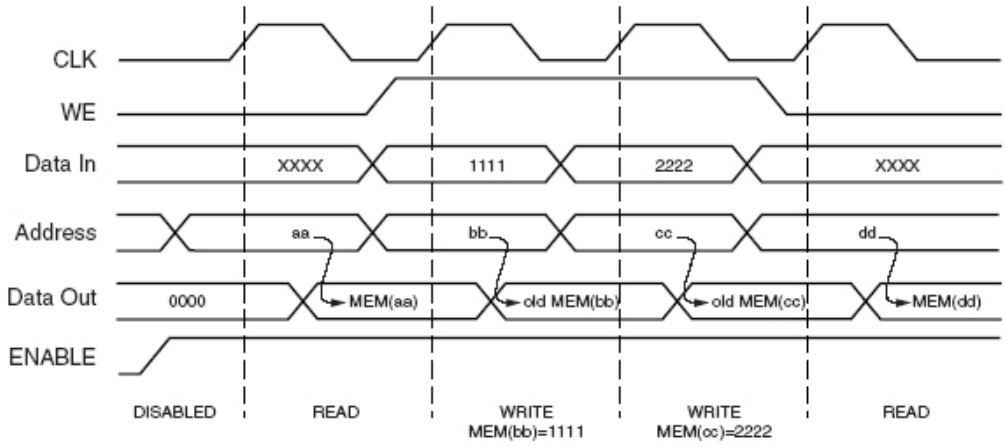


Figure 3.6 READ_FIRST mode waveform
[6, Figure 4-3, pp.113]

Figure 3.4 shows that in READ_FIRST mode, data previously stored at the write address appears on the output latches, while the input data is being stored in memory (read before write).

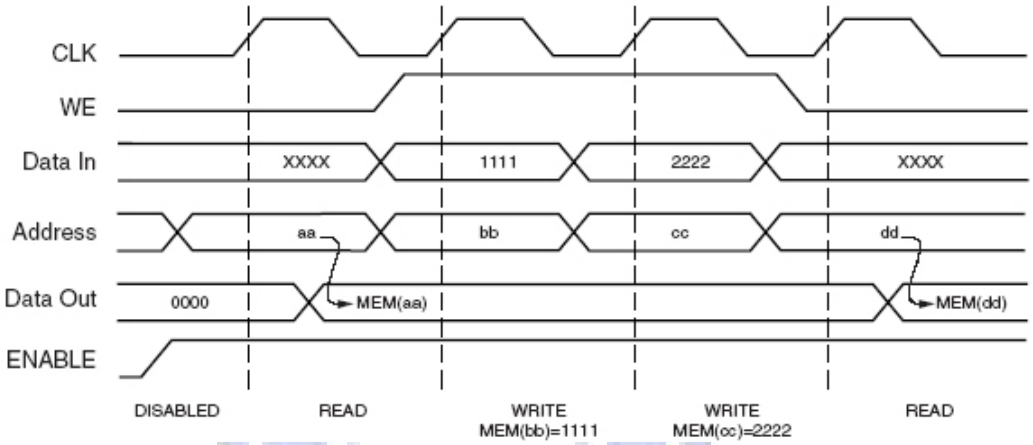


Figure 3.7 NO_CHANGE mode waveform
[6, Figure 4-4, pp.113]

As shown in Figure 3.5, in NO_CHANGE mode, the output latches remain unchanged during a write operation.

Because the block RAM is a true dual-port RAM where both ports can access any memory at any time. When accessing the same memory location from both ports, we must be very carefully to avoid conflict.

3.2.4 Intellectual Property InterFace (IPIF)

The IPIF is a module that enables us to easily integrate our own custom application specific value-added IP into CoreConnect based systems. [9][10] Figure 3.8 shows the IPIF positioned between the bus and the user IP. The interface seen by the IP is called the IP Interconnect (IPIC). The combination of the IPIF and the user IP is called a device (or a peripheral). Figure 3.8 also shows that the services provided by the IPIF. In addition to facilitating bus attachment, other services, FIFOs, DMA, Scatter Gather (automated DMA), software reset, interrupt support and bus-master access, are places in the IPIF to standardize functionality that is common to many IPs and to reduce IP development effort.

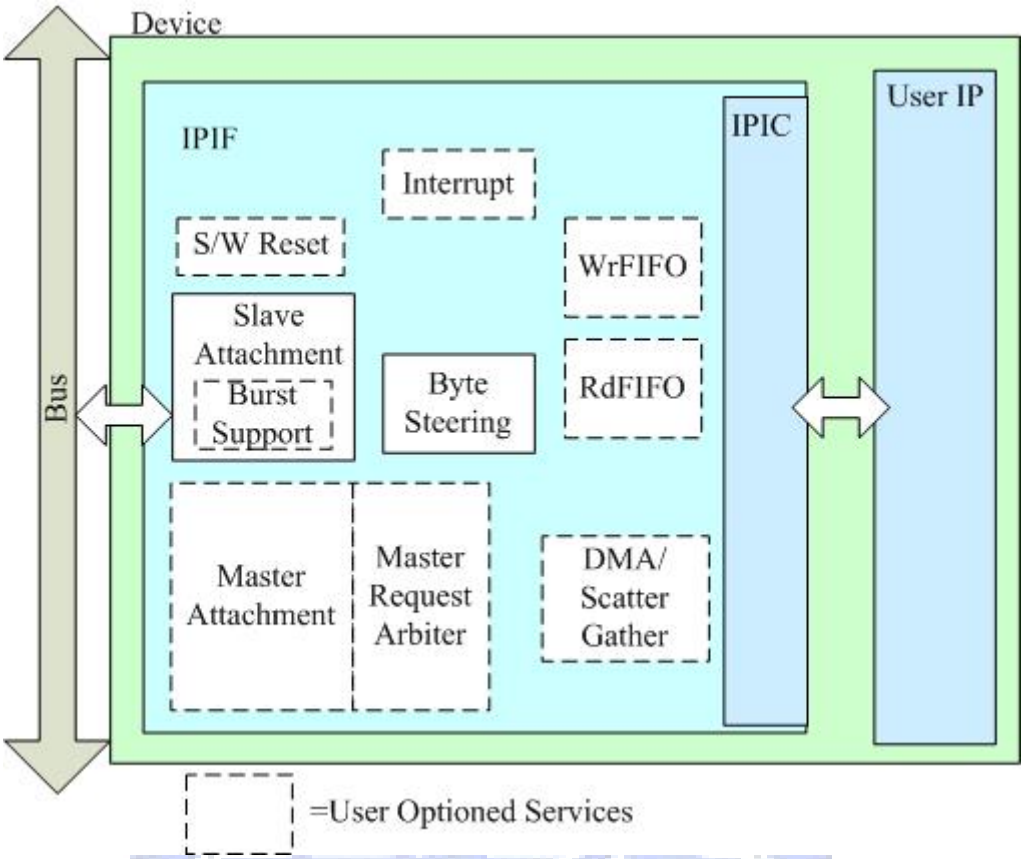


Figure 3.8 IPIF block diagram
[9, Figure 1, pp.2]

The base element of the design is the Slave Attachment. This block provides the basic functionality for Slave operation. It implements the protocol and timing translation between the Bus and the IPIC. Optionally, the Slave Attachment can be enhanced with burst transfer support. This feature provides higher data transfer rates for sequential-address accesses.

The Byte Steering function is responsible for steering data bus information onto the correct byte lanes. This block supports both read and write directions and is required when a target address space in the IPIF or the User IP has a data width smaller than the bus width (32 bits for OPB and 64 bits for PLB).

The User may use a Master Attachment Service. This is needed when the User IP needs to access the OPB as a Master device.

The IPIF provides an optional Direct Memory Access (DMA) Service. This

service automates the movement of large amounts of data between the user IP or IPIF FIFOs and other peripherals (such as System Memory or a Bridge device). The inclusion of the DMA Service automatically includes the Master Attachment Service. The DMA Service requires an access to the bus as a Master device.

Chapter 4
Hardware Graphics Accelerator and System Integration

According to the software system profiling data, the video scaling, alpha composing, and screen blitting operations are the bottlenecks of the pure software system on the ML403 platform. We plan to design and implement the hardware accelerators for these three parts to improve the performance.

The pure software implementation is deployed by many tools, including TinyX, Microwindows-based AWT, JMF, FFMpeg, and others. TinyX is an X server implementation of X Window System included in the Monta Vista Linux. Microwindows is an open-source windowing system for resource-constrained devices. JMF is an API for incorporating time-based media into Java applications. FFmpeg is a set of open-source and cross-platform libraries for multimedia applications.

The original AWT does not provide the stacked planes such as background plane, video plane, and graphics plane. In order to present video together with AWT graphics and to alpha-blend the graphics over the video at the same time, the modified graphics model as shown in Figure 4.1 is designed.

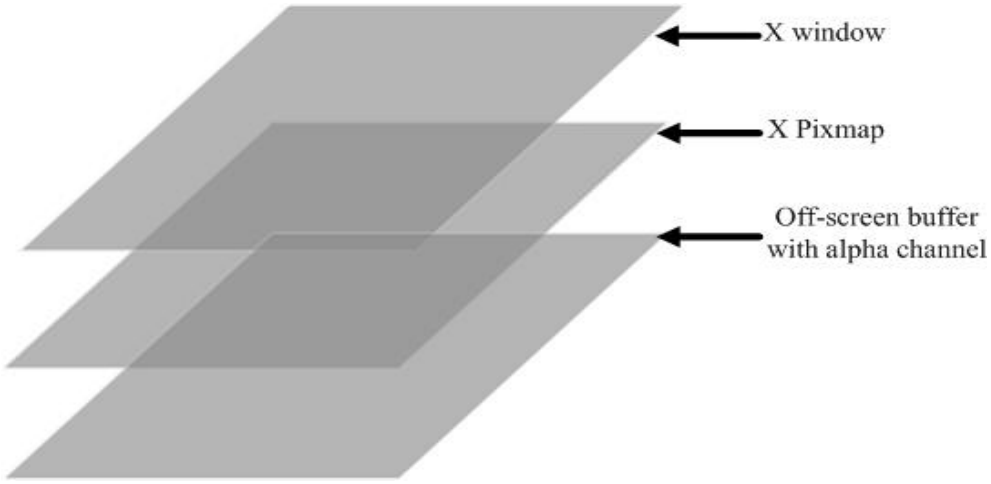


Figure 4.1 Modified graphics architecture

The off-screen buffer plays the role of the graphics plane in the MHP graphics model. The X Pixmap is designed to be both the video plane and the composition buffer of the video plane and the graphics plane. The X Window represents the final display device. The composite result is first drawn on the X Pixmap and then is blitted to the X Window to avoid the pixel-by-pixel flickers.

The third part of bottlenecks, screen blitting, is caused by X Pixmap to X Window blitting. Because the whole system is constructed on the TinyX, this part of performance loss is unavoidable.

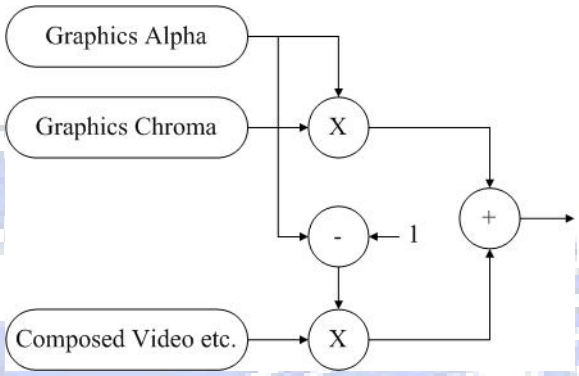
In order to accelerate the system's operating speed, we design the alpha composing hardware logic and the scaling hardware logic, and then integrate those logics circuits into the entire system to replace their software counterparts.

4.1 Alpha Composition Circuits

4.1.1 Design and Implementation

In our description in Section 2.3, the graphics components are first composed,

and then the graphics plane (component plane) is combined with the video plane (background plane). The compositions between components occur only in the overlap regions, but the compositions between component plane and background occur on the whole screen. Since the component and background composition takes much more computing time than the other operations, the alpha composition logic is designed to speedup that task. The component plane and the background plane are composed together using the SRC_OVER rule as shown in Figure 4.2.



$$\text{Result} = \text{graphics alpha} \times \text{graphics chroma} + (1 - \text{graphics alpha}) \times \text{composed video}$$

Figure 4.2 Graphics and video composition

Figure 4.3 shows the procedure of composition. There is a data buffer in the composition logic and a few control registers to control the logic or to record the state of logic. We first write one row of graphics data and video data to the data buffer. Then write the specific number “0x0A” to the start control register to enable the composition logic to start. First, the graphic and video data are composed pixel by pixel, then write back the results to the data buffer. Finally, we write the results back into the data buffer at the destination.

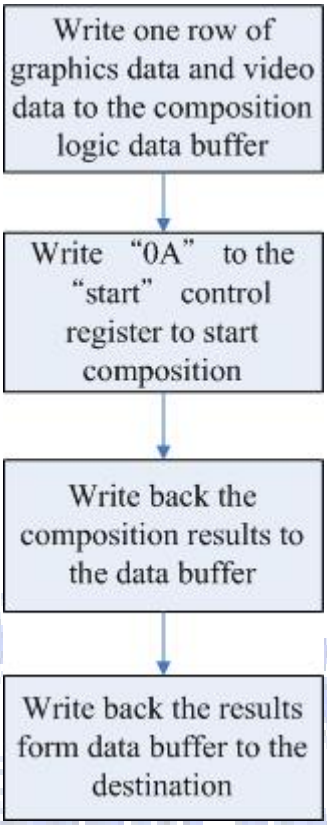


Figure 4.3 The procedure of composition logic

First of all, when we create a new peripheral, we need to choose types of buses, including OPB, PLB and fast simplex link, which are connected with the new peripheral. The PLB bus is chosen to be connected with the IP because of its higher performance. Next, for the EDK graphics user interface, we can choose the optional IPIF (Intellectual Property InterFace) services, including S/W reset and the module information register, burst transaction support, DMA, FIFO, interrupt support, S/W register support, master support and address range support. We include the DMA service to accelerate data accessing, the S/W register service to be control registers and the address range service to support Block RAM interface. Figure 4.4 shows the connections between IPIF and user logic when we include the optional functions in the above.

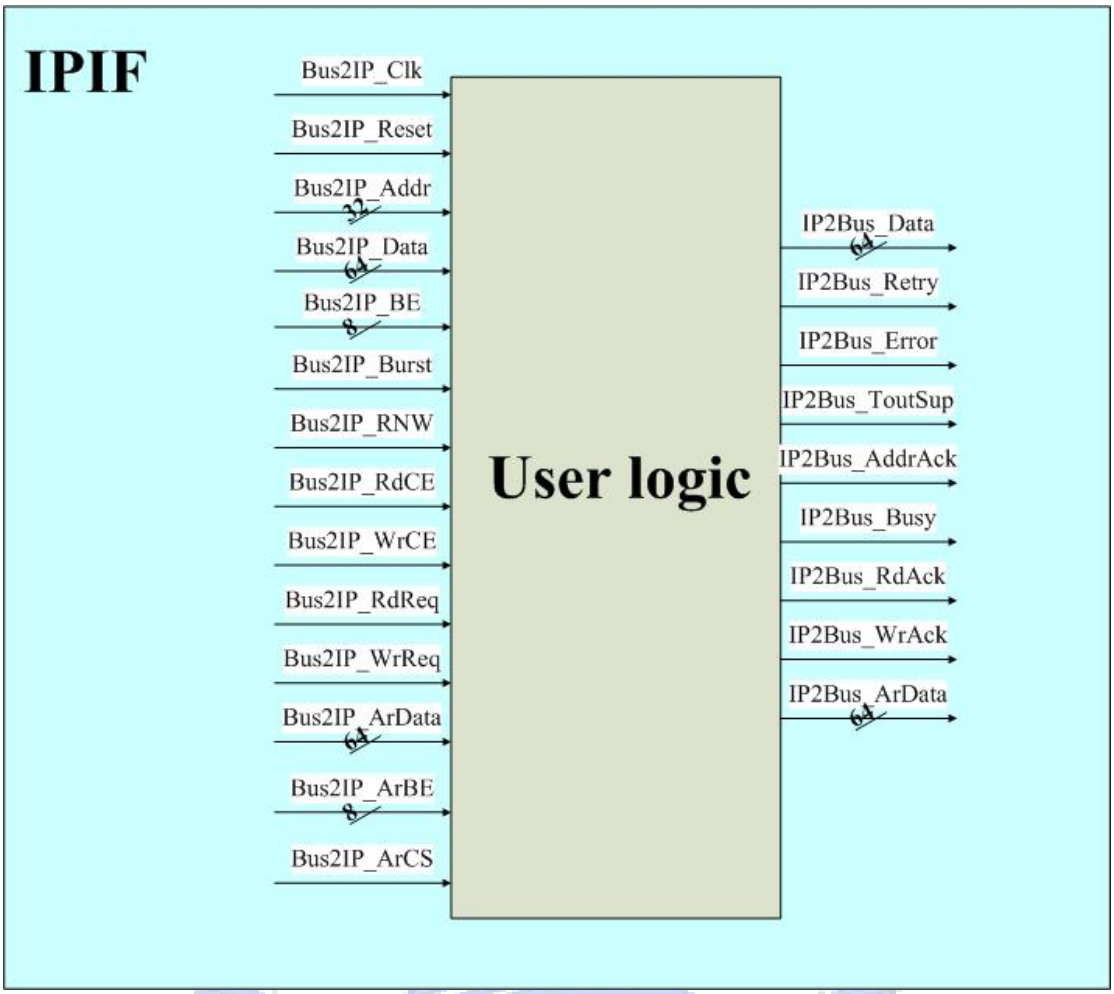


Figure 4.4 The connections between IPIF and user logic

The purpose of the Bus2IP_BE port is to support byte-by-byte data access. The Bus2IP_RdCE and Bus2IP_WrCE port can be one to 32 bits width and the register addressing is implemented by them. For example, if we want the IPIF to support four registers, the widths of Bus2IP_RdCE and Bus2IP_WrCE port will be four. And if we want to write data to the third register (memory map to BASEADDR+3×BusDataWidth), the Bus2IP_WrCE will be “0010”. The purpose of Bus2IP_ArData, Bus2IP_ArBE, Bus2IP_ArCS and IP2Bus_ArData is to support the address range service. The IPIF supports eight address ranges at the maximum.

We use eight pieces of dual-port block RAM to act as data buffer, and each piece of block RAM is configured in 8-bit×2k-deep. The EDK maps the beginning of the

data buffer to the virtual memory address: XPAR_USER_LOGIC_0_AR0_ADDR. Figure 4.5 shows the data buffer in the composition logic. Since the PLB bus is 64-bits width, the eight pieces of block RAM are set in one row so that the memory mapping is easier. For example, the virtual memories 0x10010010 to 0x1001001F are mapped to block RAM address ”000_0000_0001”, a mapping of the 17th bit to 27th bit of the virtual memories address. And the data access byte by byte can be disposed by Bus2IP_ArBE signal. If we want to write 32-bit data to the virtual memory address 0x10010110, the block RAM address will be “000_0001_0001”and the Bus2IP_ArBE will be “1111_0000”. Because the full screen size is 640×480 and the composition operation is executed row by row, we only need 640×4 bytes buffer space for each of graphic and video planes. Therefore, the first 512 deep of the data buffer is allocated to graphic, the next 512 deep of the data buffer is designed for video and the rest of the data buffer is allocated to the composition results.

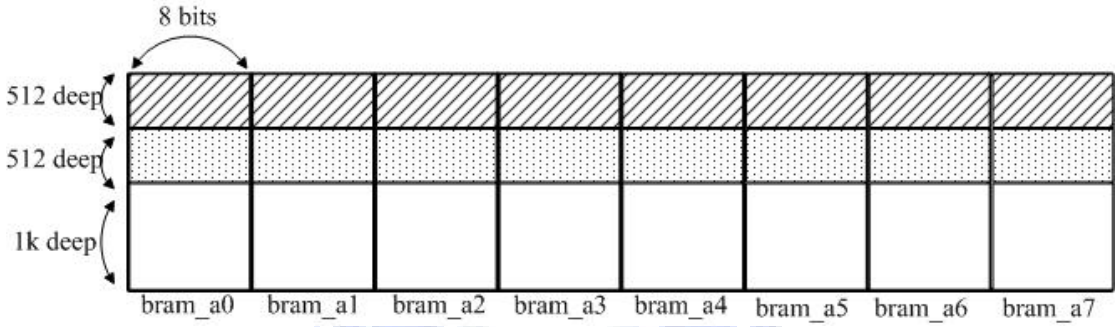


Figure 4.5 Data buffer in the composition logic

After writing one row of graphics and video data to the data buffer of the composition logic, we must write a few parameters to the registers to control the behavior of logic and to trigger the logic to start.

The EDK maps the beginning of the register to the virtual memory address XPAR_USER_LOGIC_0_ADDR. Figure 4.6 shows these control registers in the composition logic. The first eight bits is the start control register. It waits until all other data and parameters are prepared. Then it has to be written with a specific value,

0x0A. The start register is refreshed to 0x00, when the logic is ready to work. If we write any other value but not 0x0A, the start register will simply keep the value and the composition logic will not start. The bit-8 is the busy flag register and the bit-9 is the done flag register. We observe that the single row composition is done or not though these two state registers. These two state registers are initialized to “00”. When starting, they are changed to “10” and they are changed to “10” when a single row composition is finished. The bit-10 to bit-21 are empty, reserved for future expansion. The next ten bits, bit-22 to bit -31 is the width control register that tells the logic how many pixels have to be disposed.

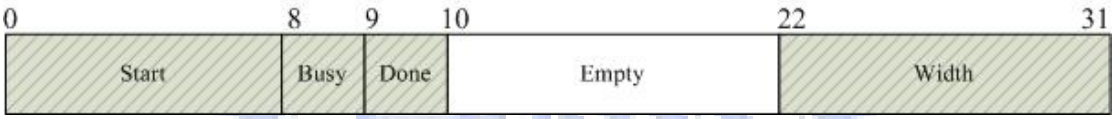


Figure 4.6 Control registers of the composition logic

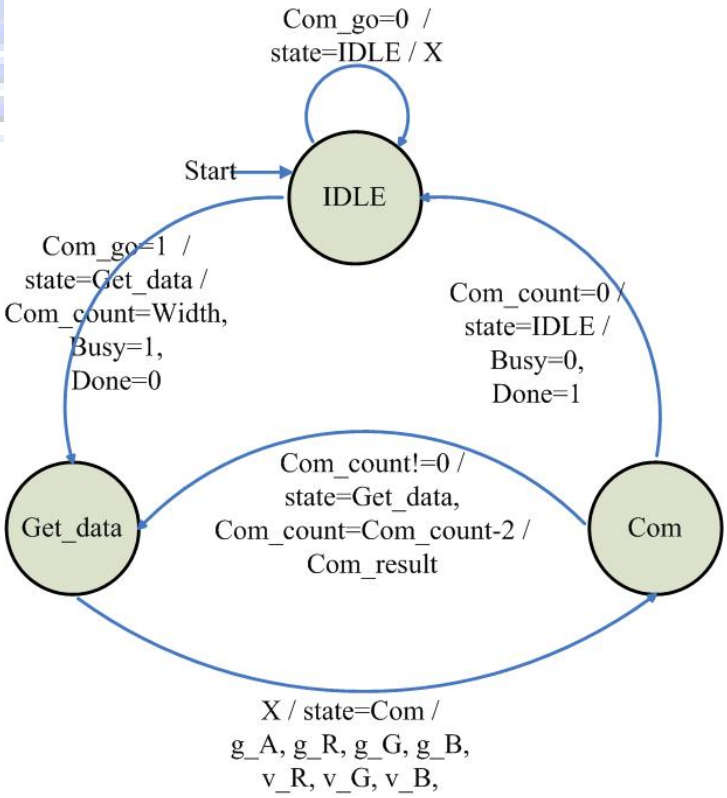


Figure 4.7 The FSMD model of the composition logic

Figure 4.7 shows the finite-state machine with datapath model of the composition logic. When the logic is reset, it is set to the “IDEL” state. The state of the logic is not changed until the start register is written “0x0A” and it makes the signal, Com_go, to be high. When Com_go becomes high, the logic moves to the “Get_data” state to get data from the data buffer, Busy is changed to high, Done is changed to low, and Com_count is initialized to Width, the value contained in the Width control register. Since the data buffer is implemented by the dual-port block RAM, we can get both data of graphics and video at the same time as shown in Figure 4.8. We send address to the block RAM address port immediately after entering the “Get_data” state, and in the next two cycles, we get the A, R, G and B data of graphics and video from the data out port. In the meanwhile, the state is changed to “COM”.

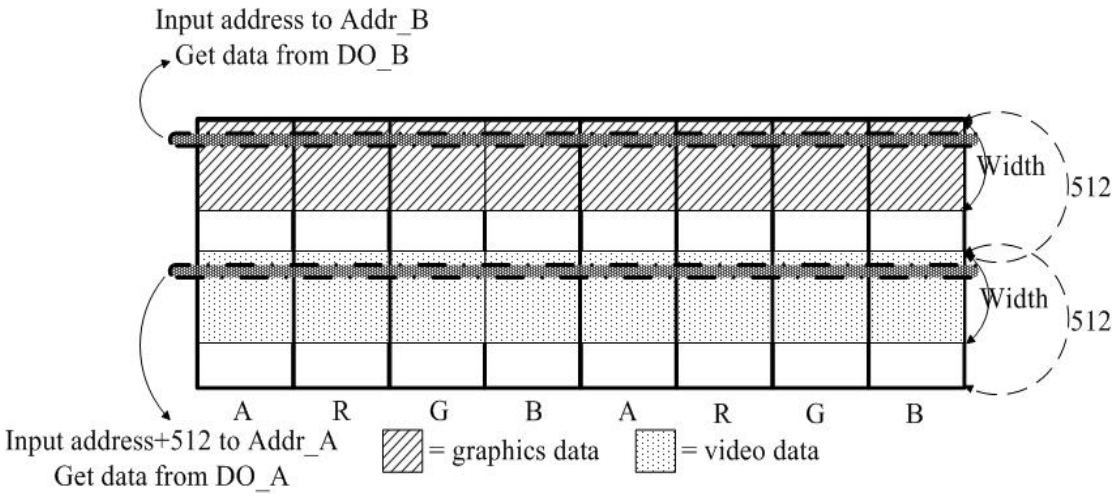


Figure 4.8 Get data from the data buffer

The “COM” state takes only one cycle time. Its behavior is computing the composition results and writing back to the data buffer immediately. The composition operation used the SRC_OVER rule described earlier. In general, it needs multiplication. We simply implement the basic three-level composition, since the

standard requires that the MHP terminals must implement at least three levels of transparency: 0% (opaque), 100% (completely transparent) and an intermediate value of approximately 30% and the implementation of additional intermediate levels of transparency is optional. That is, if the graphics alpha value is 0 or 255, the result is the original video value or graphics value, and if the graphics alpha is of any other value, we treat the alpha value as 30%. Therefore, we can use the bit-shift operation to approximate the SRC_OVER rule behavior as described below.

$$0.3 \cong 2^{-2} + 2^{-5} + 2^{-6}$$
$$0.7 \cong 2^{-1} + 2^{-3} + 2^{-4} + 2^{-6}$$
$$\begin{aligned} \text{result} &= \text{g\_value} \times 0.3 + \text{v\_value} \times 0.7 \\ &\approx \text{g\_value} >> 2 + \text{g\_value} >> 5 + \text{g\_value} >> 6 + \\ &\quad \text{v\_value} >> 1 + \text{v\_value} >> 3 + \text{v\_value} >> 4 + \text{v\_value} >> 6 \end{aligned}$$

4.1.2 Standalone Subsystem

Before integrating the composition logic into the existing pure software system, we implement a standalone subsystem and evaluate its execution time performance.

We download the ML403 reference design from the Xilinx home page and connect the composition logic to the PLB bus.

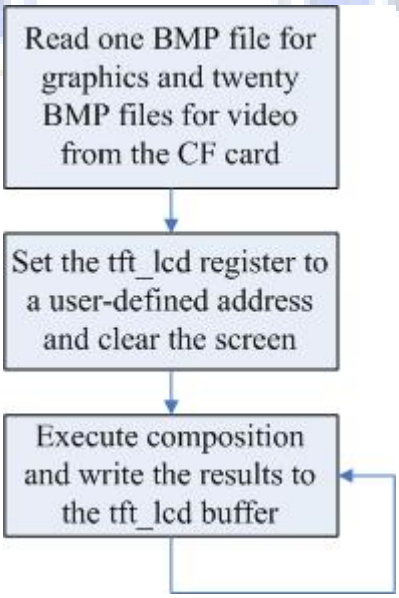


Figure 4.9 The procedure of the composition standalone subsystem

Figure 4.9 shows the procedure of the composition standalone subsystem. We first read the BMP pictures from the Compact Flash card: one picture with alpha values viewed as graphic data and twenty consecutive pictures viewed as video data. The reason why we read all the files to the memory at the beginning is to avoid the long reading time of accessing Compact Flash card. In order to access data from the Compact Flash card, we must include the “XiFatfs” filesystem library in the EDK software setting GUI, and then we can use functions such as “sysace_fopen”, “sysace_fread” and others. There is a TFT_LCD controller in the ML403 reference design. We reserve a piece of memory as the screen buffer. We write the address of this buffer to the TFT-LCD controller, and then it will display the buffer contents to the LCD screen, which resolution is 640×480. Involving the synchronization problem, the screen buffer must be larger than the full screen size, 640×480, and it is 1024×768 in practice as shown in Figure 4.10.

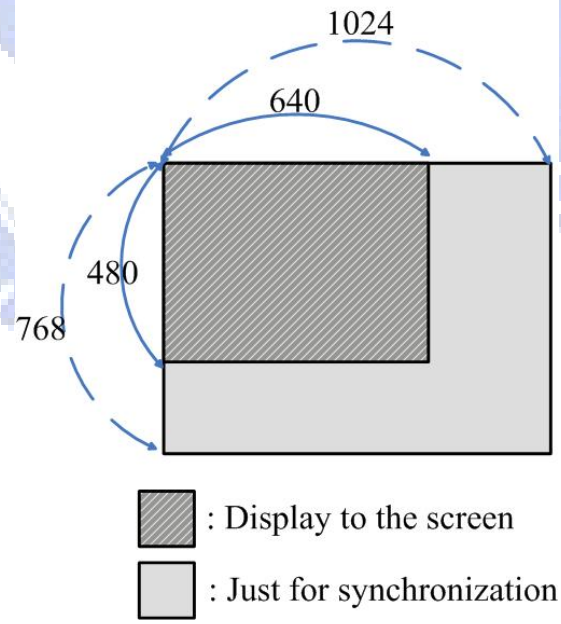


Figure 4.10 The TFT LCD screen buffer

Then, we use a loop program to compose the twenty consecutive video pictures and the graphics picture recursively and write the composition results to the screen buffer. In this way, we will see a short but smooth clip on the screen.

4.1.3 Standalone Subsystem Performance

There is a 64-bit time base counter inside the PowerPC core. The counter increases by one at every processor cycle. In the standalone subsystem, we use the 'XTime' library function, which can access the time counter, to profile the performance.

We run a 18000-frame loop, and compare the composition execution cycles of the software and hardware cases. The PowerPC core is running at 300MHz.

The software optimization level is set to be maximum, and the cache is enabled. The software composition also supports the basic three-level and uses the shift operation to replace multiplication, which is the same as the operation of the hardware composition. Table 4.1 shows the profiling results. It is clear that the hardware version increased the speed by about 50%

	Hardware composition	Software composition
Execution cycles	163399354063	279323403330
Frames / second	34.65	20.27

Table 4.1 The profiling of standalone composition

4.2 Scaling Circuits

According to the MHP specifications, the decoded video is upsampled to the full screen size, and then the clipping and scaling operations are performed against the full-screen upsampling video. Since this approach requires an upsampling operation and a scaling operation which are very time-consuming, we proposed a modified approach in the pure software MHP implementation. Figure 4.11 shows the modified

approach, which produces a pseudo full-screen video frame instead of a physical one. The clipping and scaling operations are performed directly against the decoded video frame with an appropriate coordinate transformation. The MHP applications continue to make requests of clipping and scaling to the pseudo full-screen video frame because the coordinate transformation is transparent to the MHP API.

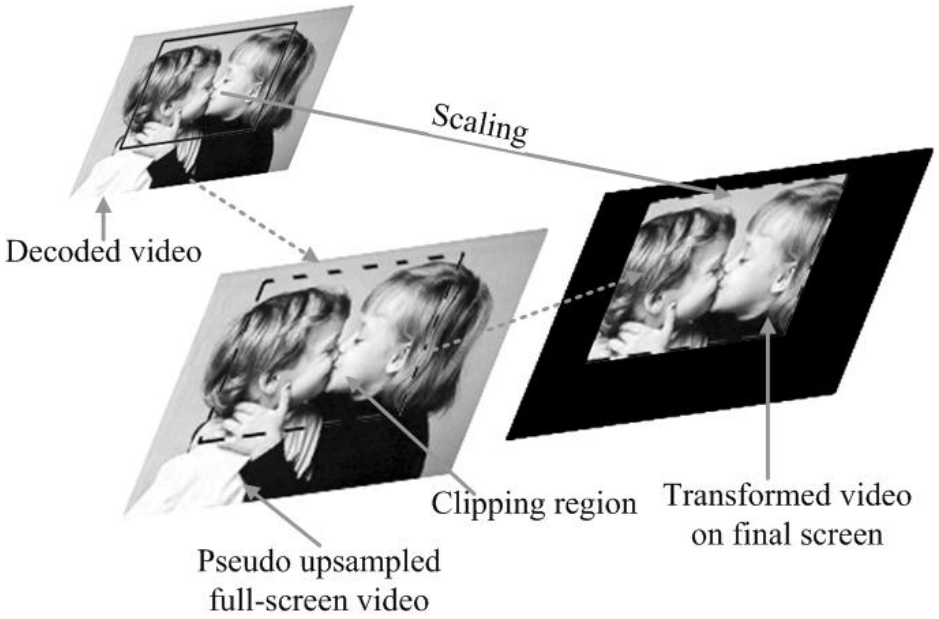


Figure 4.11 Process of the modified video format conversion

Figure 4.12 illustrates the coordinate transformation. The pixel $(X1, Y1)$ in the transformed video on the final screen is projected from the pixel $(X2', Y2')$ in the decoded video using the following transformation:

$$\begin{cases} X2' = \left(Xs + (X1\text{-}Xd) \times \frac{Ws}{Wd} \right) \times \frac{Wo}{Wf} \\ Y2' = \left(Ys + (Y1\text{-}Yd) \times \frac{Hs}{Hd} \right) \times \frac{Ho}{Hf} \end{cases}$$

In the thesis, the clipping operation is still performed by software, but the clipping region is scaled by the hardware accelerator.

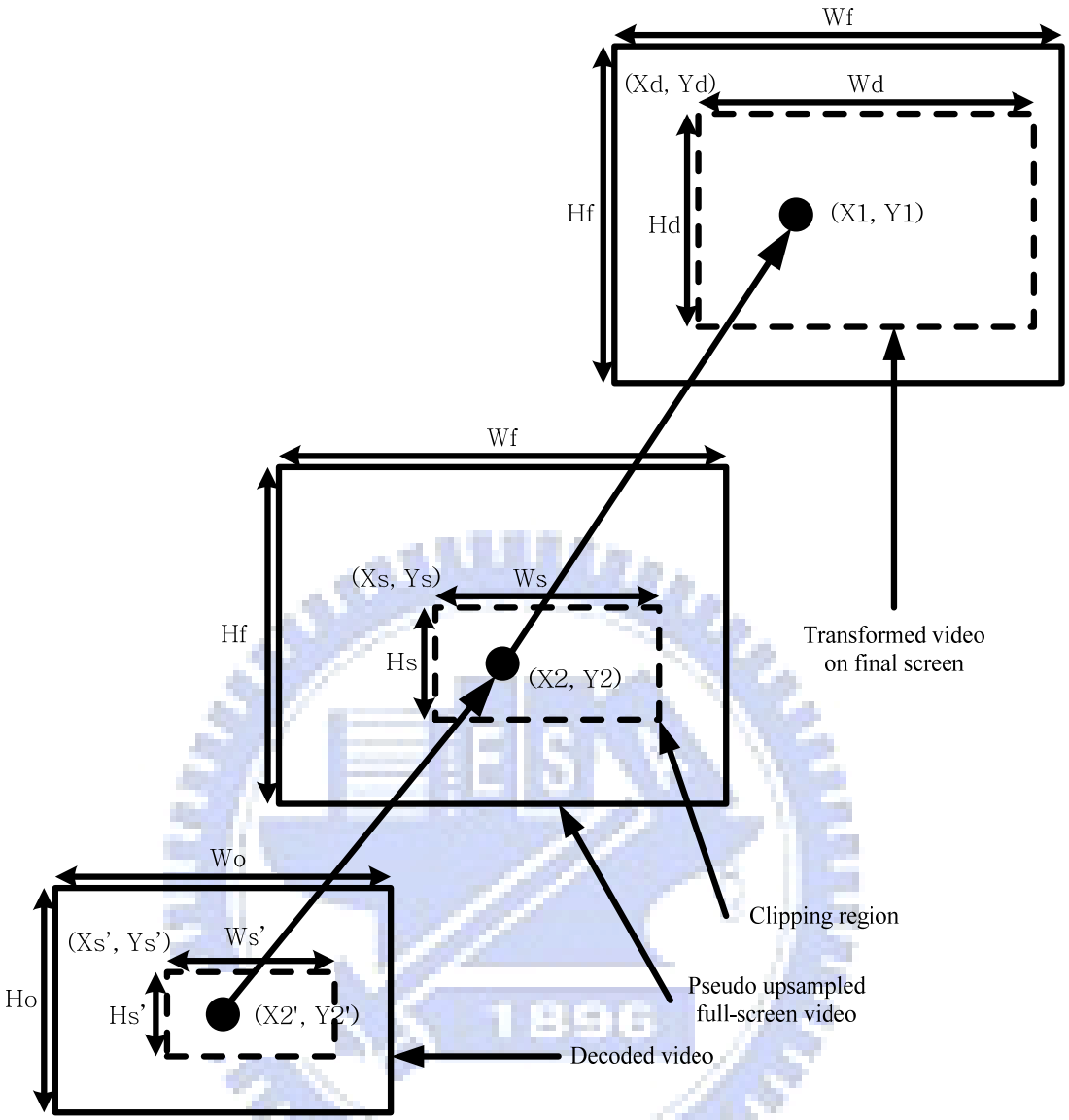


Figure 4.12 Coordinate transformation in the modified video format conversion

4.2.1 Design and Implementation

The structure of scaling circuits is similar to the structure of composition circuits, including the optional functions of IPIF, the configuration of data buffer, and the execution follow. Figure 4.13 shows the procedure of the scaling circuits. We execute the scaling operation row by row.

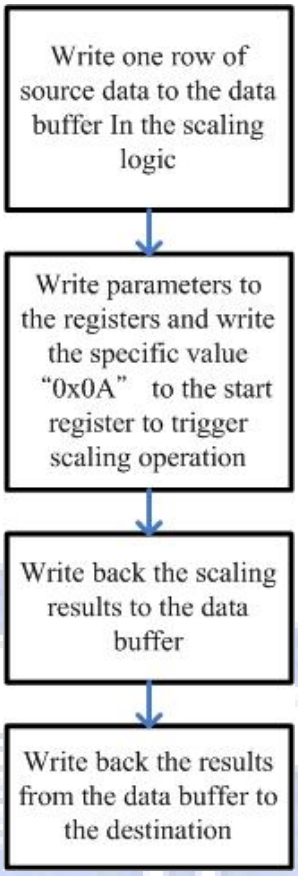


Figure 4.13 The procedure of the scaling logic

First, we write one row of source data to the data buffer in the scaling circuits. The data buffer is configured by eight pieces of block BRAM, and each piece is configured in 8-bit×2k-deep, similar to the configuration of composition circuits as shown in Figure 4.14.

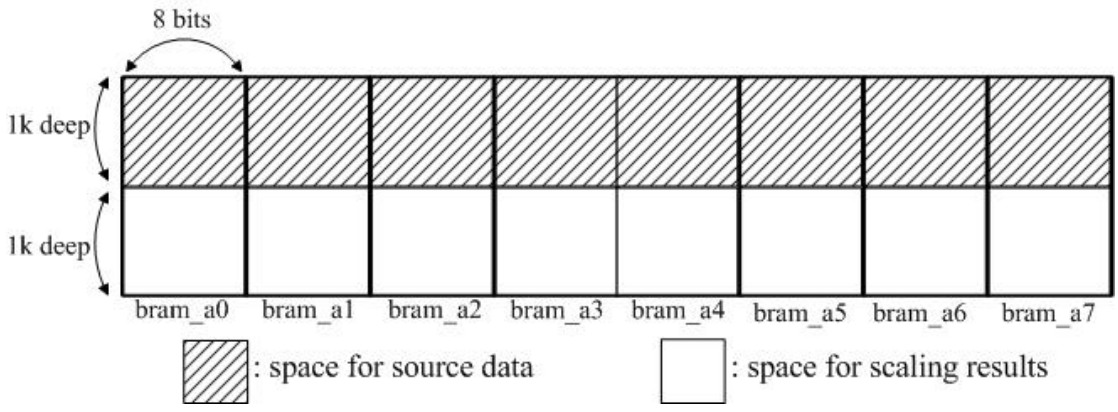


Figure 4.14 Data buffer in the composition logic

Figure 4.15 shows the control registers of the scaling circuits. The purposes of the start control register, busy state register and done state register are the same as those of the composition circuits. Since the dimension of the source can be arbitrary, the purpose of the src_W control register is to provide the source width information to the scaling circuits.

If we focus on the scaling operation, the coordinate transform equations on Figure 4.12 equal to: $\text{det\_x} = \text{src\_x} \times \frac{\text{dst\_W}}{\text{src\_W}}$, $\text{det\_y} = \text{src\_y} \times \frac{\text{dst\_H}}{\text{src\_H}}$. Since the division operation is not suitable for VHDL language, we calculate the ratio by software in advance. But, this would load to a serious rounding problem. To reduce the rounding problem, we multiply the ratio by 2^8 in advance, and then write the results to the W_ratio control register. After finish calculating the $\text{src\_x} \times \text{W\_ratio}$, we shift the result to right by eight bits to obtain the correct destination address.

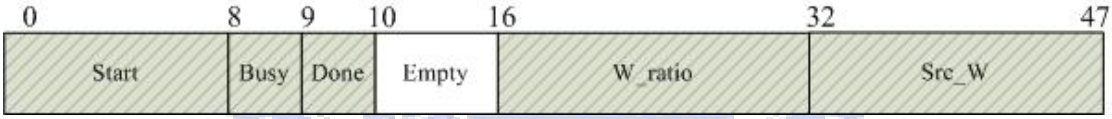


Figure 4.15 Control registers of the scaling logic

When the specific value “0x0A” is written to the start control register, the scaling operation will start. Some parameters are initialized as shown in the state diagram of Figure 4.16. For every index of the source, src_x, we calculate the corresponding maximum destination index by this equation: $\text{d\_x\_max} = (\text{src\_x} + 1) \times \text{W\_ratio}$ in the Com_addr state. In the Move data state, we simply duplicate the source data and write to the corresponding destination index: d_x_min ~ d_x_max. After the moving data operation, we write the d_x_max value to d_x_max, and return to the Com_addr state. The y-dimension scaling operation is done similarly.

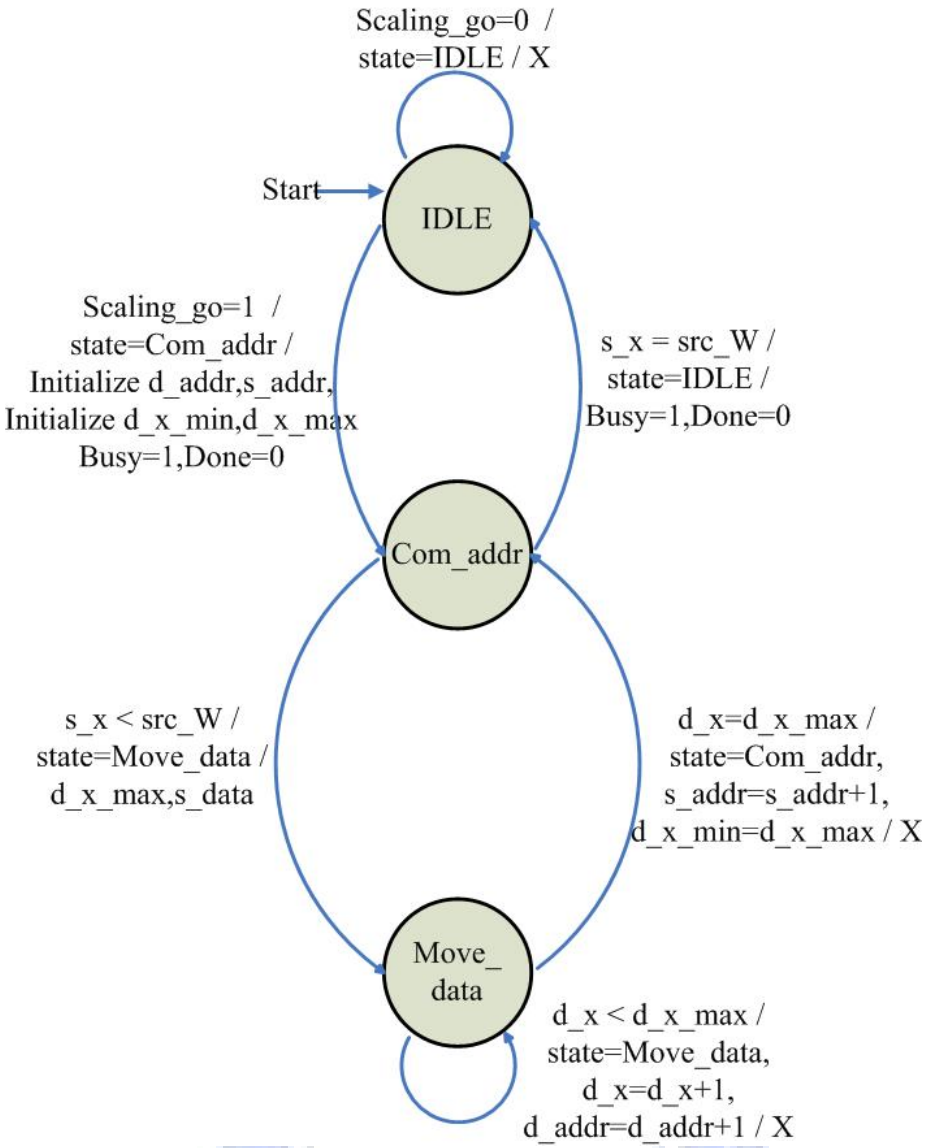


Figure 4.16 The FSMD model of the scaling logic

4.2.2 Standalone Subsystem Performance

The standalone scaling subsystem is much similar to the standalone composition subsystem. We connect our scaling logic to the reference ML403 design and then load 20 QCIF BMP pictures from the CF card to the memory. After the scaling operation, we write the results to the TFT_LCD screen buffer.

We use the 'XTime' library to profile the performance. We scale a QCIF image to the full screen size, run a 18000-frame loop, and compare the scaling execution cycles

of both the software and hardware version. The PowerPC core is running at 300MHz. Table 4.2 is the profiling results. The hardware scaling version is more than 6 times faster.

	Hardware scaling	Software scaling
Execution cycles	52213563369	334028245263
Frames / second	108.44	16.95

Table 4.2 The profiling of standalone scaling

4.3 Integration with JMF

4.2.1 Design and Implementation

The idea is to replace the software alpha composition and scaling units by the hardware ones. In order to integrate our hardware logics with the existing software JMF, we have to regenerate the ACE file, which contains the hardware binary codes and the Linux kernel [11] [12].

First, we download the ML403 reference design from the Xilinx web site and add the scaling logic and the composition logic. But, in this version, which includes the IPIF DMA service, the FPGA space is too small to contain the two hardware IPs. We remove the IPIF DMA service from the two logics, and generate the binary codes.

Then, we must build the BSP(Board Support Package) for re-build linux kernel. In order to build the BSP, we modify a few software parameters, including:

(1) Change the software platform to Linux

(2) Change the driver for GPIOs from v2.0 to v1.0

The open source Monta Vista Linux can be obtained from the web site, http://source.mvista.com. Rebuilding kernel requires the following steps.

(1) Prepare the cross compiling environment

(2) Copy BSP to the kernel source (drivers and ARCH directory)

(3) Cross-build:

- the cross-compiler should be listed in the search path
- command: make menuconfig ARCH="ppc" CROSS_COMPILE="powerpc-405-linux-gnu-"
  - turn off the network device and the character LCD support, we do not have to patch
- command: make dep ARCH="ppc" CROSS_COMPILE="powerpc-405-linux-gnu-"
- command: make zImage ARCH="ppc" CROSS_COMPILE="powerpc-405-linux-gnu-"

(4) Copy arch/ppc/boot/image/zImage.embedded to the reference design directory for building the ACE file

To integrate the binary codes and the re-building kernel:

(1) Launch EDK shell

(2) command: xmd -tcl genace.tcl -jprog -board user -target ppc_hw -hw implementation/system.bit -elf zImage.embedded -ace system.ace

The ACE file is generated after running though the above processes.

Now, we have a ACE file which contains the binary for the accelerating IPs and the corresponding Linux kernel. However, how do we access the registers or data buffers on the hardware IPs under the operation system environment? How do we map the physical addresses of these IPs to the virtual addresses? We can write drivers for these IPs, but there is a simpler way. There is a special device, /dev/mem, in linux. We can use the special device and the function, mmap(), to map the non-RAM (I/O

memory) address to the corresponding virtual address in the user-space. In order to use the mmap function, we have to include the sys/mman.h header. And the format of the call is as follows:

pa=mmap(**addr**, **len**, **prot**, **flags**, **fildes**, **off**);

The mmap() function shall establish a mapping between the address space of the process at an address **pa** for **len** bytes to the memory object represented by the file descriptor **fildes** at offset **off** for **len** bytes. A successful mmap() call shall return **pa** as its result.

The accelerating IPs are now integrated with the software JMF system successfully. Then, we make a simple performance profiling. We play a 54 seconds, thirty frames per second, QCIF MPEG-1 video, and count the execution time. Table 4.3 is the simple profiling results.

	Execution time (seconds)
Pure software JMF system	900
JMF system with hardware scaling logic	493
JMF system with hardware composition logic	1138
JMF system with hardware scaling logic and hardware composition logic	728

Table 4.3 The simple profiling of the JMF system with various configuration

We can see that the performance with JMF system is not improved as much as the performance in the standalone subsystem. Especially the performance of the composition logic is even slower than the pure software. The reason is the overhead of data access. In the standalone subsystem, DMA is in use to decrease the overhead. But, in the above simulations, we use the “memcpy()” function to access data. We also try to use DMA in this environment. However, we need to write the source and

destination physical addresses to the DMA registers. It is complicated because we do not have the physical address information in the user application space. A possible way is reserving an appropriate size of memory when the system boots with a driver for our IPs. Then, the driver can access the physical addresses of DMA, and it can also use kernel space functions to return the virtual addresses for user applications.

4.2.2 Performance Profiling

In order to evaluate the execution time performance, several scenarios are chosen and the executing time of a specific function is measured. These scenarios simulate typical DTV video and graphics presentations. This scenario simulates that a user navigate available service while watching a program as illustrated by Figure 4.17. The video that users are currently watching is presented on the video plane. At the same time, users select the EPG to navigate the other available services. The GUI components of the EPG are presented on the graphics plane, including several menu items and a component-based video to preview the video content of the service that users want to navigate.

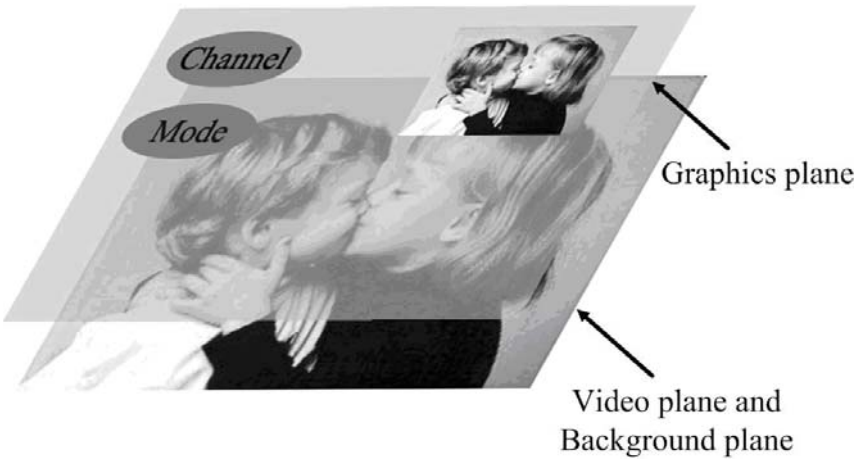


Figure 4.17 Scenario of typical EPG usage

To simulate this scenario, a component-based video and a full-screen-sized translucent AWT component are treated as the EPG GUI components. The decoded source video is in QCIF format. The background video presented on the video plane is scaled to have a full screen dimension: 640×480. The component-based video presented in the graphics plane has a dimension of the original decoded video, and therefore, no video scaling operation is needed. Both the component-based video and the background video have 891 frames.

Condition	JMF with composition and scaling accelerators (sec)	Pure software JMF (sec)	JMF with scaling accelerator (sec)
Background video Translucent AWT component Component-based video	592	987	520
Background video Translucent AWT component	211	327	191
Background video Component-based video	576	790	354
Background video	208	269	146

Table 4.4 The profiling of JMF system

We compare the performance of (a) pure software JMF, (b) JMF with composition and scaling accelerators and (c) JMF with scaling accelerator as shown in Table 4.4. We have the following observations.

(1) In each condition, the performance of JMF with composition and scaling accelerators is improved up to 30% as compared to the pure software version.

(2) The performance of JMF with the composition and scaling accelerators under the two conditions, background video and background video plus

translucent AWT component are very close. This is because the operations under the two conditions are the same. Sine every time the background video is changed, the SRC_OVER composition between video plane and graphics plane will execute.

(3) The performance of JMF with composition and scaling accelerators under the condition, background video plus component-based video, is obviously slower than that under the background video only condition. This is because we need to decode two video clips. Also, the composition operations slows down the performance, because the composition will be executed when the graphics plane is changed.

(4) The performance of JMF with the scaling accelerator is better than the performance of JMF with both the scaling and composition accelerators. This is due to the data access overhead problem that are discussed earlier.

Chapter 5
Conclusion and Future Work

In this thesis, two accelerating IPs, scaling and composition circuits, are implemented and integrated to a software MHP graphics and video system. These two hardware accelerate the system performance. To achieve this goal, several important steps have been gone through.

- We first need to understand the MHP graphics model and the composition pipeline.
- Have an in-depth knowledge of the target environment -- the Xilinx ML403 platform, including the EDK tools and the Linux operating system.
- Understand the pure software MHP graphics and video system.
- Design and implement the composition logic and the scaling logic.
- Implement standalone subsystems for the two accelerating IPs and evaluate their performance.
- Rebuild the MontaVista Linux kernel, connect our IPs to the ML403 reference design, generate the binary and finally generate the ACE file.
- Replace the software scaling and composition operation by these two accelerating IPs.

However, a few points are worth to continue to work on:

- Use more sophisticated algorithms for the scaling logic to eliminate the blocking effect.
- Use DMA to speed up the hardware.

References

[1] ETSI TS 101 812 V1.3.1, “Digital Video Broadcasting (DVB);Multimedia Home Platform (MHP) Specification 1.0.3”, June 2003.

[2] IBM, Inc., http://www-03.ibm.com/chips/products/coreconnect/, “CoreConnect Bus Architecture”

[3] Xilinx, Inc., http://www.xilinx.com/ipcenter/processor_central/coreconnect/coreconnect.htm, “CoreConnect Bus Architecture”

[4] Xilinx, Inc.,” ML401/ML402/ML403 Evaluation Platform User Guide”, May 2006

[5] Xilinx, Inc.,” Virtex-4 Family Overview”, January 2007

[6] Xilinx, Inc.,” Virtex-4 User Guide”, October 2006

[7] Xilinx, Inc.,” Embedded System Tools Reference Manual”, October 2005

[8] Xilinx, Inc.,” OS and Libraries Document Collection”, January 2006

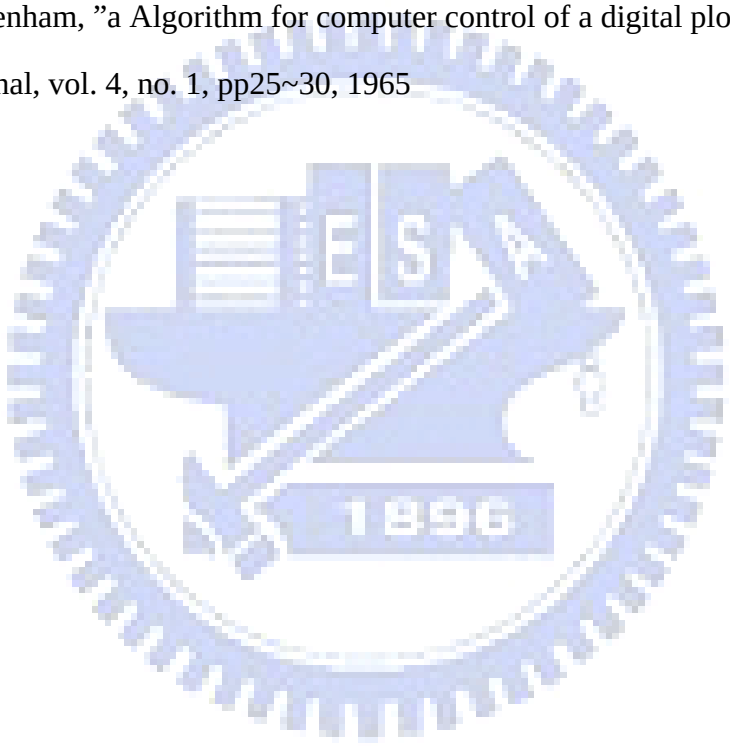
[9] Xilinx, Inc.,” OPB IPIF Product Specification”, April 2005

[10] Xilinx, Inc.,” PLB IPIF Product Specification”, August 2004

[11]Brigham Young University, http://splish.ee.byu.edu/projects/LinuxFPGA/configuring.htm , “Linux onML403 tutorial”

[12]Xilinx, Inc., http://toolbox.xilinx.com/docsan/xilinx8/help/platform_studio/html/ps_p_dld_using_genace_script.htm ,”commands to generate ACE file”

[13]J.E. Bresenham, ”a Algorithm for computer control of a digital plotter.” IBM Systems Journal, vol. 4, no. 1, pp25~30, 1965



自傳

朱浩廷，男，民國七十一年十一月十九日生於台灣省基隆市。民國九十四年六月畢業於交通大學電子工程學系，並於同年錄取交通大學電子研究所碩士班，從事多媒體系統之相關研究，指導教授為杭學鳴博士。民國九十六年八月取得碩士學位。