國 立 交 通 大 學

電機與控制工程學系

碩 士 論 文

生醫應用系統單晶片之 FPGA 實現

FPGA Implementation of Biomedical Application SoC

研 究 生：楊寓鈞

指導教授：林進燈 博士

中華民國九十八 年 一 月

生醫應用系統單晶片之 FPGA 實現

FPGA Implementation of Biomedical Application SoC

研 究 生：楊寓鈞　　Student：Yu-Chun Yang

指導教授：林進燈　　Advisor : Dr. Chin-Teng Lin

國立交通大學

電機與控制工程學系

碩士論文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

September 2008

Hsinchu, Taiwan, Republic of China

中華民國 九十八 年 一 月

生醫應用系統單晶片之 FPGA 實現

學生：楊寓鈞　　指導教授：林進燈 博士

國立交通大學電機與控制工程研究所

中文摘要

在目前生醫電子領域研究中，如何能夠即時處理與分析大量的生理訊號是一個重要的討論問題。以往電子醫療和分析儀器都相當龐大且價格貴，因此可攜式與嵌入式的生醫電子產品，需求日益增加。基於這兩個原因，本論文提出生醫應用系統單晶片的設計，以提供資源整合性的使用。

本論文提出一個生醫訊號處理的系統單晶片設計，具有(1)可程式化的監控與分析生理訊號能力及(2)整合性硬體擴充的空間：只要配置好暫存器存取位址，便能將客製化硬體設計透過系統匯流排整合進系統中。此系統單晶片架構包含可程式化的中央處理器、生醫訊號處理單元、系統匯流排、通訊和顯示介面。可程式化的中央處理器負責系統程序的排程和輸出入控制；生醫訊號處理單元能對生理訊號做訊號分離與分析的數位訊號處理核心；系統匯流排的設計能夠方便數位訊號處理 IP 的加入與移除。

經由實驗的結果顯示，本論文所提出生醫系統單晶片架構能在 FPGA 的平台上以 100 MHz 執行頻率達到訊號即時分析與顯示解析波形的效果。因為透過硬體執行與其他高速 DSP 的使用有很大的節省空間，如較少執行的週期、較低的功率消耗、減少 PCB 設計的面積等。最後，本論文所提出的架構透過 FPGA 實現並在 Altera DE2 發展板展示，共使用 29,640 邏輯單元。

FPGA Implementation of Biomedical Application SoC

Student：Yu-Chun Yang Advisor : Dr. Chin-Teng Lin

Department of Electrical and Control Engineering
National Chiao Tung University

Abstract

In the study of biomedical electronics nowadays, processing and analyzing masses of physiological signals in time is a critical issue in real world. In the past, electronic treatment and analysis instruments are very expansive and large. Therefore, the requirement of portable and embedded biomedical electronic product is growing rapidly. Based on these two reasons, this thesis presents a system-on-chip (SoC) design for biomedical applications to provide the integration of systematic resources.

In this thesis, the SOC design for biomedical signal processing is presented. It has the following features. (1) The ability of programmable monitoring and analyzing biomedical signals. (2) The flexibility for further hardware extension. The SoC design is composed of a programmable CPU, biomedical signal processing (BSP) units, system bus, communication, and display interface. The programmable CPU manages the process schedule and I/O control. The BSP unit is treated as digital signal processing (DSP) cores which can separate and analyze physiological signals. The system bus makes it flexible to add or to remove DSP IPs.

By experimental results, the proposed design implemented on FPGA can achieve real-time analysis and waveform displays under at 100 MHz. Comparison with other high-speed DSP processors, the system presents some optimization such as less execution cycles, lower power consumption, use of fewer PCB area. Finally, the SoC design is demonstrated in this thesis with the Altera DE2 development board. The whole design is consisted of 29,640 logic elements.

誌謝

兩年的研究所生涯隨著論文的完成劃上了句號，這兩年間，要感謝許多人的鼓勵和幫忙，使我獲得充實的專業能力並順利完成研究所的學業。

首先要感謝的是我的指導教授-林進燈老師。感謝老師提供了很理想的研究環境、豐富的資源及正確的引導，使我在研究上非常順利。在老師悉心的指導下，讓我學習到解決問題的能力及做研究應有的態度，使我獲益良多。感謝范倫達教授時常關心我學業上的研究，時常與我討論論文方向及進度。

另外也感謝實驗室的麻吉們，感謝經翔學長開啟我雙主修之路的大門，還有在專業上跟生活上給予我許多的建議。感謝訓緯在生活上很照顧我。德瑋、俊傑、靜瑩及智文等學長姐們謝謝你們。還有我的同學們：煒忠，心情好或壞總是有人互相支持。儀晟，我們是706閃靈二人組喔。孟修，是不是一輩子的好兄弟，陽台時間總是很開心。福雄，謝謝你的照顧啦。建昇、俊彥，最後這段時間有你們一起衝刺真的很好。依伶，感謝一姐給我一流的建議跟幫助。毓廷、孟哲，有你們的生活真的是歡樂不斷。

我也要謝謝實驗室的學弟妹：昕展，你跟我都走一樣的路，一樣辛苦，要加油！也謝謝你了！哲睿，真的很高興能跟你玩在一起耶，很高興我們時常能互相討論耶，是不是陽台四人組！介恩，謝謝你在半夜心情不好時的義氣相挺，加油，跟你在一起的陽台時間總是特別白爛！家欣、有德，謝謝你們在研究上及生活上的互相扶持及鼓勵。志賢，很謝謝你的陪伴，能玩在一起真是太開心了。雞蛋瑄，是不是生活上，運動上，心情上大家都是很好的咖，很高興有你這個好友喔。阿航，跟你瘋癲哈啦還真是快樂。文文，感謝你這麼挺我這個學長，讓我很感動。Frank，謝謝你這個半夜咖加早餐咖。聖祥，謹譽、敬婷、凱能，你們真是太可愛的一群人了。蘋果鈴，謝謝你這個早餐咖常在半夜義氣相挺，有你的生活真的是歡樂不斷，小～黑，有些話真的是盡在不言中啦，感謝感謝。

Thanks Kevin, you are really my buddy, I enjoy small break with you all the time. Thanks Clare, you make me grow up to be who I am now.孟宏，謝謝你這個摯友，總是挺我挺到底，有你這好友真好。

感謝教會的好朋友們，郁翔，真的是換帖的。仲石、孟蓉，你們真真正正是自己人，太感謝你們的照顧了。還有C團的大家、社青團契還有關心我的人，謝謝。Cindy, thanks for your prayer and tears, I'll keep the word I promised you.

最後要感謝家人:老爸，你真的是我的偶像，是我很大的靠山，謝謝你全力的支持跟呵護，我都收到啦。感謝老媽的支持，你真的真的從頭挺到尾耶，有沒有這麼可愛的媽媽呀，愛妳喔。還有老姊跟姊夫，你們真的是太照顧我了，讓我感受到很溫暖的關心，謝謝你們。我也要謝謝上帝，賞賜的是耶和華，收取的也是耶和華，Thanks god!

Contents

List of Figures

List of Tables

Chapter1
Introduction

1-1 Motivation

"Ageing population" is an important healthcare issue in advanced countries. A statistic report issued by Executive Yuan, R.O.C (Taiwan) pointed out until May 2008 geriatric population in Taiwan has reached 2362,223, that is, 10.28% among the total population in Taiwan and has exceed the threshold of 10%. Obviously, Taiwan has become a geriatric society and therefore the development of geriatric healthcare is an important policy of the government. The difficulty of geriatric healthcare is the abrupt change of physical condition of elder people so it is necessary to monitor remote and real-time physical. In recent years, the study of cognitive neuroscience has revealed the mystery of human brain-behavior. Thus we can get more specific information by analyzing brain activities signals associated with other physical signals for example, electrocardiograms (EKG), blood pressure, etc.

The Brain Computer Interface (BCI) system refers to a set of sensors and signal processing components which are able to acquire and analyze brain activities. With the goal of establishing a reliable communication channel directly between human brain and an external device such as a computer, portable monitoring device, neuroprosthesis, etc. Several existing brain monitoring technologies such as electroencephalogram (EEG), functional magnetic resonance imaging (fMRI),etc, have been tested in BCI fields for data acquisition. The research of Ashwin et al. 0 presented a system that monitored EEG of epileptic patients to improve the quality of their daily life and also helped healthcare providers to make a better diagnosis for

geriatric or patients who is suffering from neurological disorders.

Independent Component Analysis (ICA) [2], which had been widely studied during last two decades, is one of popular EEG signal separation method. ICA has been proved as a powerful algorithm to solve blind source separation (BSS) [3] problems in a variety of signal processing applications such as speech [4], image, or biomedical signal processing. Many groups are now engaged in exploring the potential of BSS for revealing new information from the brain activities and physical signal [5]. Hill et. al.[6][7][8], demonstrated the use of ICA for an EEG-based BCI. However, the characteristic of general ICA is limited to process enormous but off-line data. Clinically, off-line ICA is defiantly unable to satisfy the requirement of real-time diagnosis or healthcare monitoring system. In order to achieve the goal of portability and wearability of systems applied in surgery, clinic diagnosis or healthcare monitoring, more and more researches focus on faster on-line ICA implementation in software or hardware manner. Given the development of System-on-Chip (SoC) and signal processing techniques, it is now practical to implement these sophisticated algorithms in real-time SoC systems for real-time EEG monitoring and/or BCI.

Fast Fourier transform (FFT) is a well developed algorithm for signal analysis in frequency spectrum, which is first discussed by Cooley and Tukey [9]. They conspicuously improved discrete Fourier transform (DFT). Gauss had actually described the critical factorization step as early as 1805. The FFT faster calculation manner saves lots of multiplication and also saves abundant computation time. This valuable property of FFT makes it easy to meet the requirement of real-time digital signal processing. Another valuable property is that FFT inherits the discrete computation nature of DFT, therefore it is easy to be implemented with VLSI technology.

This study details the design and simulation result of a real-time BCI system in a SOC viewpoint, which features two digital signal processing unit, they are, a four-channel ICA for blind data separation and a 64-points Fast Fourier transform for spectral analysis. Since the large number of mathematic computation is required, it is hard to real-time process in embedded systems. However, integration of these two DSP integrated circuits (IC) with system resources and system control unit (micro processor, DRAM, BUS interface, VGA, etc,) on the FPGA can not only accelerate the speed of the operation circuit by parallel processing, but also show real-time computation and low-power property.

1-2 Objectives

It is important for a user to feel that the healthcare device is easy-to-use and efficient. In order to meet the requirement of operation convenience and real-time processing, a convenient user interface and high performance signal processing unit are necessary. Based on the above two reasons, a novel biomedical application SoC will be presented and implemented on FPGA.

In this thesis, there are two DSP units (ICA and FFT) integrated into SoC. Through the Avalon bus interface, other resources such as SRAM, DRAM, and Flash memory can also be integrated successfully. Furthermore, peripheral interfaces such as UART, a 16×2 literal liquid crystal display (LCD), a VGA with monitoring LCD which are also included in the proposed design. Because of customized design of these hardware resources, hardware driving program needs to be provided. Through the driver program, four-channel brain signals and there frequency spectrum can be shown in a window interface. With the aid of DSP units, the proposed design can achieve large amount of computations under only 100 MHz operating frequency,

therefore, both the real-time signal processing and low-power consumption can be achieved at the same time.

1-3 Organization of the Thesis

This thesis is organized as follows. The system level architecture and system level simulation results are revealed in chapter 2. FPGA implementation details of each component are introduced in chapter 3 individually. The experimental results and discussions are presented in Chapter 4. Conclusions and future work are made in the last chapter.

Chapter2
Biomedical SoC Architecture

2-1 System Architecture

The architecture of the system is shown in Fig. 2-1, mainly consists of nine modules that communicate through Avalon BUS interface: NIOS II processor, ICA module, FFT module, SDRAM/VGA controller, UART interface, 16×2 LCD controller, 7 segment display controller, and SDRAM controller.

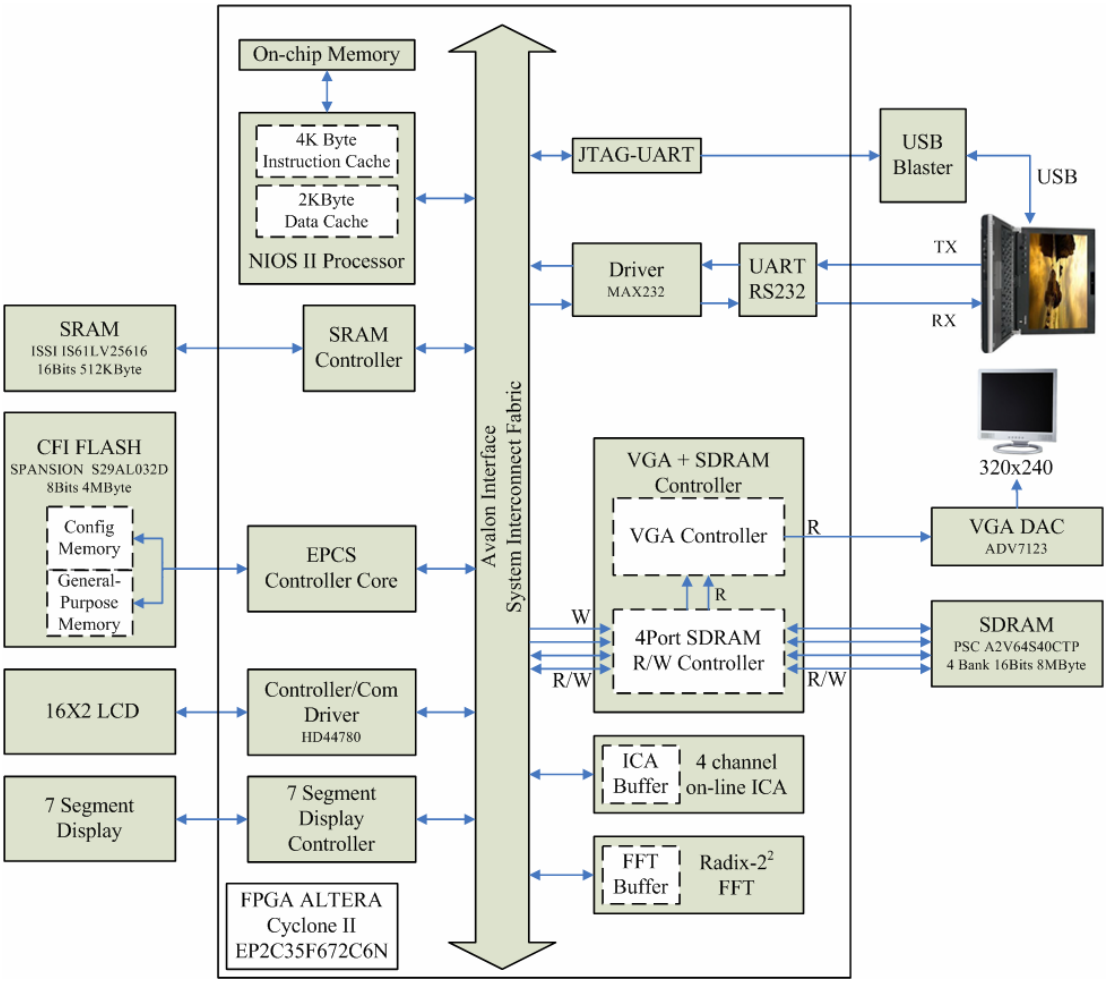


Fig. 2-1 System architecture.

The system accepts four-channel brain activity EEG through UART interface then transfers incoming datum to SRAM. Next, blind source is sent to ICA module and is separated into four channels. After separation, data will send to FFT for further spectral analysis. Either EEG in time domain or EEG frequency spectrum can be shown on a 640×480 pixels LCD screen(actually only 320×240 pixels of 640×480 are used in this work). Some peripheral I/O such as 7 segment display and 16×2 literal LCD can show intermediate result or execution messages.

The functionalities of each module are briefly introduced above. In the rest of this chapter, we will specifically depict the architecture and implementation of each module.

2-1-1 Hardware Configuration

The real hardware resource and RTL level design are distinguished by the black rim in Fig. 2-1. The RTL level design would finally programmed into FPGA chip (Cyclone II EP2C35, Altera Corp). Some major hardware configuration is depicted as follows:

(1) SRAM [10]

The SRAM in this system is a 512-Kbyte asynchronous CMOS static RAM memory chip produced by ISSI (Integrated Silicon Solution, Inc). It is Accessible as memory for the Nios II processor. The SRAM is organized as 256K words by 16 bits. It is fabricated using ISSI's highly reliable process coupled with innovative circuit design techniques, yields high-performance and low power consumption.

The functional block diagram is given in Fig. 2-2. In order to access correct physical cell, address decoder decodes input address into row address and column address. Input/Output data correspond to write data and read data. The rest pins are control pins. When $\overline{CE}$ is high (deselected), the SRAM assumes a standby mode at

which the power dissipation can be reduced down with CMOS input levels. Easy memory expansion is provided by using chip enable and output enable inputs, $\overline{CE}$ and $\overline{OE}$. The active low write enable ($\overline{WE}$) controls both writing and reading of the memory. A data byte allows upper byte ($\overline{UB}$) and lower byte ($\overline{LB}$) access.

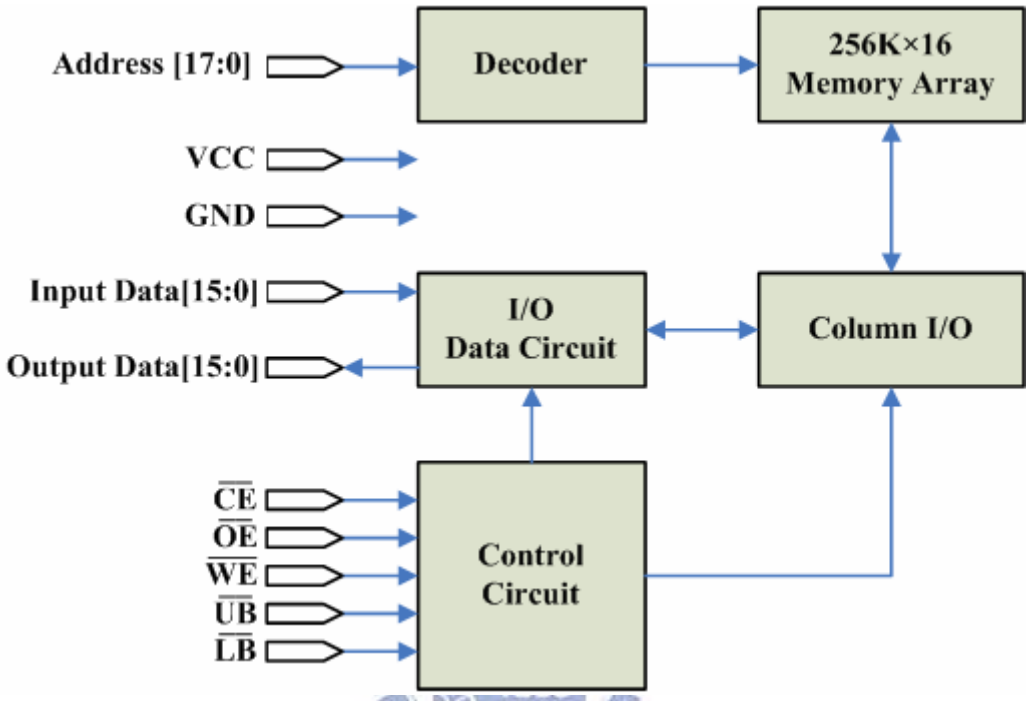


Fig. 2-2 Functional block diagram of SRAM.

(2) SDRAM [11]

The SDRAM in this system is an 8-Mbyte synchronous high data rate dynamic RAM chip manufactured by PSC (Powerchip Semiconductor, Corp).

It is organized as 4 banks ×1M words by 16 bits. It is accessible as memory for VGA controller and other IPs. Synchronous design allows precise cycle controls with the use of system clock I/O transactions are possible on every clock cycle. The operating frequency is under 100MHz.

The functional block diagram is given in Fig. 2-3. Chip select ($\overline{CS}$) enables and disables the command decoder. All commands are masked when $\overline{CS}$ is high. $\overline{CS}$ is considered part of the command code. $\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$ are command inputs, they

define the command being entered. Bank address (which is not shown in Fig. 2-3) defines to which bank an ACTIVE, READ, WRITE or PRECHARGE command is begin applied. Address inputs provide the row address (Address[11:0]) for ACTIVE commands, the column address(Address[7:0]), and AUTO PRECHARGE bit for READ/WRITE commands, to select one location out of the memory array in the respective bank.

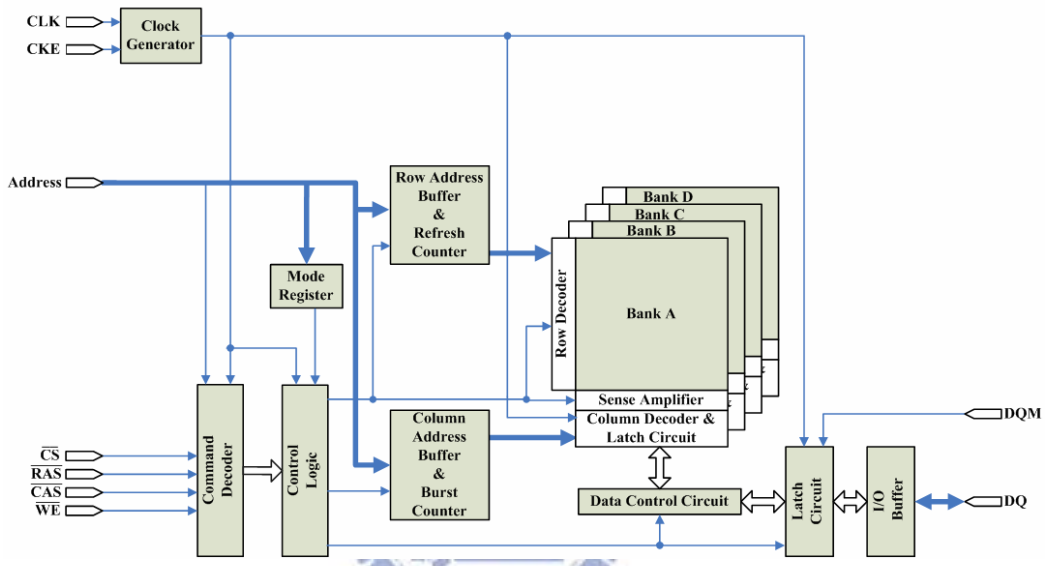


Fig. 2-3 Functional block diagram of SDRAM.

(3) Flash memory [12]

The Flash is a 4-Mbyte, 3.0 volt-only NOR flash memory device, organized as 2M words of 16 bits each or 4M words of 8 bits each. The functional block diagram is given in Fig. 2-4. Word mode data appears on DQ0-DQ15, byte mode data appears on DQ0-DQ7. The device can be programmed in standard EPROM programmers. Standard control pins- chip enable ($\overline{CE}$), write enable ($\overline{WE}$), and output enable ($\overline{OE}$)-control normal read and write operations, and avoid bus contention issues.

It is accessible as read-only memory the system. Essential GUI components are stored in the Flash memory in the form of zipped file system. The contents can be programmed by flash programmer, a tool kits provided by Altera.

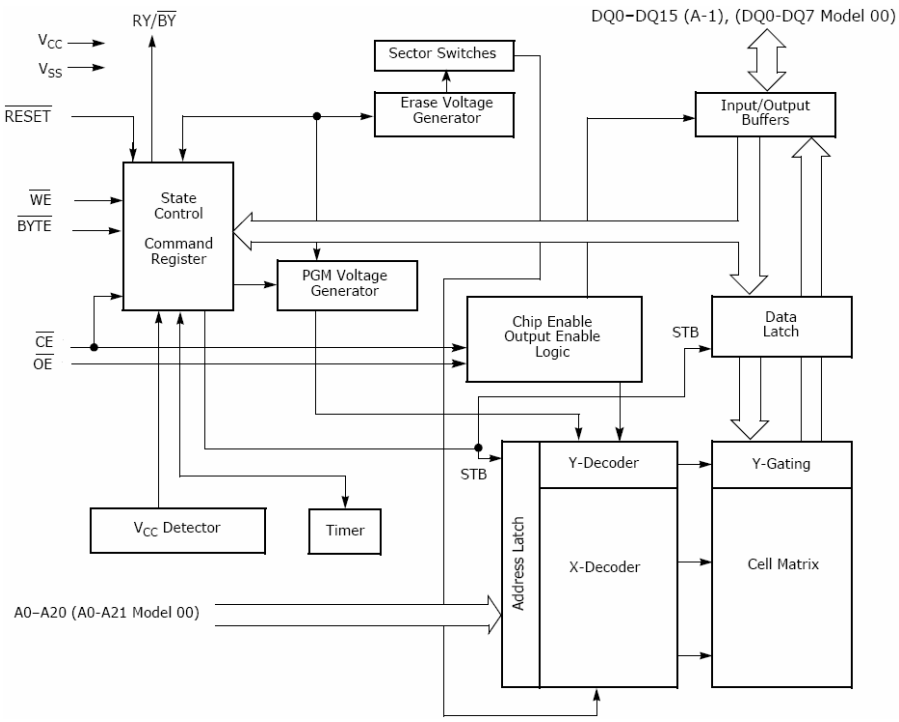


Fig. 2-4 Block Diagram of Flash memory.

(4) VGA DAC [13]

The VGA D/A converter is a triple high speed, digital-to-analog converter on a single monolithic chip. It consists of three high-speed, 10-bit, video D/A converters with complementary outputs. The VGA DAC has three separate 10-bit-wide input ports. The functional block diagram is given in Fig. 2-4. It has additional video control signals, $\overline{\text{SYNC}}$ and $\overline{\text{BLANK}}$. When $\overline{\text{BLANK}}$ is logically zero, it drives the analog outputs, IOR, IOB and IOG, to the blanking level, say, the R0–R9, G0–G9 and R0–R9 pixel inputs are ignored. A logical zero on the $\overline{\text{SYNC}}$ input switches off a 40 IRE current source. Also, $\overline{\text{SYNC}}$ does not override any other control or data input, therefore, it should only be asserted during the blanking interval. R, G, B, are red, green and blue pixel data inputs. IOR, IOG, IOB are red, green, and blue current outputs. All the control signal and pixel data are latched on the rising edge of clock. In this work VGA DAC supports 640×480 at 60-Hz refresh rate

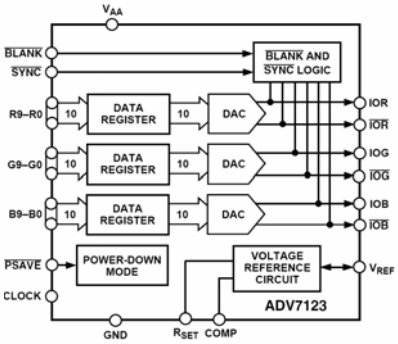


Fig. 2-5 Functional block diagram of VGA DAC.

(5) RS232 Serial port [14]

The serial port is driven by a dual driver/receiver chip (Texas Instruments, Inc) that includes a capacitive voltage generator to appropriate voltage levels from a single 5-V supply. It works at the baud rate 115200 bits per second.

2-1-2 Systematic Data Flow

Below, a brief introduction of systematic data flow is described referring to system architecture in Fig. 2-6. The netlist level design of whole system design is programmed to FPGA through USB blaster. Also, system program and driver program can be compiled on user PC and loaded to SRAM by using USB blaster. As Fig. 2-5 shows, SRAM is the main program memory and on-chip memory is the exception stack.

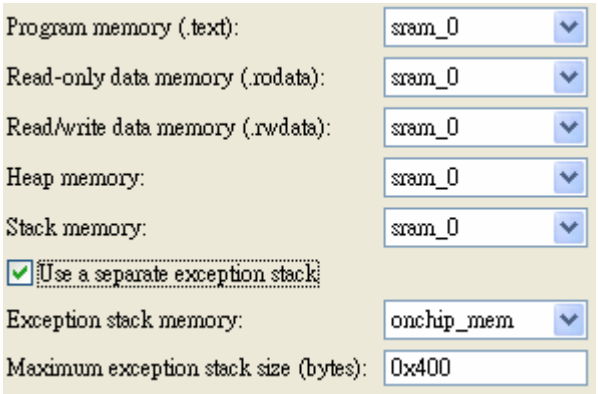


Fig. 2-6 Detail memory configuration.

The system program is always waiting inputs from UART receive (RX) port, the received datum are transferred to CPU's register through Avalon BUS interface. Next, system program will write received RX datum to hardware ICA. Once ICA has its result, CPU will read the result through system BUS that is the separated four channel datum. After getting separated datum, CPU writes datum to hardware FFT. When FFT finishes frequency spectrum analysis, CPU will read the result through system BUS.

System program also allocate a 320×240 32-bit array to store the RGB value of the frame. This frame RGB array updates the graphic interface and wave form of ICA and FFT result. User can manually select the demo mode to switch between ICA demo mode and FFT demo mode. The graphic interface components are stored in flash memory. Therefore, if user changes demo mode, CPU will load GUI component from flash memory and renew the wave form then saves new contents in the frame RGB array. Every frame renew will cause CPU to write the content of frame RGB array into SDRAM, which is the actual hardware frame buffer of the display.

2-2 Microprocessor Core

This section is a brief introduction to the Nios II embedded processor, which acts an important role in this system. The systematic data flow controlling programs and customized driving programs are all compiled into Nios II instruction set, and are executed by Nios II processor.

2-2-1 Introduction of Nios II Processor

Nios II is a RISC soft-core embedded-processor architecture designed specifically for the Altera family of FPGA. The Nios II processor is a general-purpose RISC processor core, featuring with:

- 32-bit instruction set, data path, and address space.
- 32 general-purpose registers and 32 external interrupt sources.
- 32×32 single-instruction multiplier and divider producing a 32-bit result.
- Instructions for computing 64-bit and 128-bit products of multiplication.
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and/or user defined peripherals.
- Software development environment based on the GNU C/C++ tool chain and Eclipse IDE.

2-2-2 Hardware Architecture of Nios II Processor

The Nios II architecture describes an instruction set architecture (ISA). A Nios II processor core is a hardware design that implements the Nios II instruction set. The processor core does not include peripherals or the connection logic to the outside world. It includes only the circuits required to implement the Nios II architecture.

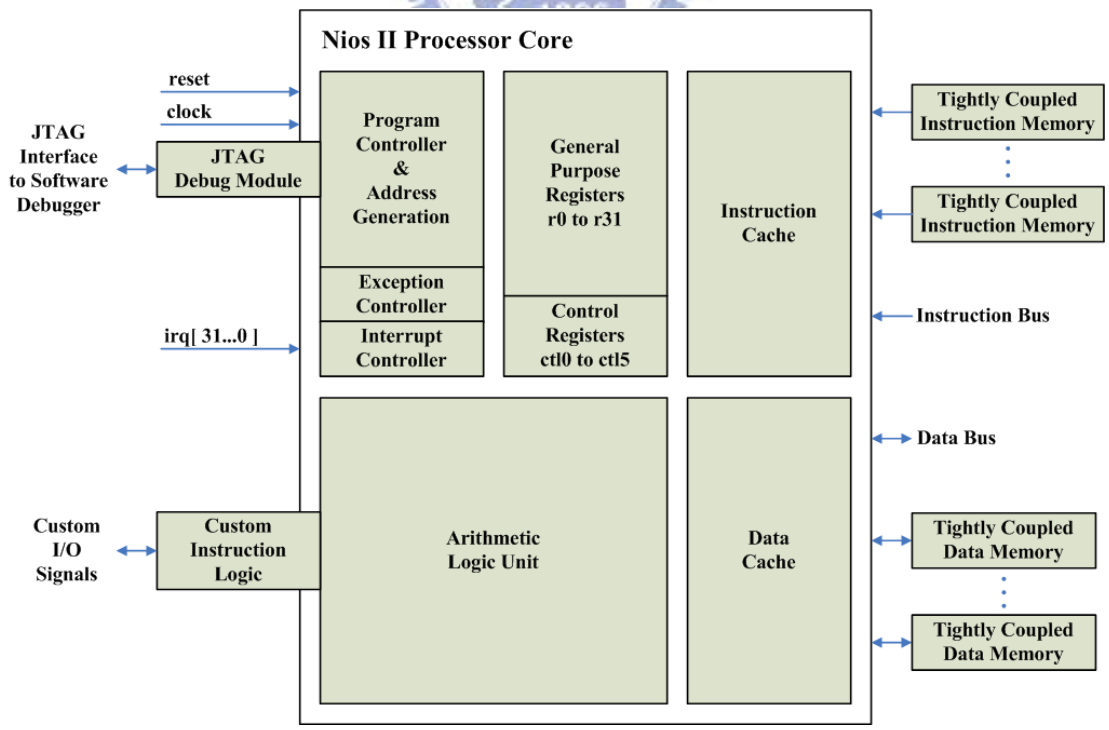


Fig. 2-7 Nios II processor core block diagram.

The block diagram of Nios II processor is shown in Fig. 2-7. The Nios II architecture defines some user-visible functional units, including register file, arithmetic logic unit (ALU), interface to custom instruction logic, exception and interrupt controller, instruction/data bus, instruction/data cache, tightly coupled memory interfaces for instructions and data, JTAG debug module. The register file consists of thirty two 32-bit general-purpose integer registers, and six 32-bit control registers. ALU operations take one or two inputs from data stored in general-purpose registers, and store a result back to a register.

Nios II processor core do not provide hardware to perform multiplication or division operations. Some instructions related to multiplication/division are known as unimplemented instructions. They are instructions not begin implemented in hardware. On the contrary, their operations are emulated in software. The processor generates an exception whenever it issues an unimplemented instruction, and the exception handler calls a routine that emulates the operation in software.

All exceptions, including hardware interrupts, cause the processor to transfer execution to a single exception address. The exception handler at this address determines the cause of the exception and dispatches an appropriate exception routine.

2-2-3 Memory & I/O Organization of Nios II Processor

Figure 2-8 shows a diagram of the memory and I/O organization for a Nios II processor. The Nios II architecture supports separate instruction and data buses. Both data memory and peripherals are mapped into the address space of the data master port.

The Nios II instruction bus is implemented as a 32-bit Avalon master port. It performs a single function, that is, fetch instructions. In addition, it does not perform

any write operations. The instruction master port is a pipelined Avalon master port. Support for pipelined transfers minimizes the impact of synchronous memory with pipeline latency and increases the overall throughput of the system. Thus it can issue successive read requests before data has returned from prior requests. The Nios II processor can prefetch sequential instructions and perform branch prediction to keep the instruction pipe as active as possible. The instruction master port always retrieves 32 bits of data.

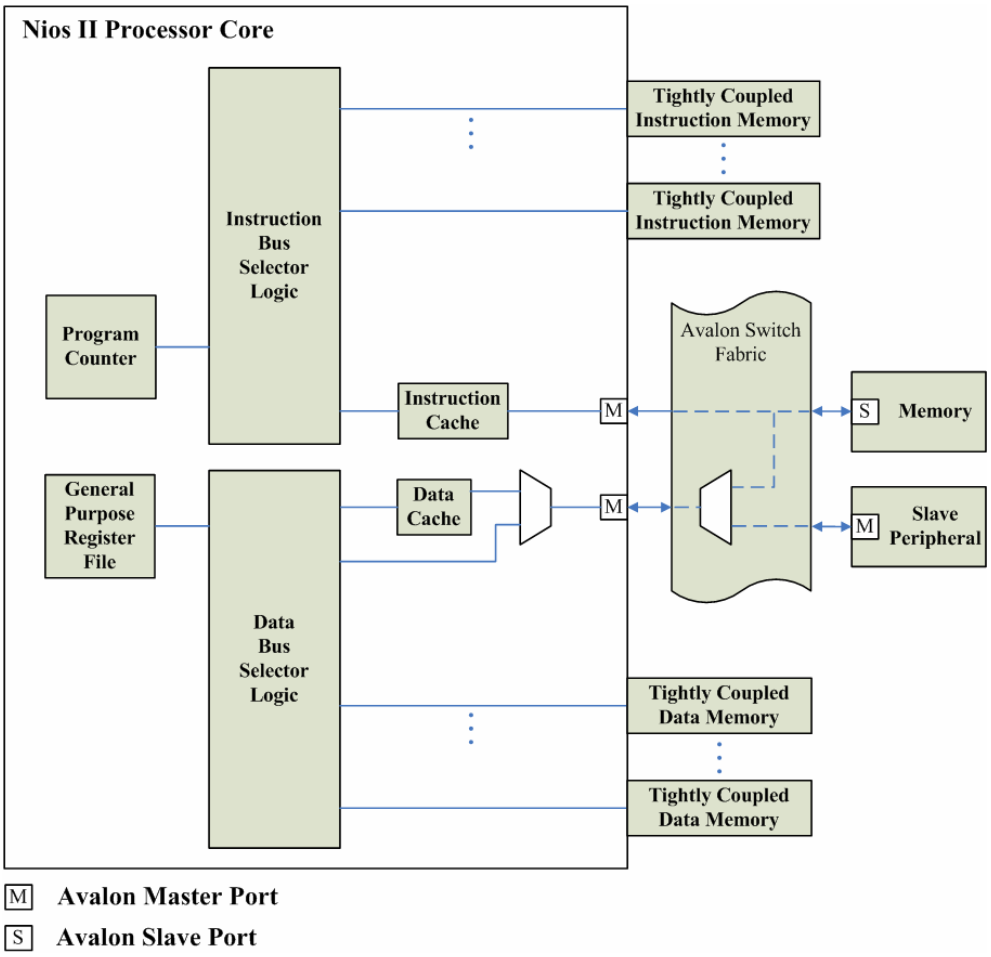


Fig. 2-8 Nios II Memory & I/O Organization.

The Nios II data bus is a 32-bit Avalon master port. The data master port performs two functions. One is reading data from memory or a peripheral when the executing a load instruction. The other one is writing data to memory or a peripheral

when a store instruction is executed.

The Nios II architecture supports on-chip cache memory for improving average data transfer performance when accessing slower memory. Usually the instruction and data master ports share a single memory that contains both instructions and data. While the processor core has separate instruction and data buses, the overall Nios II processor system may present a single, shared instruction/data bus to the outside world.

2-3 Biomedical Signal Processing Units

In usual case, it is easier to develop signal processing function in software and execute the function by microprocessor. One inevitable drawback of signal processing in software manner is that microprocessor may fail to finish such large amount arithmetic operation between two contiguous incoming data. Since we intend to monitor physical signal, it is necessary to achieve the goal of real time signal processing, thus we process biomedical signal in a hardware manner.

There are two signal processing unit in this biomedical signal processing core. One unit is independent component analysis, which is able to separate four channel input signal and decrease noise. The other one is Fast Fourier transform, which is responsible for signal analysis in frequency domain.

2-3-1 Introduction of on-line ICA

ICA is created to solve cocktail-party problems in signal processing. There are situations where there are a number of signals produced by some physical sources. These signals could be, for example, electric signals from different brain areas, speech signals from different people speaking in the same room [15], or radio waves from different mobile phones in the same area [16]. The sensors are placed in

different positions, so that mixtures are different from one another as a result of space factors.

In practice, the information about the original signals and the mixing system are unknown, and the information of mixed signals from sensors. For this reason, drawing out original signals from those mixtures is professed Blind Source Separation (BSS). The BSS problem is illustrated in Fig. 2-9. ICA is one of the useful methods to precede BSS problems which separate signals mainly by independence. In the following contents, representations will be called components due to the name of ICA.

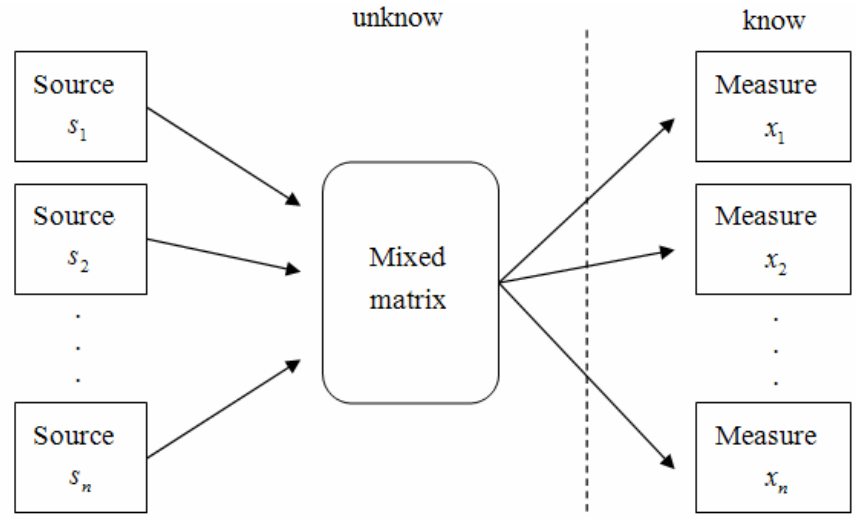


Fig. 2-9 Illustration of the BSS problem

ICA looks for components that are both statistically independent and non-Gaussian from mixed data which distinguishes ICA from other BSS methods. Besides, ICA gives good representations of source signals through the linear combination of mixed signals with non-linear decorrelation methods. In practical situations, it is easy to find the components which are really non-Gaussian.

On the other hand, the best de-mixing matrix that makes the components really independent to each other can not be found in general. It should be noted that algorithms exist to make the components as independent as possible.

Most of the work on BSS so far addresses the case of mixtures, where a linear mixture model is assumed:

$$x(t) = A \times s(t), \tag{2.1}$$

where s(t) is the vector of sources at instant t, A is the mixing matrix, and the observed vector of mixtures (ignoring noise). ICA now consists of estimating both the matrix A and s when x is the only given signal. It should be noted that the number of independent components $s(t)_i$ equals to the number of observed variables $x(t)_i$ which is a simplifying assumption and is not completely necessary. ICA obtains a n×n matrix W where

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \cong \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = W \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \tag{2.2}$$

Above the equation, $s_i(t) \approx y_i(t) = W \times x_i(t)$, y_i is called the representation of sources from the measurements. If is more similar to S, it is a better representation. Figure 2-10 shows the formulation of ICA.

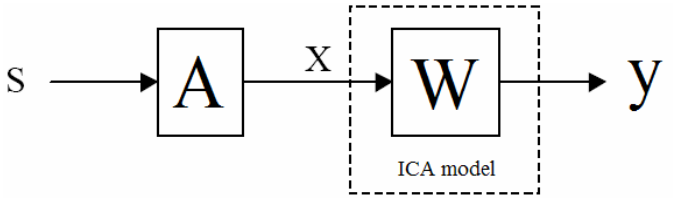


Fig. 2-10 Illustration of ICA formulation

Since the original blind source separation algorithm by Jutten and Herault [17], several on-line and batch mode algorithms have been formulated under the umbrella of independent component analysis. While some of the batch ICA algorithms such as JADE [18] and FastICA [19] give relatively fast convergence in estimating W, they are not quite suitable for on-line implementation in a real-time setting. Depend on

Gradient information learning rules, the weight need update at each new division. In order to fix the direction of weight are the same in every division, overlapping previous mixed sources will fix the order of the weight. The method of data-stream processing illustrate in Fig. 2-11.

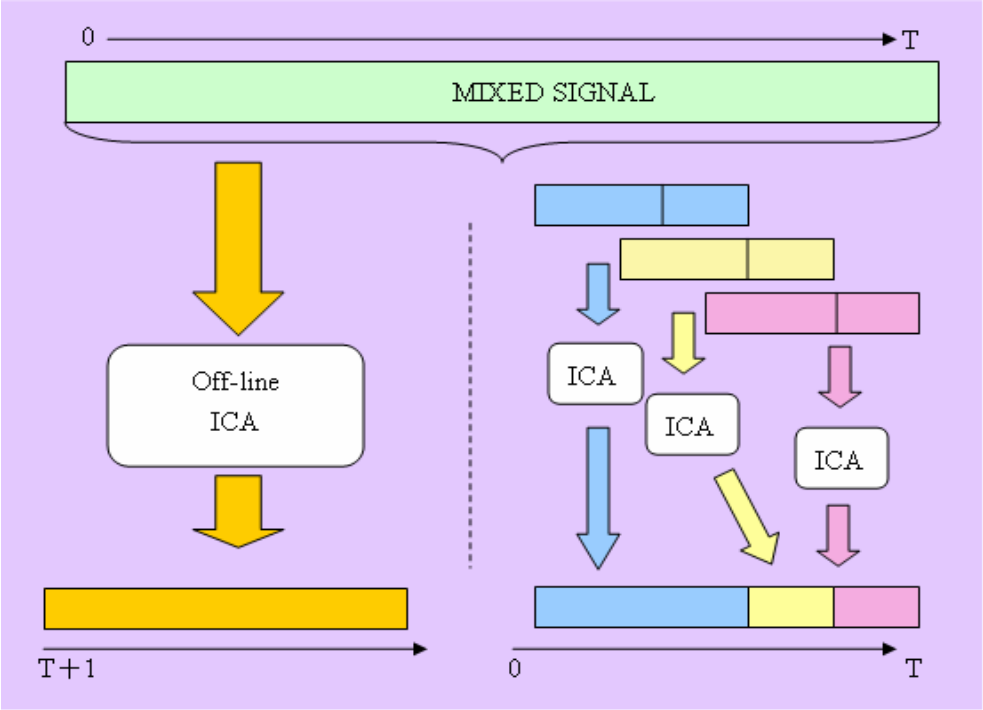


Fig. 2-11 Illustration of off-line and on-line algorithm.

Figure 2-12 show the ICA MODEL, X(t) are measure data. After preprocessing, the model will enter the main calculation unit, including non-linear transformation, and gradient information update. However, in on-line processing of data stream, each division data stream will flow through the ICA model, then, each time the weight be calculated for each division data. Briefly, we take eight seconds division data to processing, and updating two seconds because of the overlap are six seconds. Finally, if the weight is stable, Y(t) represent the independent components which is the product of input X(t) and weight matrix.

ICA MODEL:

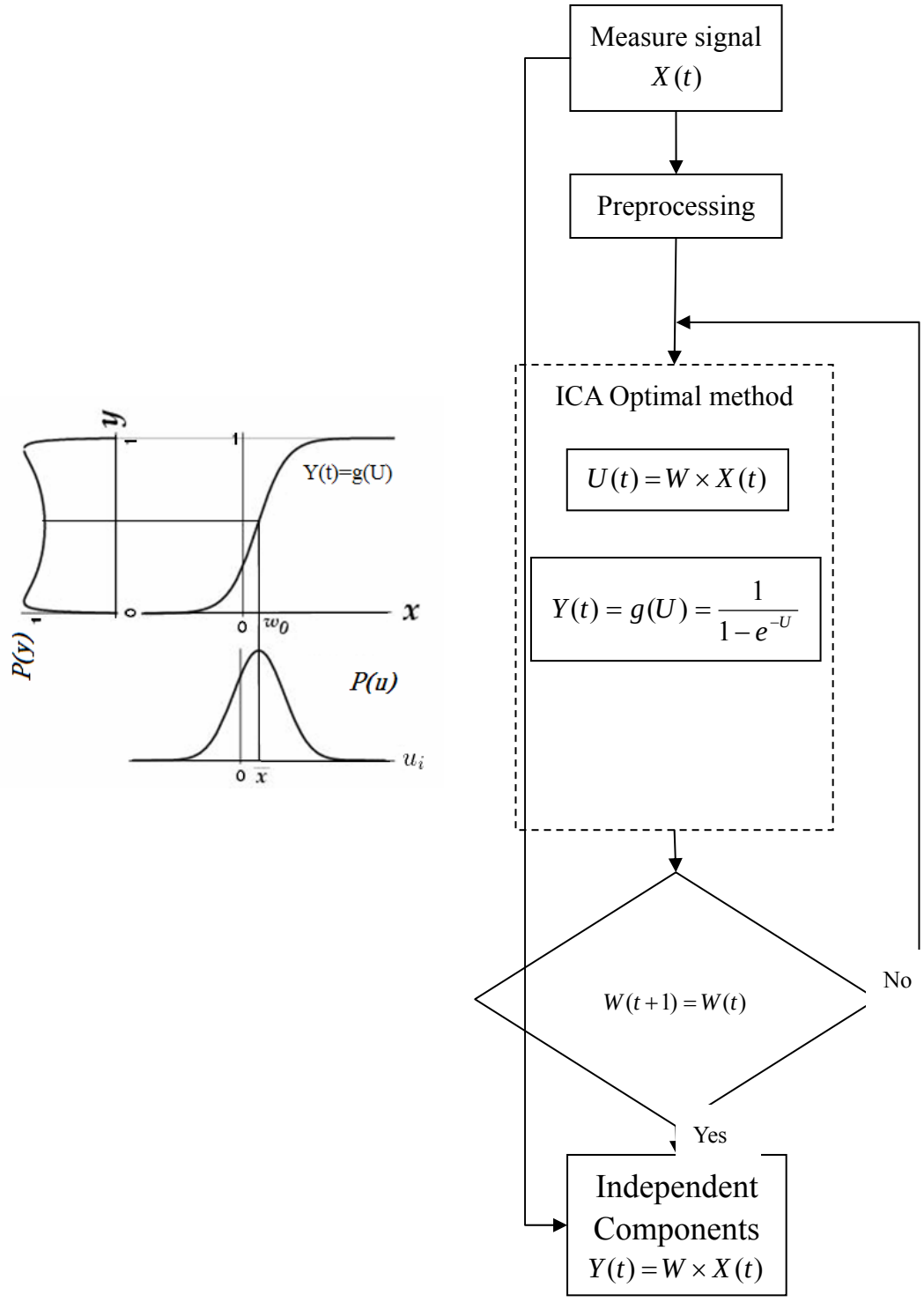


Fig. 2-12 Diagram flow of the computation of implementation of on-line ICA learning algorithm.

2-3-2 Structure of on-line ICA

Top level real-time hardware architecture shown in Fig. 2-13.

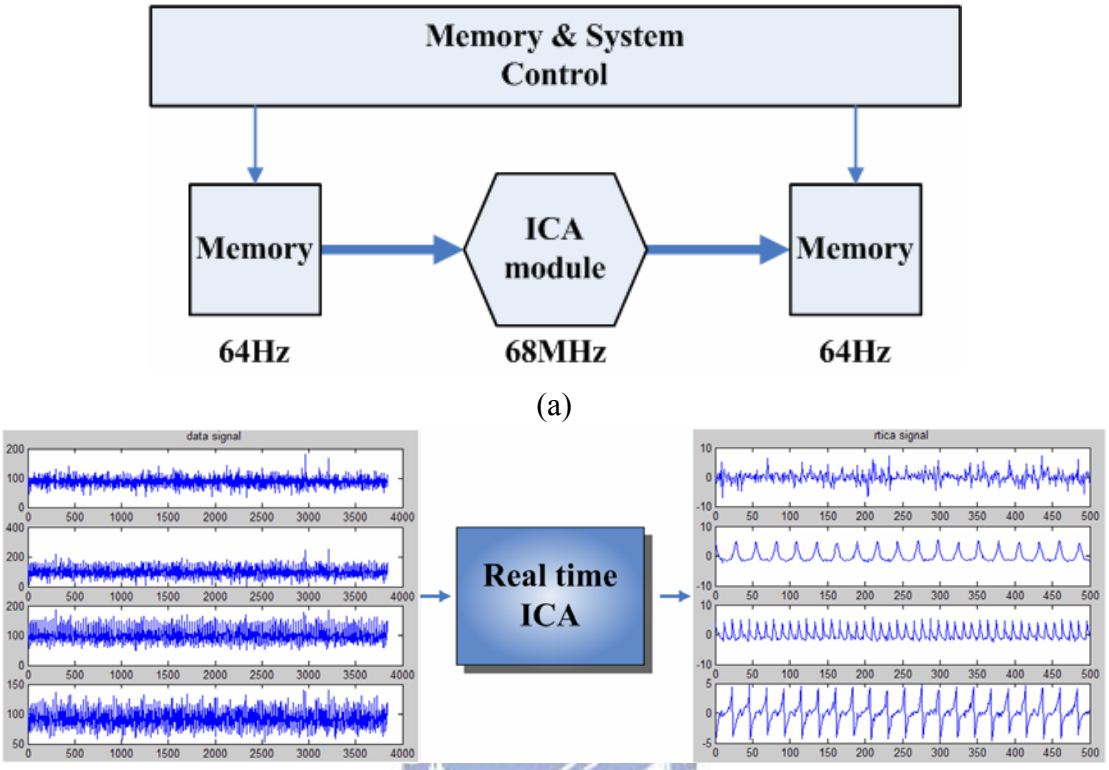


Fig. 2-13 (a) Top level hardware architecture and (b) Illustration of real-time systems.

In this section we discuss the architecture of the digital circuit of implementation of the algorithm. In overall IC, the main architecture of ICA required three parts in Fig. 2-14: Infomax operation circuit, memory control circuit, and I/O scheduler circuit. Arithmetic operation circuit consists of matrix operation circuit and non linear transform. Though a large number of computation in ICA arithmetic operation circuit, the memory access will be frequently and complex. By using efficient memory controller to optimize the memory scheduling and reduce the circuit power consumption. The concept of the non linear transform, it is hard to execute for real-time cause of the DSP processor will approximate the value by loop iteration. If we implement by FPGA, it will be developed with very low cost hardware, and

reduce unnecessary operation, and fast enough to execute for real-time.

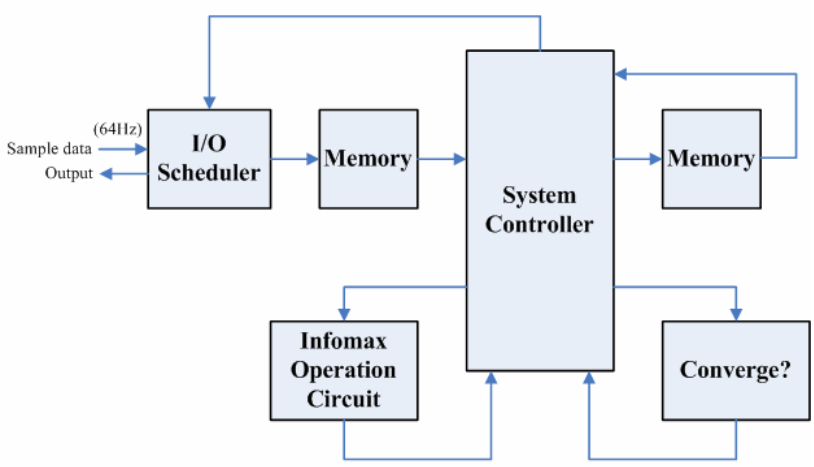


Fig. 2-14 ICA main system architecture.

2-3-3 Implementation of on-line ICA

(1) Implementation of Recursive Operation Circuit

The stability and high-precision are the properties of the recursive operation circuit. The errors may grow up with the growing iteration. In order to reduce errors in iteration, we develop a precision symmetrical non-linear piecewise look up table. Besides, we simplify the complex weight updating by deep pipeline design. And if the final weight coverage, the effective and fast matrix multiplier would be also designed in the system.

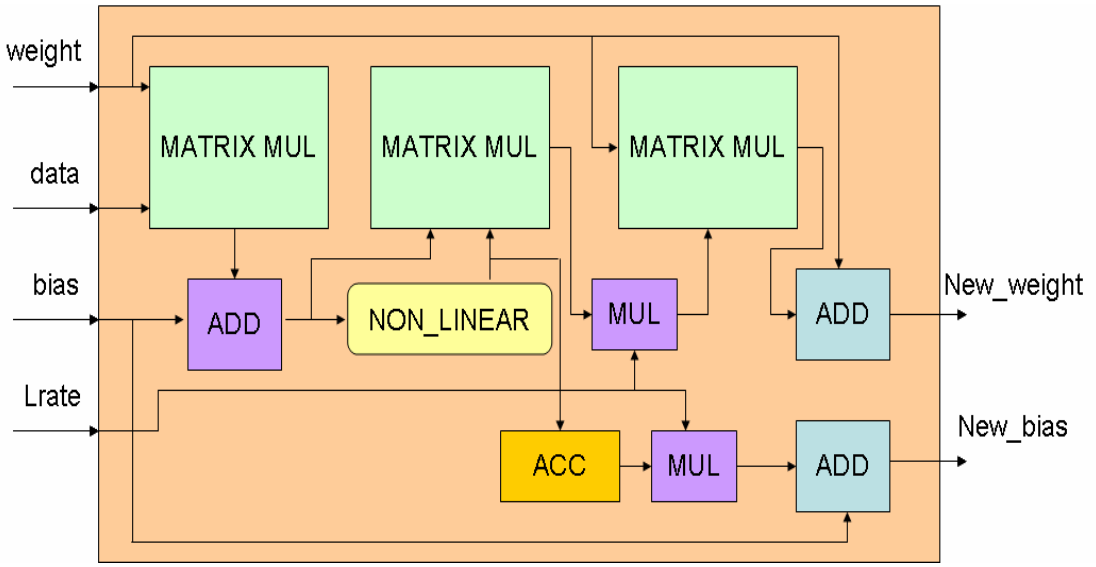


Fig. 2-15 Integrated computing unit.

Figure 2-15 shows the integrated computing unit, it can process 4*512 matrix operations. The unit includes three 4*4 matrix operation, one accumulation, and mathematics operator. All speed must be greater than 68MHz to achieve real time execution. Then we design the calculation module with pipeline. In order to on-line execution, we estimate the consumption of cycle must less than 8300 cycles when core speed is 68MHz and 128 times training. As a result, the unit consumes 8192 cycles to find a new weight with gradient information update. The expression of operation cycle is as follows:

$$operation\_cycle = \frac{core\_speed}{sample\_rate * iteration}. \quad (2.3)$$

(2) Implementation of system Controller

In Fig. 2-16, it includes asynchronous memory controller and ICA system controller. The ICA system controller mainly controls the data from system BUS and sent the control signal to various modules in computing. Block diagram is as follows:

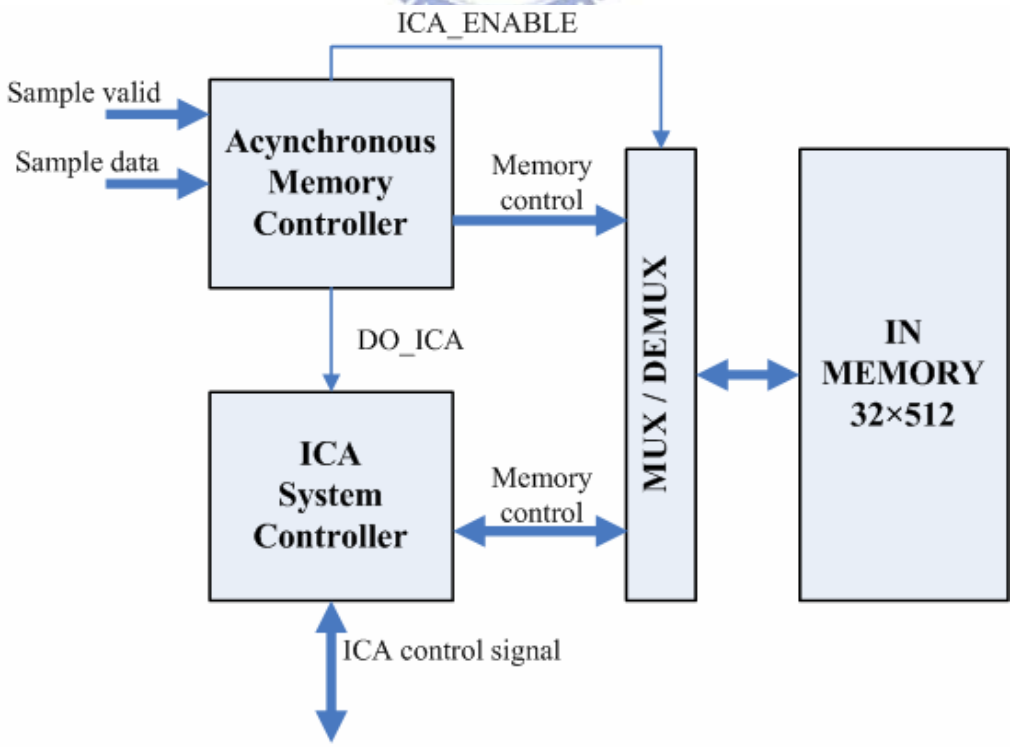


Fig. 2-16 Main controller architecture.

In asynchronous memory controller, because of the external data input frequency is different from internal in Fig. 2-17, thus we use asynchronous conversion to the same speed of system input. External data would be sent into the system memory by a similar way as interrupted. We also placed a data counter that can be judged by the amount of data, and the ICA system controller would send signals to the correct path.

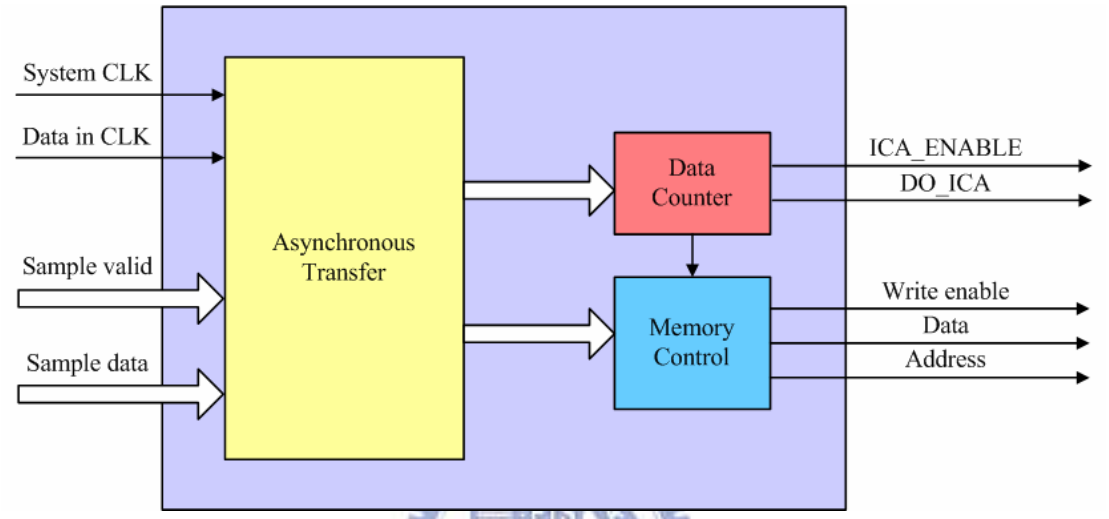


Fig. 2-17 Asynchronous memory controller circuit.

Following, ICA system controller controls various components which includes operate unit and memory unit. In order to avoid the control signal conflict, we divide the controller into two parts: system control signal and memory control signal. The control signals of system in Fig. 2-18, old weight, data have sent to operated in weight update module. And the result is given into converge module, new_weight is sent into weight buffer for initial parameter of next iteration. In the memory control, we use an effective memory scheduling. In DSP instruction scheduling, it may waste on memory space and the efficiency of execution because of the data hazard may cause by read after write (RAW). In Fig. 2-19, we use two recursive circuits and pipeline flow to reduce half amount of memory access. It is an effective memory scheduling.

In this design, we make the memory scheduling close together to reduce waiting time for hazard and to achieve on-line. In another hand, we also use enable signal in memory. All memory registers ports are controlled by the enable signal. When system needs access memory, the signal will trigger memory. We add a counter in memory bank, when specific memory block was accessed, the counter would count up. When the counter reached a critical value, controller will found that the memory has no demand for access. Therefore, the memory can into the power save mode. It can save power at memory access dynamically.

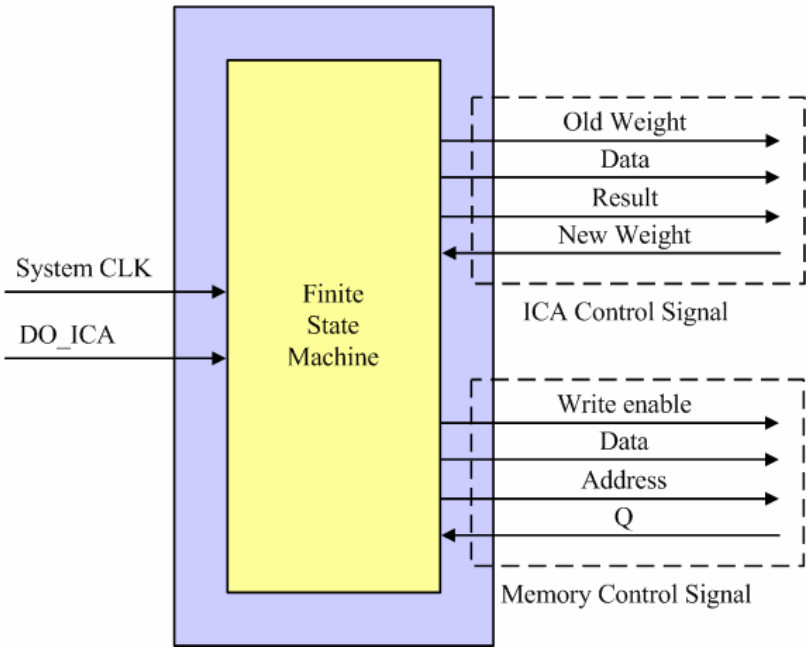


Fig. 2-18 System controller circuit.

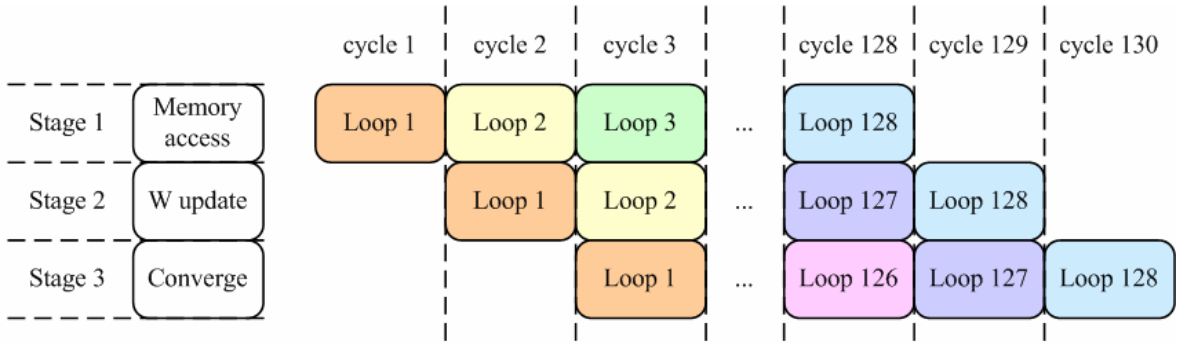


Fig. 2-19 System pipeline flow.

Refer to Fig. 2-20, DO_ICA which will drive the entire system controller, the detail of finite state machine show in Table 2-1. The state IDLE means that the system is receiving data, and can not do other operations. During state TRAINING, informax is mainly being computed. Each time the training will consume 8192 cycles, and it will jump to next state CONVERGE when computing end. The state will jump to DONE when the largest to 128 times of neural training. When state is DONE, the data can only be written into memory. So the overall core speed depends on sample rate. The whole design detail of the micro-controller shows as follows:

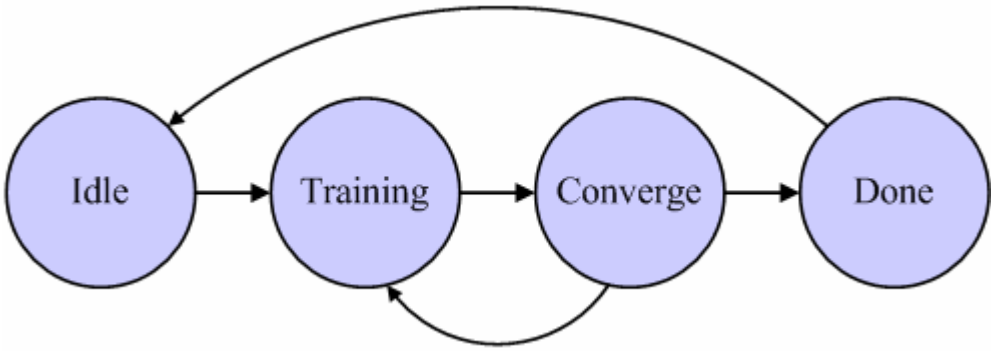


Fig. 2-20 Illustration of micro-controller.

Table 2-1 FSM of micro-controller

	State: IDLE	State: TRAINING	State: CONVERGE	State: DONE
Next state	TRAINING	If((&counter)& (&block)) CONVERGE else TRAINING	If(decision_step) If(&step) DONE else TRAINING else CONVERGE	If(DOICA) IDLE else DONE

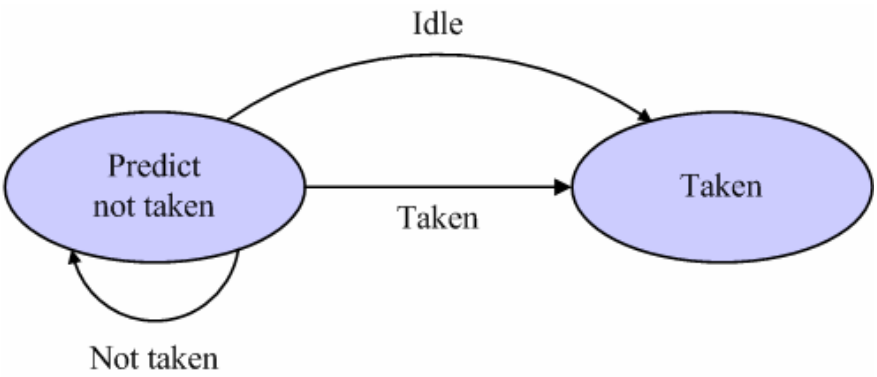


Fig. 2-21 Dynamic Branch Prediction.

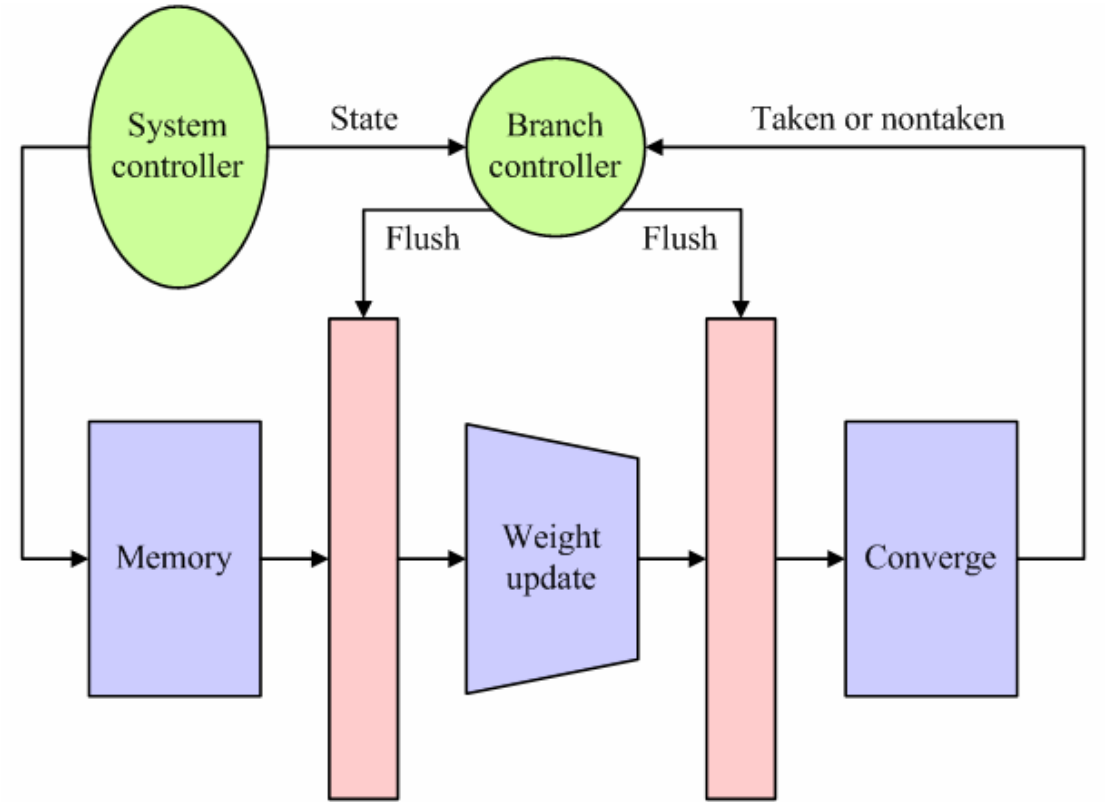


Fig. 2-22 Branch Controller and Flush Line.

In order to overlapping memory access time by pipeline, this thesis use dynamic branch prediction in Fig. 2-21. And add flush line to clear forward pipeline register in Fig. 2-22. According to the characteristic of ICA algorithm, the branch prediction can reduce memory access time effectively. Fig. 2-22 illustrate that we predict the branch always not taken, the branch controller would send a flush signal to clear forward pipeline register if taken happen and state at idle.

2-3-4 Introduction of FFT

A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and inverse DFT, in this work we do not use inverse FFT. FFT is based on structuring the DFT computation by forming smaller and smaller subsequences of the input sequence. Alternatively, we can consider dividing the output sequence into smaller and smaller subsequences in the same manner.

Let $x[0]$, ..., $x[N\text{-}1]$ be input sequence (they are complex number). The DFT is defined by the formula:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \qquad k = 0,1,\ldots, N-1. \tag{2.4}$$

Following will introduce some different kinds of FFT algorithms, from the viewpoint of real time processing, easy to implement, smaller chip area, Radix-2^2 algorithm is chosen in this work.

(1) Cooley-Tukey Algorithm (Radix-2 Algorithm)

By far the most common FFT is the Cooley-Tukey algorithm. This is a divide and conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1N_2$ into many smaller DFTs of sizes N_1 and N_2, along with $O(N)$ multiplications by complex roots of unity, traditionally called twiddle factors.

The most well-known use of the Cooley-Tukey algorithm is to divide the transform into two pieces of size $N/2$ at each step, and is therefore limited to power-of-two sizes, but any factorization can be used in general. This is called the Radix-2 algorithm.

Radix-2 algorithm divides output sequence X[k] into odd-numbered points and even-numbered points. The DFT of an N points discrete time sequence x[n] can be written as

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N-1} x[n] W_N^{nk}, \qquad k = 0,1,\ldots, N-1, \\
&= \sum_{n=0}^{N/2-1} x[n] W_N^{nk} + \sum_{n=0}^{N/2-1} x[n+\frac{N}{2}] W_N^{(n+\frac{N}{2})k} \\
&= \sum_{n=0}^{N/2-1} \left[x[n] + x[n+\frac{N}{2}] W_N^{\frac{N}{2}k} \right] W_N^{nk}.
\end{aligned} \tag{2.5}
$$

where $W_N^{nk} = e^{\frac{-j2\pi nk}{N}}$ $, k = 0,1,\ldots, N-1.$

We can now consider obtaining the odd-numbered frequency points, given by

$$
\begin{aligned}
X[2k+1] &= \sum_{N=0}^{n/2-1} \left[x[n] + x[n+\frac{N}{2}] W_N^{\frac{N}{2}(2k+1)} \right] W_N^{n(2k+1)} \\
&= \sum_{N=0}^{n/2-1} \left[x[n] - x[n+\frac{N}{2}] \right] W_N^n W_N^{n(2k)} \qquad , k = 0,1,\ldots (N/2)-1 \\
&= \sum_{N=0}^{n/2-1} \left[x[n] - x[n+\frac{N}{2}] \right] W_N^n W_{N/2}^{nk} \\
&= \mathrm{DFT}_{N/2} \left\{ (x[n] - x[n+\frac{N}{2}]) \cdot W_N^n \right\},
\end{aligned} \tag{2.6}
$$

in a similar way the even-numbered frequency points can be obtained as following:

$$
\begin{aligned}
X[2k] &= \sum_{N=0}^{n/2-1} \left[x[n] + x[n+\frac{N}{2}] W_N^{\frac{N}{2}(2k)} \right] W_N^{n(2k)} \\
&= \sum_{N=0}^{n/2-1} \left[x[n] - x[n+\frac{N}{2}] \right] W_{N/2}^{nk} \qquad , k = 0,1,\ldots (N/2)-1 \\
&= \mathrm{DFT}_{N/2} \left\{ (x[n] - x[n+\frac{N}{2}]) \right\}.
\end{aligned} \tag{2.7}
$$

According Eq. (2.6) and Eq. (2.7), we can get butterfly flow graph as Fig. 2-23.

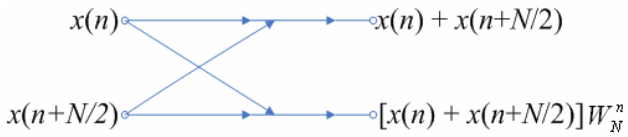


Fig. 2-23 Butterfly flow graph of Radix-2 algorithm.

Figure 2-24 shows a Flow graph of decomposition of an 8-point DFT computation into two 4-point DFT computations. We can decompose each 4-point DFT in a similar manner recursively, and we have full expansion butterfly

computation flow graph of 8-point DFT as Fig. 2-25 shows.

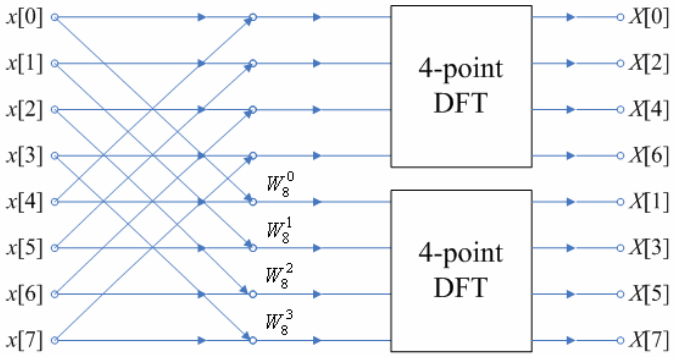


Fig. 2-24 Flow graph of 8-point DFT computation into two 4-point DFT computations.

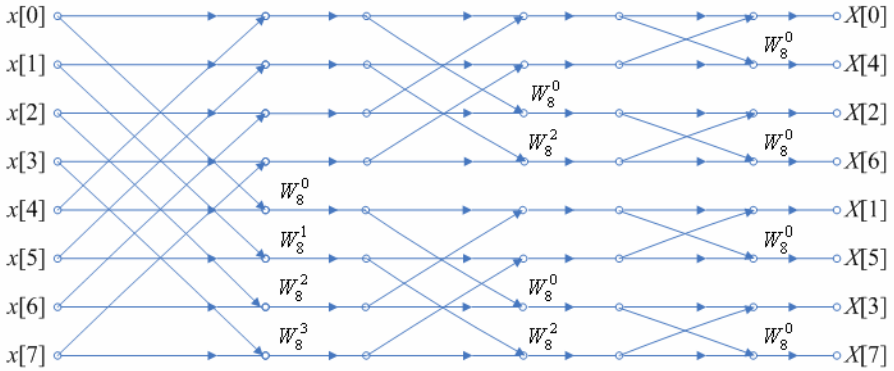


Fig. 2-25 Flow graph of complete decimation-in-frequency decomposition of an 8-point DFT computation.

(2) Radix-4 Algorithm

Similarly, Radix-4 decimation-in-frequency algorithm representation can be derived from Eq. (2.4) into Eq. (2.8).

$$X[k]=\sum_{n=0}^{N-1}x[n]W_N^{nk},\qquad k=0,1,\ldots,N-1,$$

$$=\sum_{n=0}^{N/4-1}x[n]W_N^{nk}+\sum_{n=0}^{N/4-1}x[n+\frac{N}{4}]W_N^{(n+\frac{N}{4})k}+\sum_{n=0}^{N/4-1}x[n+\frac{N}{2}]W_N^{(n+\frac{N}{2})k}+\sum_{n=0}^{N/4-1}x[n+\frac{3N}{4}]W_N^{(n+\frac{3N}{4})k} \quad (2.8)$$

$$=\sum_{n=0}^{N/4-1}\left[x[n]+x[n+\frac{N}{4}]W_N^{\frac{N}{4}k}+x[n+\frac{N}{2}]W_N^{\frac{N}{2}k}+x[n+\frac{3N}{4}]W_N^{\frac{3N}{4}k}\right]W_N^{nk},$$

where $W_N^{nk}=e^{\frac{-j2\pi nk}{N}}$ $,k=0,1,\ldots,N-1$

Now frequency points can be divided into four groups which are $X[4k]$, $X[4k+1]$, $X[4k+2]$ and $X[4k+3]$.

$$X[4k]=\sum_{n=0}^{N/4-1}\left[x[n]+x[n+\frac{N}{4}]+x[n+\frac{N}{2}]+x[n+\frac{3N}{4}]\right]W_N^{n(4k)}$$
$$=\sum_{n=0}^{N/4-1}\left[x[n]+x[n+\frac{N}{4}]+x[n+\frac{N}{2}]+x[n+\frac{3N}{4}]\right]W_{N/4}^{nk}\quad \mathrm{k}=0,1,...,\mathrm{N}/4-1 \qquad (2.9)$$
$$=\mathrm{DFT}_{\mathrm{N}/4}\left\{x[n]+x[n+\frac{N}{4}]+x[n+\frac{N}{2}]+x[n+\frac{3N}{4}]\right\},$$

$$X[4k+1]=\sum_{n=0}^{N/4-1}\left[x[n]-j\cdot x[n+\frac{N}{4}]-x[n+\frac{N}{2}]+j\cdot x[n+\frac{3N}{4}]\right]W_N^nW_N^{n(4k)}$$
$$=\sum_{n=0}^{N/4-1}\left[x[n]-j\cdot x[n+\frac{N}{4}]-x[n+\frac{N}{2}]+j\cdot x[n+\frac{3N}{4}]\right]W_N^nW_{N/4}^{nk}\quad \mathrm{k}=0,1,...,\mathrm{N}/4-1 \qquad (2.10)$$
$$=\mathrm{DFT}_{\mathrm{N}/4}\left\{\left(x[n]-j\cdot x[n+\frac{N}{4}]-x[n+\frac{N}{2}]+j\cdot x[n+\frac{3N}{4}]\right)W_N^n\right\},$$

$$X[4k+2]=\sum_{n=0}^{N/4-1}\left[x[n]-x[n+\frac{N}{4}]+x[n+\frac{N}{2}]-x[n+\frac{3N}{4}]\right]W_N^nW_N^{n(4k)}$$
$$=\sum_{n=0}^{N/4-1}\left[x[n]-x[n+\frac{N}{4}]+x[n+\frac{N}{2}]-x[n+\frac{3N}{4}]\right]W_N^nW_{N/4}^{nk}\quad \mathrm{k}=0,1,...,\mathrm{N}/4-1 \qquad (2.11)$$
$$=\mathrm{DFT}_{\mathrm{N}/4}\left\{\left(x[n]-x[n+\frac{N}{4}]+x[n+\frac{N}{2}]-x[n+\frac{3N}{4}]\right)W_N^n\right\},$$

$$X[4k+3]=\sum_{n=0}^{N/4-1}\left[x[n]+j\cdot x[n+\frac{N}{4}]-x[n+\frac{N}{2}]-j\cdot x[n+\frac{3N}{4}]\right]W_N^nW_N^{n(4k)}$$
$$=\sum_{n=0}^{N/4-1}\left[x[n]+j\cdot x[n+\frac{N}{4}]-x[n+\frac{N}{2}]-j\cdot x[n+\frac{3N}{4}]\right]W_N^nW_{N/4}^{nk}\quad \mathrm{k}=0,1,...,\mathrm{N}/4-1 \qquad (2.12)$$
$$=\mathrm{DFT}_{\mathrm{N}/4}\left\{\left(x[n]+j\cdot x[n+\frac{N}{4}]-x[n+\frac{N}{2}]-j\cdot x[n+\frac{3N}{4}]\right)W_N^n\right\}.$$

Using Eq. (2.9) to Eq. (2.12) can derive basic Radix-4 FFT butterfly flow graph which is shown in Fig. 2-26.

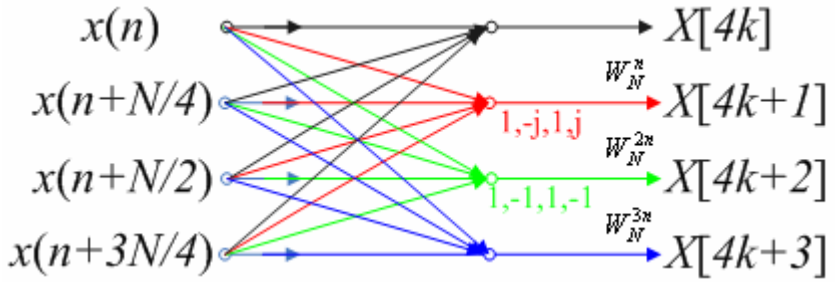


Fig. 2-26 Basic butterfly flow graph of a Radix-4 FFT

(3) Radix-2^2 Algorithm

Implementation of FFT by Radix-4 algorithm will cause higher complexity but more computation in each stage. Alternatively implementation through Radix-2 algorithm has the simplest structure but fewer computations than the former. In order to combine benefits of both Radix-4 algorithm and Radix-2 algorithm, Radix-4 algorithm can be simplified into Radix-2^2 algorithm. A new representation of decimation-in-time decomposition of an N points discrete time sequence $x[n]$can be derived by substituting $n = \frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3$, $k = k_1 + 2k_2 + 4k_3$ in Eq. (2.4).

$$\begin{aligned} X[k_1 + 2k_2 + 4k_3] &= \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1} x[\frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3]W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} \\ &= \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\left\{ B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2 + n_3)W_N^{(\frac{N}{4}n_2+n_3)k_1} \right\} W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} \end{aligned} \quad (2.13)$$

where $B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2 + n_3) = x(\frac{N}{4}n_2 + n_3) + (-1)^{k_1}(\frac{N}{4}n_2 + n_3 + \frac{N}{2})$.

Decomposing the composite twiddle factor and observe that

$$\begin{aligned} W_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} &= W_N^{Nn_2k_3} W_N^{\frac{N}{4}n_2(k_1+2k_2)} W_N^{n_3(k_1+2k_2)} W_N^{4n_3k_3} \\ &= (-j)^{n_2(k_1+2k_2)} W_N^{n_3(k_1+2k_2)} W_N^{4n_3k_3}. \end{aligned} \quad (2.14)$$

Substituting Eq. (2.14) in Eq. (2.13) and expand the summation with index n_2. After simplification we have a set of 4 DFTs of length N/4,

$$X[k_1 + 2k_2 + 4k_3] = \sum_{n_3=0}^{\frac{N}{4}-1}[H(k_1, k_2, n_3)W_N^{n_3(k_1+2k_2)k_1}]W_{N/4}^{n_3k_3} \quad (2.15)$$

where H(k1, k2, n3) is expressed in Eq. (2.16).

$$H(k1,k2,n3)=\overbrace{\underbrace{\left\{x[n_3]+(-1)^{k_1}x[n_3+\frac{N}{2}]\right\}}_{BF\ I}+(-j)^{(k_1+2k_2)}\underbrace{\left\{x[n_3+\frac{N}{4}]+(-1)^{k_1}x[n_3+\frac{3N}{4}]\right\}}_{BF\ I}}^{BF\ II}. \quad (2.16)$$

Equation (2.16) reveals that $H(k1,k2,n3)$ can be separated into a two stage butterfly unit which is corresponding to BF I and BF II. Let

$$\begin{aligned} m1 &= x(n)+x(n+N/2) \\ m2 &= x(n+N/4)+x(n+3N/4) \\ m3 &= x(n)-x(n+N/2) \\ m4 &= x(n+N/4)-x(n+3N/4). \end{aligned} \quad (2.17)$$

Substituting Eq. (2.17) in Eq. (2.9) through Eq. , we have

$$\begin{aligned} X[4k] &= DFT_{N/4}\{m1+m2\} \\ X[4k+1] &= DFT_{N/4}\{(m3-j\cdot m4)W_N^n\} \\ X[4k+2] &= DFT_{N/4}\{(m1-.m2)W_N^{2n}\} \\ X[4k+3] &= DFT_{N/4}\{(m3+j\cdot m4)W_N^{3n}\} \end{aligned} \quad (2.18)$$

Observing Eq. (2.18), Radix-4 butterfly unit can be transformed into Radix-2^2 butterfly unit which basically composed by Radix-2 butterfly unit. Fig. 2-27 reveals flow graph of Radix-2^2 decimation-in-frequency.

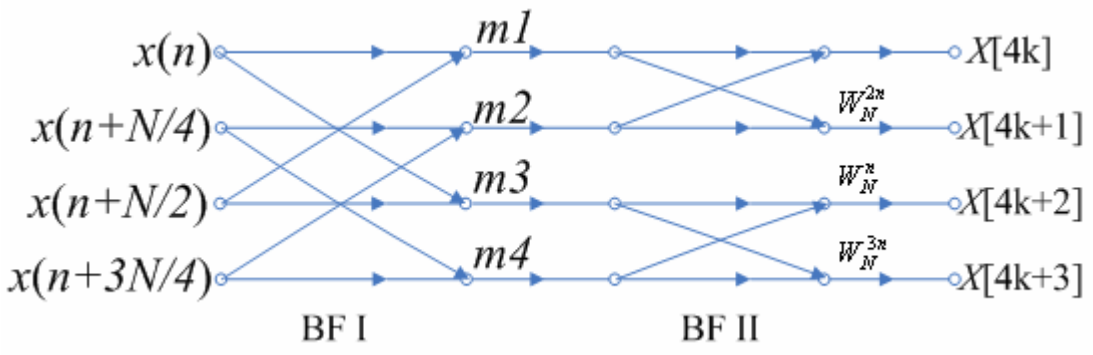


Fig. 2-27 Basic butterfly flow graph of a Radix-2^2 FFT

We can fully expand a 16-point DFT computation by recursively using basic butterfly unit. The fully expansion flow graph is shown as below. Radix-2^2 algorithm has the same computational complexity as Radix-4 algorithm but the butterfly structure is as simple as Radix-2 algorithm [20].

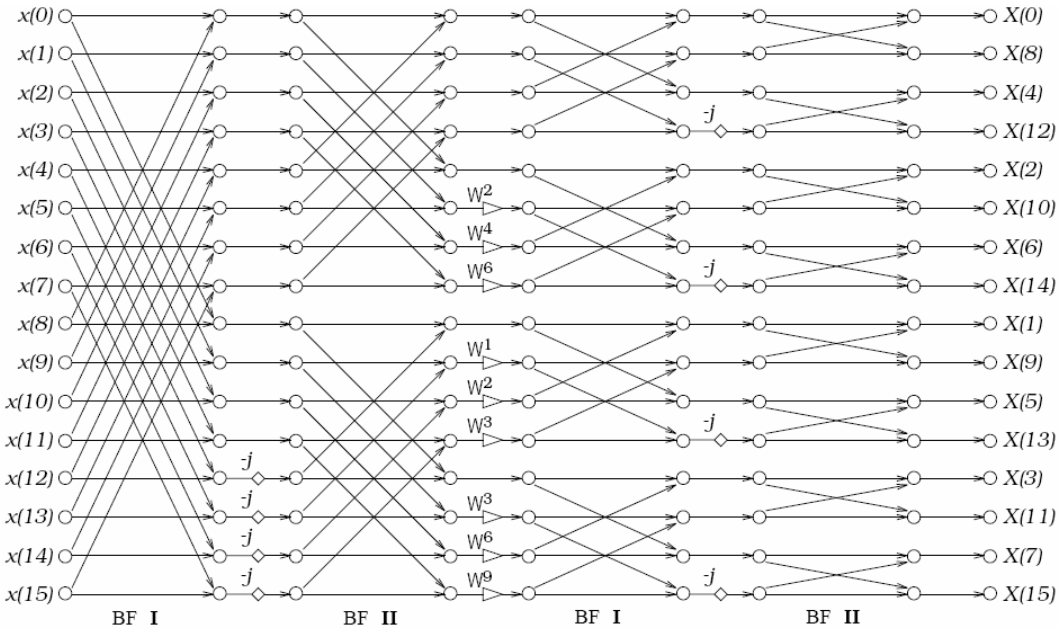


Fig. 2-28 Flow graph of complete decimation-in-frequency decomposition using Radix-2^2 algorithm of a 16-point DFT computation.

2-3-5 Different Structure of FFT Nowadays

Generally speaking, pipeline-based FFT design is a specified class for DFT computation utilizing fast algorithms. It is characterized with real-time, non-stopping processing as the data sequence passing the processor. There are two main categories of implementation of pipeline-based FFT. They are: “delay-commutator” and “delay-feedback” respectively. Nevertheless, delay-feedback has higher throughput but more hardware usage. In order to satisfy real time requirement, delay-feedback is chosen in this design.

(1) Introduction of Single-path Delay Feedback

Single-path Delay Feedback (SDF) uses the registers by storing the butterfly output in feedback shift registers. In a butterfly unit, registered input does not compute with incoming input until feedback shift register is full. Then computation result of butterfly unit is passed to next stage. Iteratively repeat the computation process just mentioned, we will have complete FFT result. SDF naturally has simpler

architecture than delay-commutator, also has less memory requirement and higher utility on both storage element and complex number multiplier.

(2) Radix-2 SDF Structure

Radix-2 Single-path Delay Feedback (R2SDF) [21] uses the registers more efficiently by storing the butterfly output in feedback shift registers. A single data stream goes through the multiplier at every stage. It has same number of butterfly units and multipliers as in Radix-2 multiple delay-commutator approach, but with much reduced memory requirement: N-1 registers. Its memory requirement is minimal. Fig. 2-29 shows a 16 points Radix-2 SDF FFT structure.

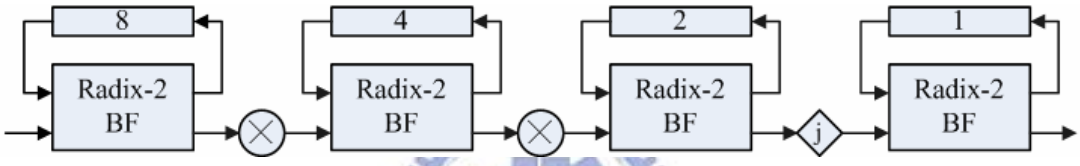


Fig. 2-29 16 points Radix-2 SDF structure.

(3) Radix-4 SDF Structure

Radix-4 Single-path Delay Feedback was proposed as a radix-4 version ofR2SDF, thus it has 4 outputs for each butterfly unit. Three outputs are stored in feedback shift registers. Fig. 2-30 shows a 256 points Radix-4 SDF FFT structure.

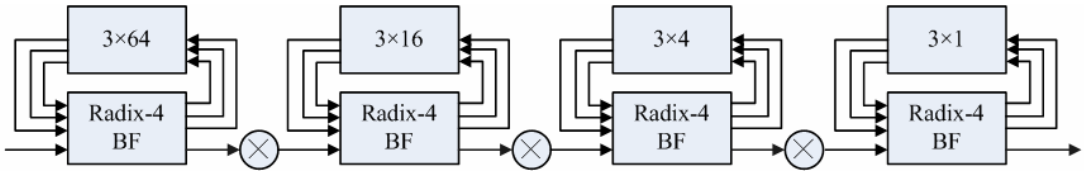


Fig. 2-30 256 points Radix-4 SDF structure.

(4) Radix-2^2 SDF Structure

Radix-2^2 SDF is similar to Radix-2 SDF, the only differences is Radix-2^2 SDF needs 2 butterfly unit in single stage. For the datum begin calculated are all complex numbers, multiplexer is used to deal with coefficient –j by data switch and sign

inversion. According to Eq. (2.15), a single stage butterfly unit can be categorized into BF I and BF II, Fig. 2-31 is an N point Radix-2^2 SDF FFT structure.

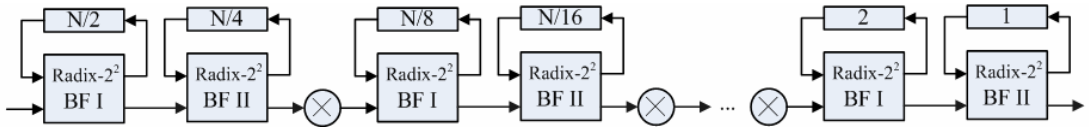


Fig. 2-31 N point Radix-2^2 SDF FFT structure.

2-3-6 Implementation of Radix-2^2 FFT

A modified SDF structure is presented in this section. By adding memory scheduler to schedule input sequence for each stage, the architecture of FFT unit in this design is revealed in Fig. 2-32. The data precision and twiddle factor precision are both 16-bit, in order to save the cost of CPU data movement, the output sequence is in natural order instate of bit reverse order.

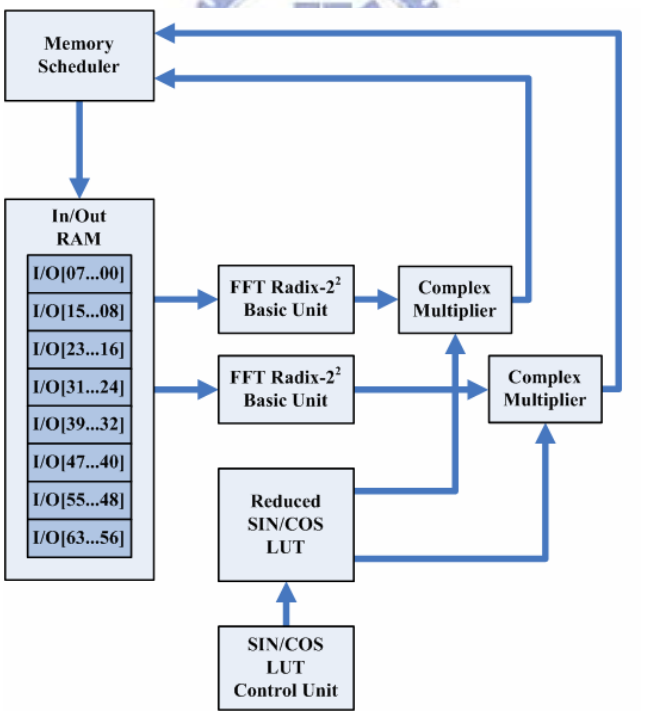


Fig. 2-32 The architecture of modified radix-2^2 FFT.

The design of this FFT eventually takes only 132 cycles to finish its computation including natural order output which outperforms original version (bit reverse order) about 9%.

(1) Implementation of FFT Radix-2^2 Basic Butterfly

Traditionally, radix-2^2 structure uses only one BF I and BF II as its basic arithmetic unit, it brings the benefit of smaller area cost but pays the penalty for extra FIFO depth and delay control logics. Also traditional radix-2^2 structure takes four cycles to complete four points BF I and BF II in single stage. In this work, we combine BF I and BF II as the basic arithmetic unit. Since there are totally only three levels addition in BF I and BF II, it is reasonable making BF I and BF II to be finished in single cycle. Refer to Fig. 2-27 the basic butterfly unit in this work is as follows:

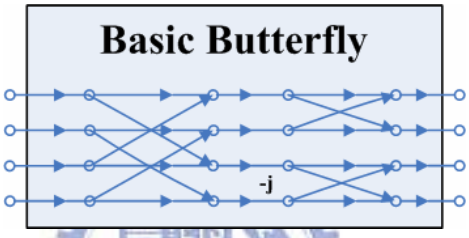


Fig. 2-33 Basic butterfly of modified radix-2^2 FFT.

The comparison for the 16-bit basic butterfly after synthesis in FPGA (Altera Cycleon II) of traditional SDF radix-22 FFT and this work is list in Table 2-2.

Table 2-2 Hardware cost comparison of traditional radix-2^2 FFT and this work.

	Traditional (BF I + BF II)	This work
Total logic elements	176	288
Total combinational functions	176	288
Dedicated logic registers	128	128
Total registers	128	128
Execution cycle per stage	4	1

Concerning with high performance, we use fixed point as the data representation instead of floating point. Consequently, it takes extra bits for precision but makes a

faster multiplier than floating point multiplier.

(2) Implementation of Reduced Sinusoidal Lookup Table

The twiddle factors are all 16-bit and represented in fixed point format (2-bit for integer part, 14-bit for fractional part). Observing all 64 twiddle factors used in a 64-points radix-2^2 FFT, we needs only half amount of 64 twiddle factors due to its sinusoidal symmetric nature. In practice, we use a 2-ports read only memory (ROM) to build up this lookup table for higher throughput. The structure of this reduced look up table is shown below.

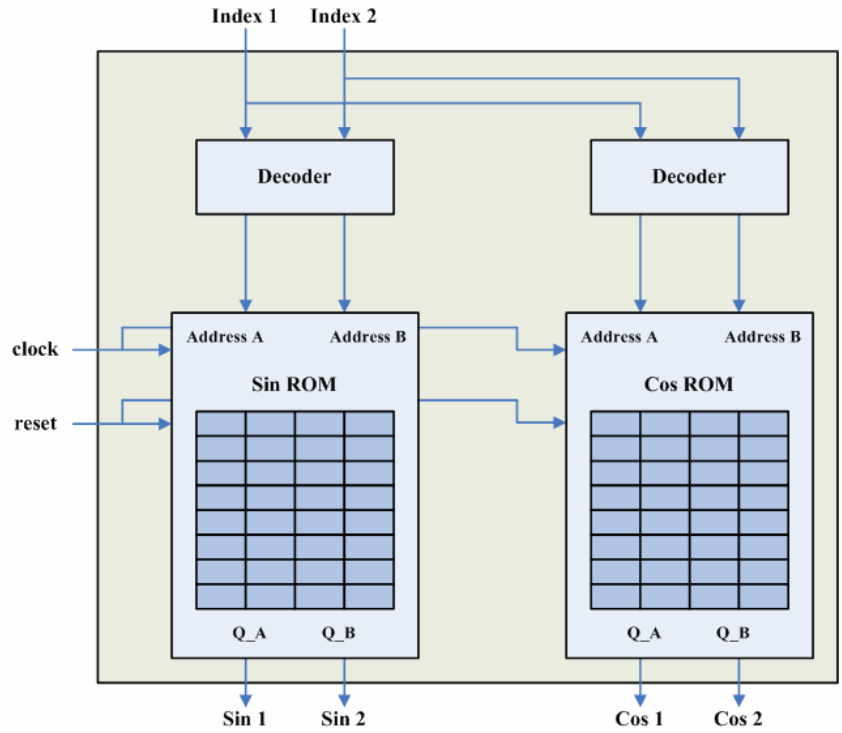


Fig. 2-34 Reduced Sinusoidal look up table.

(3) Implementation of Complex Multiplier

The structure of the complex multiplier in Fig. 2-32 is depicted in Fig. 2-35. The internal multipliers are 16-bit multipliers. For the purpose of hardware reuse (of basic butterfly) the doubled size of the result must be truncated. In order to keep a high

precision during computation, truncation is done after addition. Due to FFT takes the outputs of ICA as its input, and the size of ICA outputs is 8-bit each channel, extra 8-bit of the basic butterfly can be used to keep the precision information of intermediate computation.

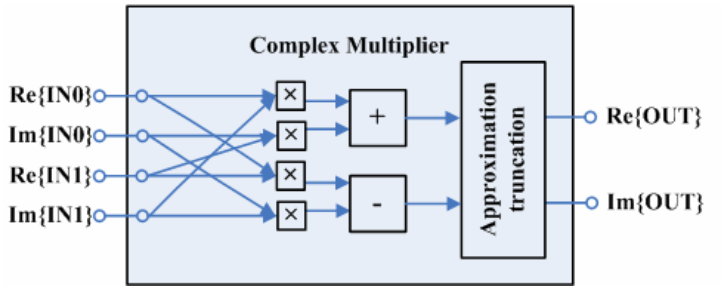


Fig. 2-35 Structure of the complex multiplier.

A dynamic size of truncation window is used in approximation truncation. The most significant bit is used as sign extension bit. The bit under the least significant bit of truncation window is used to round off the value in window. The strategy of the approximation truncation is shown in Fig. 2-36.

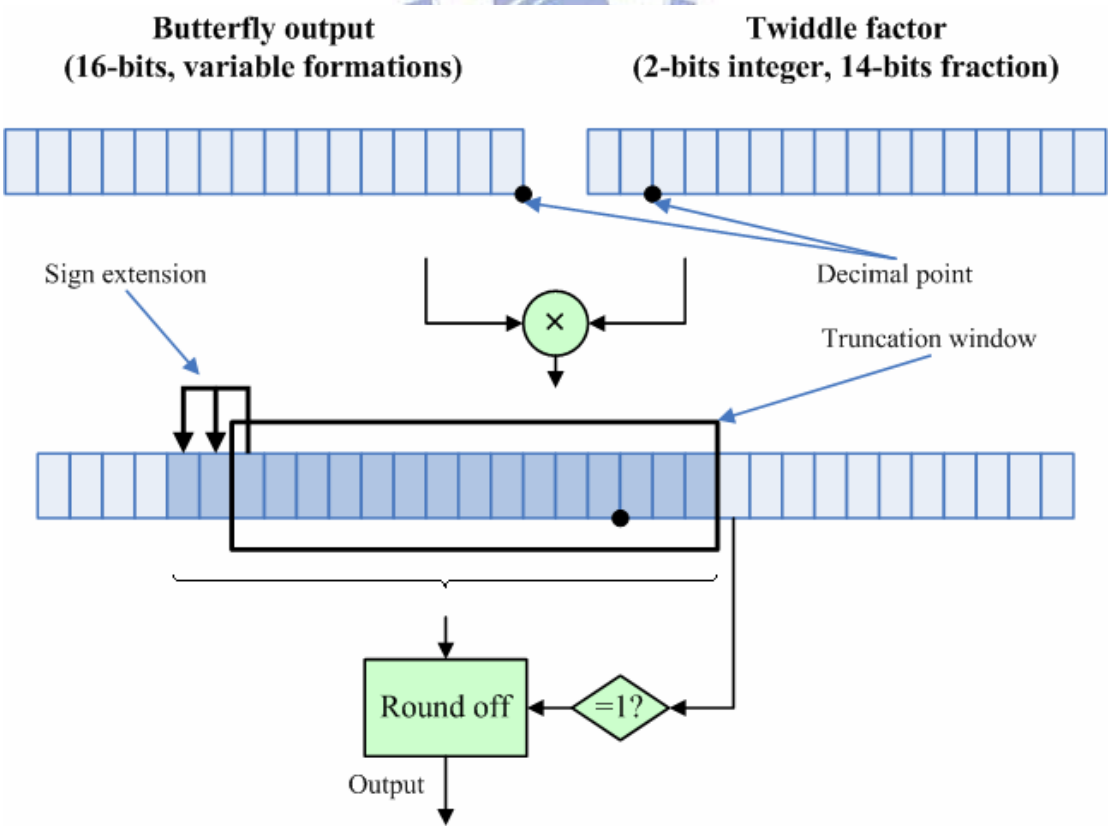


Fig. 2-36 Approximation truncation strategy.

(4) Implementation of Memory Scheduler

For the sake of limitation of logic element on FPGA, 64 points are stored in random access memory (RAM). In stead of single 64-entry RAM, we use eight 2-ports 8×16 bits RAMs for more accessible entries simultaneously.

The fully extended 64-points modified radix-2^2 FFT is shown in Fig. 2-37. The execution of the 64-points modified radix-2^2 FFT mainly can be divided into five stages. The first stage is receiving stage, which waits for input and input valid signal. According to Fig. 2-37, it can be told that the middle three stages is the recursive structure of 64-points FFT. In the fourth stage, it rearranges bit reverse order of output sequence to natural order. Memory scheduler deals with the control signal and schedule of eight RAMs in these five stages.

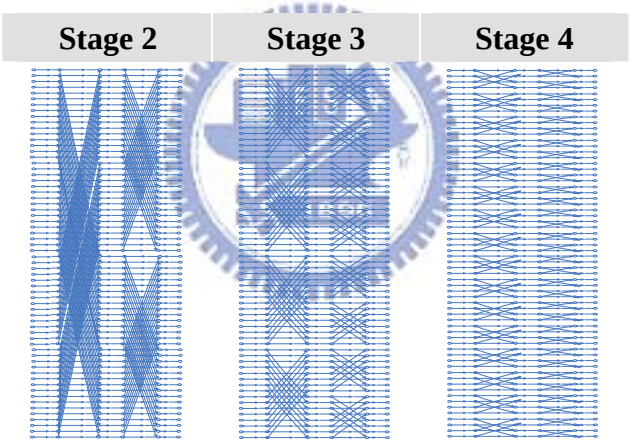


Fig. 2-37 Fully extended 64-points modified radix-2^2 FFT.

Observing indices of the points of the basic butterfly in each stage, we can find there exist some special distance between them which is called “input distance” for short. Once the first four inputs’ index has been decided, the scheduler needs to increment all four indices by one for the next execution of basic butterfly. The index increment between execution of basic butterfly is called “vertical increment” since it represents vertical movement in fully extended FFT structure like Fig. 2-37 shows. In some stage the vertical increment may have glitch (increments more then one) after a

period of stable increment by one.

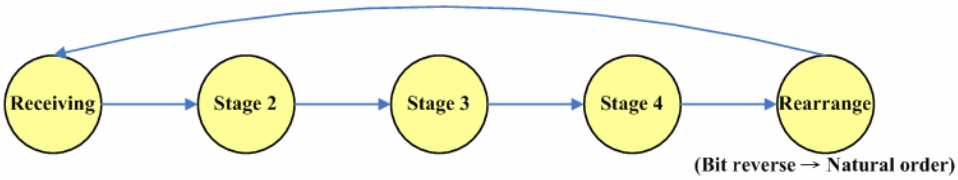


Fig. 2-38 FSM in FFT.

The architecture of memory scheduler is depicted in Fig. 2-39. According to the state transition in Fig. 2-38 the stage controller controls all control logic in memory scheduler. Write data controller selects the data from system BUS while in "receiving" state, during stage 2, 3 and 4 it takes truncated intermediate result after basic butterfly and complex multiplier.

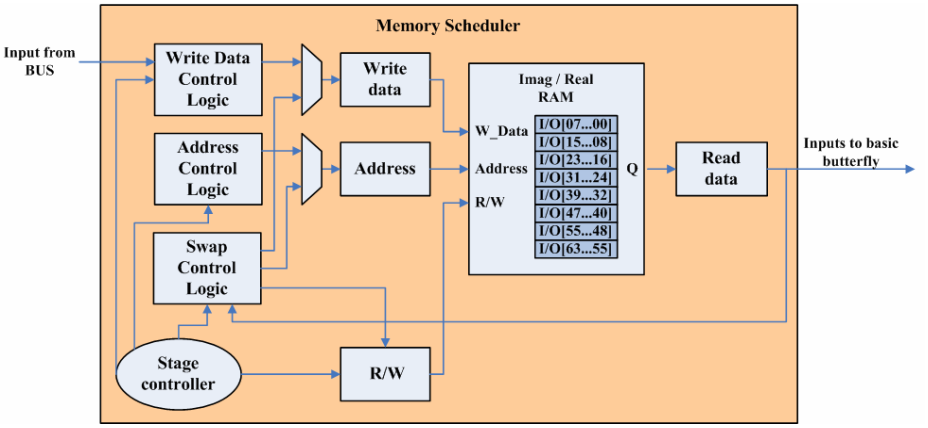


Fig. 2-39 Architecture of memory scheduler.

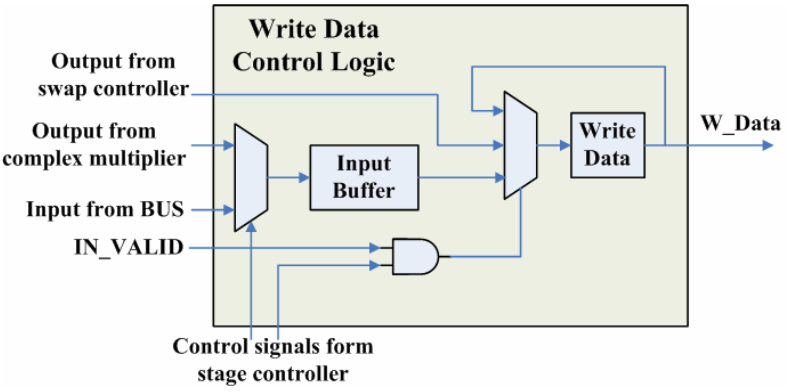


Fig. 2-40 Write data controller.

The address controller increments address by one in "receiving" stage, and computes corresponding value in stage 2, 3 and 4 according to input distance and vertical increment which is also controlled by stage controller.

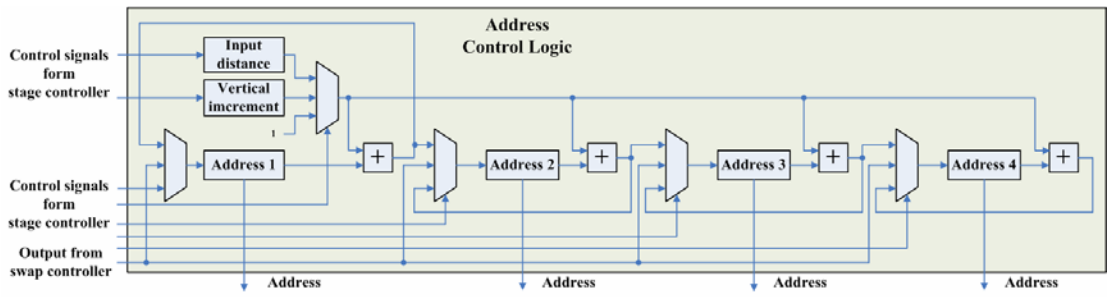


Fig. 2-41 Address controller.

Swap controller plays an important role in “rearrange” stage. Since each bit-reverse address corresponds to exactly one natural order address, swapping memory contents of this pair of addresses would get us natural order output sequence. When FFT steps into “rearrange” stage, stage controller sends an enable signal to swap controller to trigger address counter to increase. Next, we check whether the binary representation of the address is symmetric or not, if it is, that means the corresponding natural order address is itself and swapping is unnecessary. Otherwise, there must exist one corresponding natural address, swap controller then issues a read request with this pair of addresses to RAM. After receiving the contents of this pair of addresses, controller swaps the contents and issues a write request with addresses to RAM. Note, the logic value of symmetric taken is delayed to synchronize with read latency of the RAM.

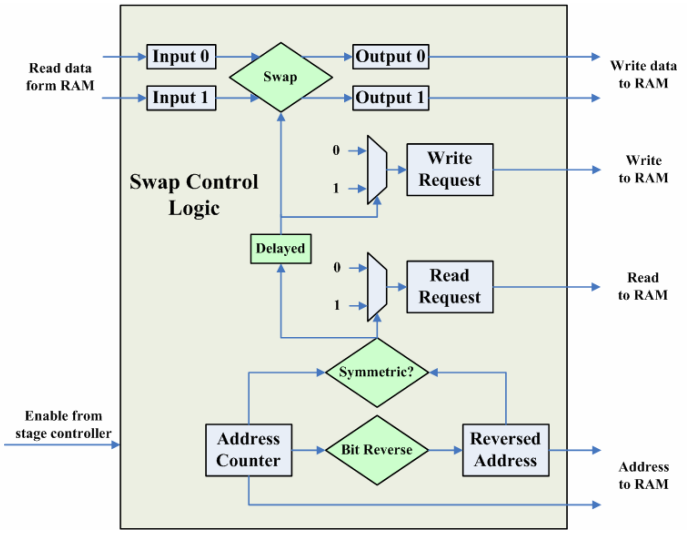


Fig. 2-42 Swap controller.

2-4 VGA Controller

In this work, the resolution of display is 640×408, and the refresh rate is 60Hz. In order to contain the R, G, B information (each color is 10-bit) for all 307,200 pixels in single frame, we need at least 115,200 bytes storage space for data registration. Because the hardware resource on FPGA is limited, part of on-board SDRAM is split as display buffer for VGA. For the sequential reading for every pixel, VGA needs 307,200 reads from SDRAM. Since VGA does not write back to system and the only task it does is read, there is no reason for VGA to connect to SDRAM through BUS interface.

The SDRAM has four banks and is shared by the VGA and DSP units, thus four internal read/write channels for independent control on each bank is adopted. For the VGA read timing, R, G, B information must ready in single clock cycle, that is, 30-bit data is read in single clock cycle. Thus two banks (each bank has data width 16-bit, in other words, 32-bit data width for two banks) are allocated for VGA display buffer.

2-4-1 Introduction VGA Controller

A LCD is composed of a two dimensions pixel array, the refresh of a frame starts from top left to bottom right of a display. For the VGA video signals, two synchronization signals are required, they are, horizontal and vertical synchronization respectively.

(1) The Basics of VGA Signals

The two synchronization signals in this design are: horizontal synchronization signal and the vertical synchronization signal. Both of these signals have similar waveforms as depicted in Fig. 2-43.

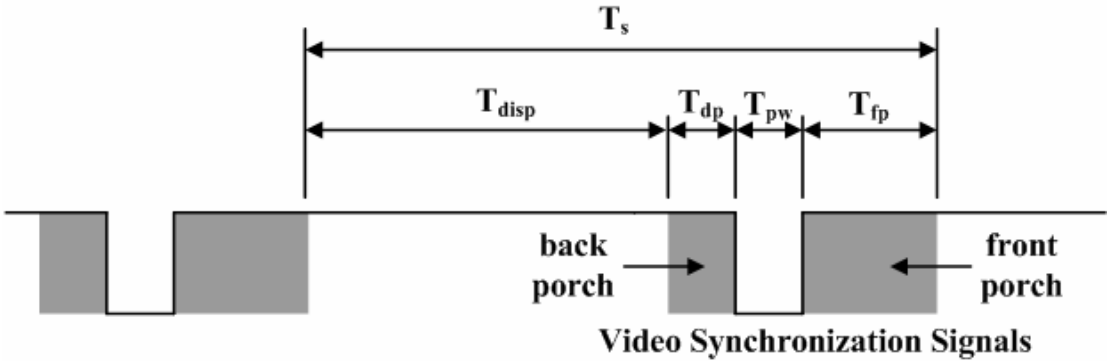


Fig. 2-43 Basic format for synchronization signals.

The global period of the synchronization signal is T_s, as a time reference for the synchronization signals mentioned above. T_{disp} is the time interval at which the video signal is visible on the screen. All synchronization pulses have a unique duration of T_{pw}. In a LCD the pulse causes the pixel electrode controller for the horizontal or vertical scan to be reset, causing the pixel scan to return to the left (for the horizontal sync) or to the top (for the vertical sync) signals. The times just before and immediately following the synchronization pulses are normally not shown on the display. They are called the "back porch" and the "front porch". The time durations of "back porch" and "front porch" are respectively called: T_{bp} and T_{fp}. The relation of the pixel scan and the synchronization signals is well illustrated by Fig. 2-44. This figure also illustrates the coordinates of a number of screen pixels for a 640×480 video signal when using a 25.175MHz clock is reported.

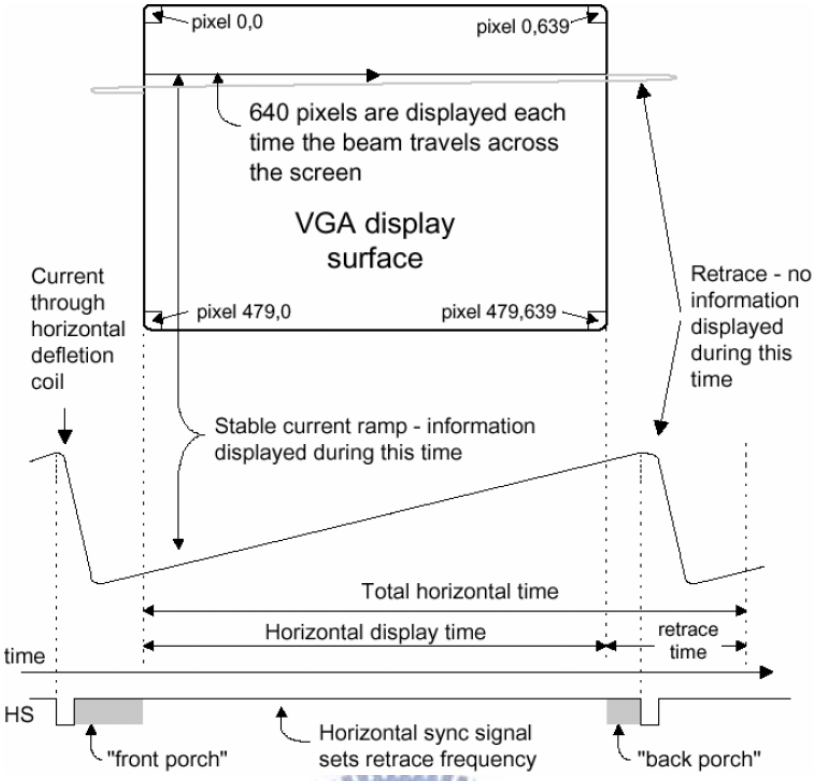


Fig. 2-44 Relationship of display of a horizontal line and the horizontal synchronization signal.

(2) Timing Information of 640×480×60 VGA Signals

The timing characteristics for the 640×480 horizontal synchronization signal are given in Table 2-3 below.

Table 2-3 Horizontal synchronization signal characteristics for 640×480 video generation, using a 25.175MHz clock.

Symbol	Name	Time duration	Number of 25.2MHz clock periods	25.2MHz Clock periods from time reference
T_{pw}	Pulse Width	3.81μs	96	95
T_{fp}	Front Porch	1.91μs	48	143
T_{disp}	Display Time	25.42μs	640	783
T_{bp}	Back Porch	635.52ns	16	799
T_s	Sync pulse time	31.77μs	800	799

The timing characteristics for the 640×480 vertical synchronization signal are given in Table 2-4 below. The 640×480 video signals use a frame time Ts of 16.6824ms. This corresponds to a frame rate of 60 Hz.

Table 2-4 Vertical synchronization signal characteristics for 640×480 video generation, using a 25.175MHz clock.

Symbol	Name	Time duration	Number of 25.2MHz clock periods	25.2MHz Clock periods from time reference
T_{pw}	Pulse Width	63.55μs	2	1
T_{fp}	Front Porch	1.02ms	32	33
T_{disp}	Display Time	15.25ms	480	513
T_{bp}	Back Porch	349.54μs	11	524
T_s	Sync pulse time	16.7ms	525	524

2-4-2 Implementation of VGA Controller

In a 60Hz frame rate 640×480 display the pixel clock frequency is 25.175 MHz. This controller contents two counters: H_Count and V_Count indicating horizontal position and vertical position of pixels. In the VGA controller module, the output horizontal synchronization signal (oVGA_H_SYNC) can be generated based on H_Count which increases every clock period. When H_Count reaches the last cycle of the horizontal synchronization signal (at cycle 799), counter will be reset. In this way, one oVGA_H_SYNC signal period is equal to 800 (H_SYNC_TOTAL represents the constant “800”) cycles. As can be seen in Fig. 2-46, the oVGA_H_SYNC can be generated based on the synchronization timing depicted in Fig. 2-45. All necessary parameters are listed in Table 2-5. At every oVGA_H_SYNC pulse, it triggers the V_Count to increase by generating a pulse, which is needed for the generation of the vertical synchronization signal. The vertical synchronization signal oVGA_V_SYNC can be generated in a similar way as

oVGA_H_SYNC. Once V_Count equals to 524, it will be reset to 0. V_Count drives oVGA_V_SYNC in the same way as oVGA_H_SYNC does. When oVGA_BLANK is logic zero indicates the generated video signal is ignored, otherwise generated video is composed by i_RED, i_GREEN and i_BLUE read from SDRAM. oRequest signal is treated as read request to the SDRAM, thus it must be asserted two cycle before R, G, B output for the sake of read latency.

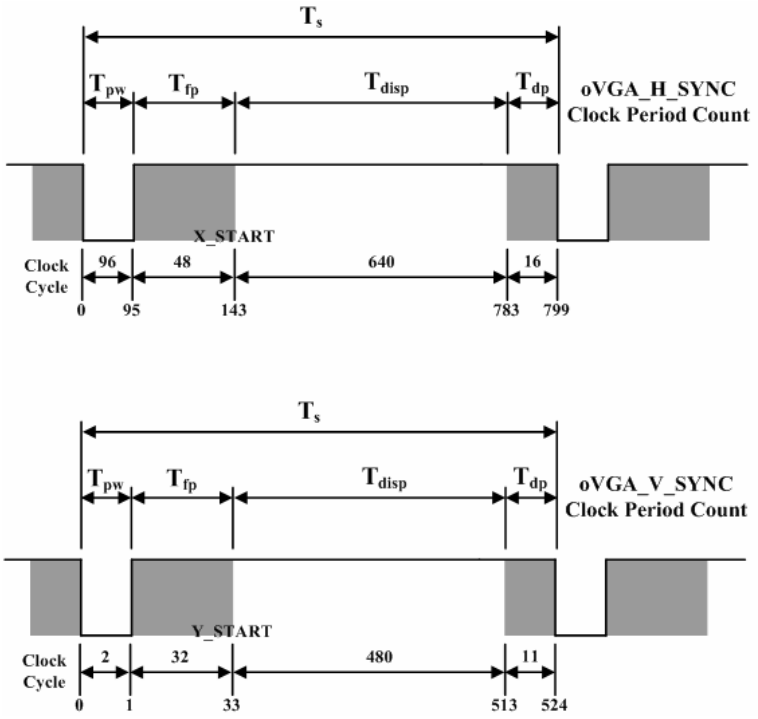


Fig. 2-45 Timing of oVGA_H_SYNC and oVGA_V_SYNC.

Table 2-5 Parameter list for VGA controller.

oVGA_H_SYNC		oVGA_V_SYNC	
Parameter	Value	Parameter	Value
H_SYNC_CYC	96	V_SYNC_CYC	2
H_SYNC_BACK	48	V_SYNC_BACK	32
H_SYNC_ACT	640	V_SYNC_ACT	480
H_SYNC_FRONT	16	V_SYNC_FRONT	11
H_SYNC_TOTAL	800	V_SYNC_TOTAL	525
X_START	H_SYNC_CYC + H_SYNC_BACK	Y_START	V_SYNC_CYC + V_SYNC_BACK

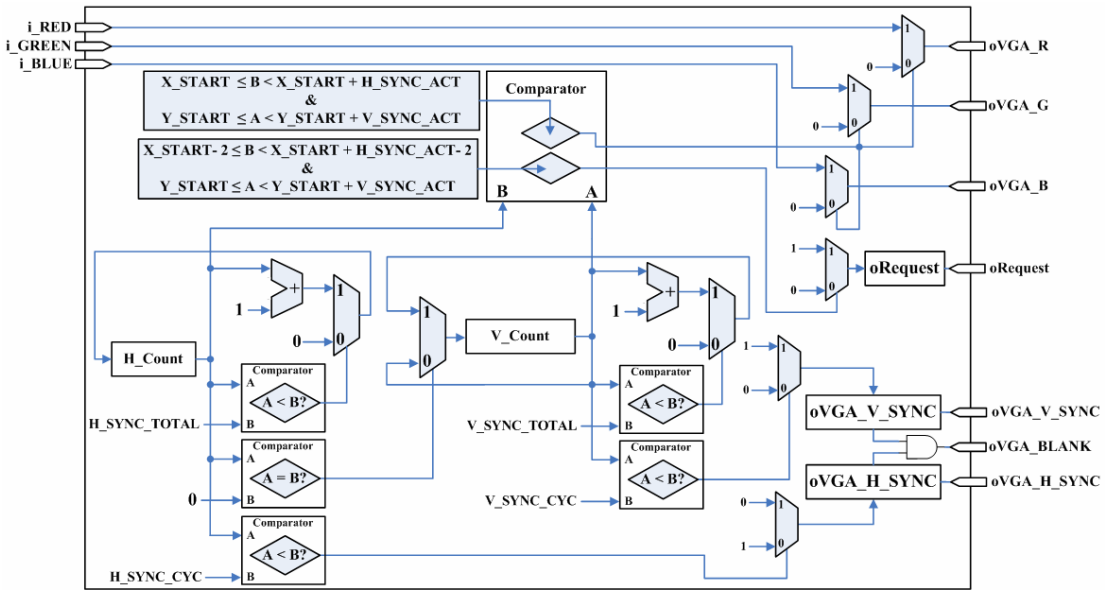


Fig. 2-46 VGA controller architecture.

2-5 SDRAM Controller

Knowing how SDRAM works is necessary for implementation of a SDRAM controller. A brief introduction of SDRAM specification and operation will be illustrated in this section according to the specification of PSC A2V64S40CTP 1.[11].

2-5-1 Introduction of SDRAM

The term “synchronous” means that SDRAM chip is synchronous with the system bus clock frequency, internal command dispatch and data transfer are base on the same clock frequency. In short, SDRAM inside is logically a two dimension storage array. Accurate access for memory cell can be done by fist designates row address then column address. The internal storage of an array of these memory cells is called a logic bank. For the process and cost constrain, it is impossible to produce a full capacity logic bank. Furthermore, base on the operation principle, single logic bank would cause severe addressing conflict and extremely internal access inefficiency while mass access. Therefore, SDRAM features multiple logic banks.

Consequently, bank addressing should be done first, row address and column address in the bank should be given next. While internal access, there is only one operating bank at the same time.

SDRAM does not like SRAM which can finish one access within one cycle. One SDRAM access usually takes about three to seven cycles and several commands to finish it. Before a SDRAM read or write a sequence of actions are needed:

1. Initialization of internal registers and memory cells.
2. SDRAM controller receives request from system BUS.
3. SDRAM controller receives data, analyzes address then actives bank with row address.
4. SDRAM controller issues read or write command with column address.
5. SDRAM finishes read or write by row address and column address.

Refer to Fig. 2-3, SDRAM receives address in two phases, thus row address buffer and column address are used to latch value form address line. After decoding row and column address a two dimension bank sends output to I/O buffer then outputs by DQ.

2-5-2 Pin Description of SDRAM of PSC A2V64S40CTP

Table 2-6 is SDRAM standard pin description:

Symbol	Type	Description
CLK	Input	Clock input.
CKE	Input	Clock enable: Deactivating the clock provides PRECHARGE and SELF REFRESH operations.
CS	Input	Chip select: turn on and off the command decoder.
WE	Input	Write enable.
CAS	Input	Column address strobe.
RAS	Input	Row address strobe.
DQM	Input	Input/output mask.
A0-A11	Input	Address inputs: A0-A11 are sampled during row address strobe (row address A0-A11) and column address strobe with READ/WRITE command (column-address A0-A7; with A10 defining auto precharge) to select one location out of the memory array in the respective bank. The address inputs also provide the op-code during a LOAD MODE REGISTER command.
BA	Input	Bank address.
DQ	I/O	Data Input/Output.

Table 2-6 Pin Description of SDRAM of PSC A2V64S40CTP

2-5-3 The Principle of Internal Operation and Timing

Some main operations and their principles are introduced in this section with realistic timing diagram given from PSC A2V64S40CTP datasheet is illustrated in this section.

(1) Power-up Sequence of SDRAM

On every power-up a necessary power up sequence is operated to stabilize pins like: CLK, CKE, DQM, etc, within 100 μs after power stabilizes. This is done automatically in SDRAM. After the power-up sequence, some procedural needs to be done for initialization.

(2) Initialization Sequence of SDRAM

As Fig. 2-3 shows, control logic and mode register which provide control signals for SDRAM. Thus, initialization of these control components is necessary after every power-up. When 200 μs has past after power-up sequence, all banks must be precharged. After precharge, refresh SDRAM at least eight times. Next, the mode register is set. Figure 2-47 shows timing of initialization sequence.

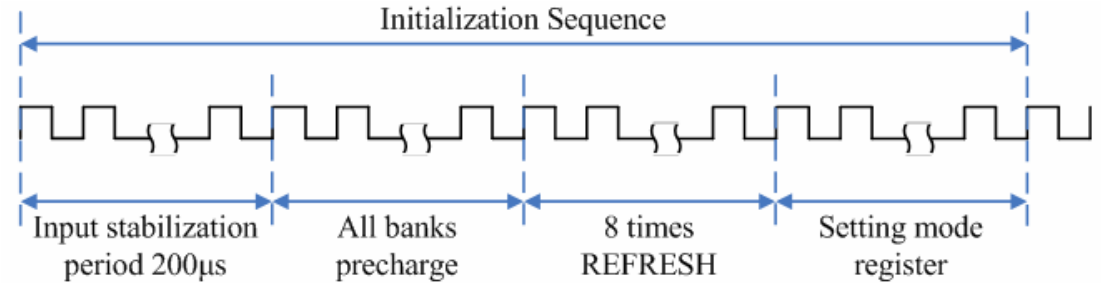


Fig. 2-47 Initialization sequence.

(3) The Concept of Row Address Strobe

After initialization, addressing in a logic bank in SDRAM needs to specify row address and active the row first, next then specify column address in the row.

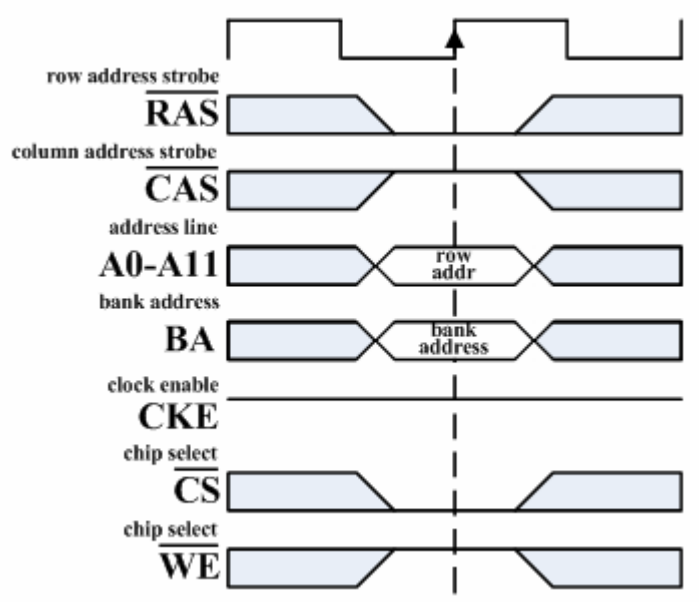


Fig. 2-48 Row address strobe.

This step is called row address strobe, chip select and bank address select can proceed at the same time. Refer to Fig. 2-48, $\overline{CS}$, BA are used for chip select and bank addressing, meanwhile, $\overline{RAS}$ is in a active state (active low). At this time, address line sends specific row address. The 12-bit address can specify 4096 rows in a bank.

(4) The Concept of Column Address Strobe

After row address has been specified we need to specify column address. In this stage, address line is the same 12-bit used in row address strobe. Read/write command is issued when doing column address strobe. Although column address uses some common bits with row address, $\overline{CAS}$ signal can differentiate between them. But column address strobe uses only eight bits of address line that can allocate 256 columns in each row.

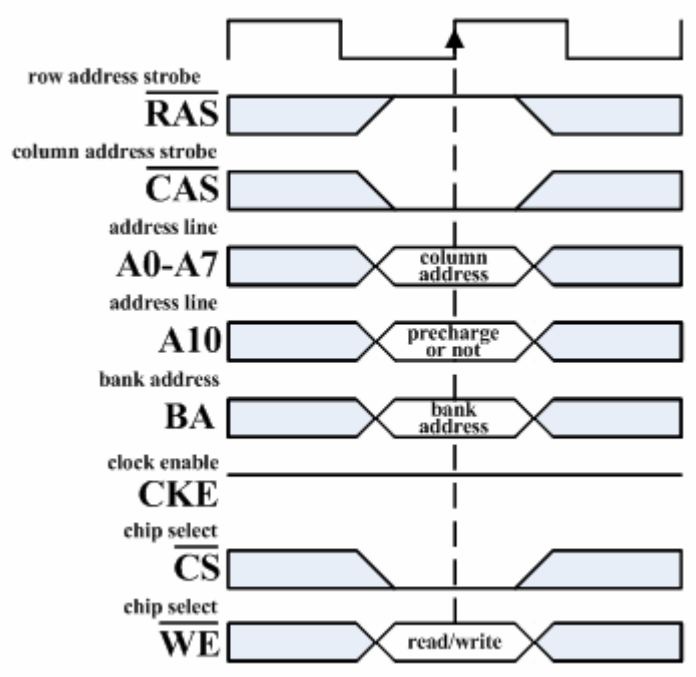


Fig. 2-49 Column address strobe

Refer to Fig. 2-49 $\overline{CAS}$ signal is triggered (active low) together with read/write command, nevertheless, there must has an interval between $\overline{RAS}$ and $\overline{CAS}$, this interval is defined as RAS to CAS delay, as known as t_{RCD}. In this time the internal row signal settles enough for the charge sensor to amplify it. t_{RCD} is measured in clock cycle count.

Observing at Fig. 2-50, $\overline{CAS}$ is validated together with read/write command after $\overline{RAS}$ with row active. The period between $\overline{RAS}$ and $\overline{CAS}$ is t_{RCD}.

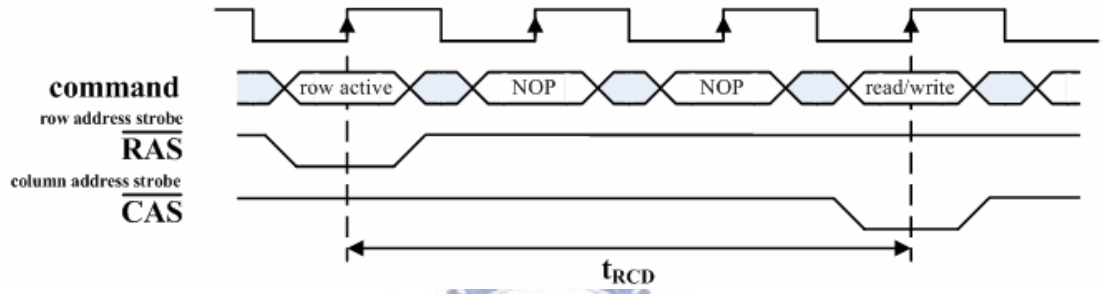


Fig. 2-50 $\overline{CAS}$, $\overline{RAS}$ timing diagram when t_{RCD} = 3.

(5) Read

A certain cell in the bank will be specified after addressing column address, next step is transferring data form I/O (DQ) to internal data line. But there needs a certain period from $\overline{CAS}$ to first data output which is called CL which is abbreviated from CAS latency. CL is measured in clock cycle count. Note that $\overline{CAS}$ is passed to memory cell instantly ratter after CL cycles. Actually, $\overline{CAS}$ and $\overline{RAS}$ arrive to memory cell instantly but $\overline{CAS}$ has a shorter response time than $\overline{RAS}$. Assuming we have an N rows × C columns bank, a row address needs to pass through N×C cells but column address needs to pass though only N cells. Due to nature response time of transistor in a cell makes data unable to be triggered with the same rising edge of $\overline{CAS}$, it takes at least an extra clock cycle. Figure 2-51 shows timing of a burst read and its output data. The concept of burst read/write will be introduced

later. Figure 2-52 shows how read works with $\overline{RAS}$, $\overline{CAS}$ in SDRAM in this system.

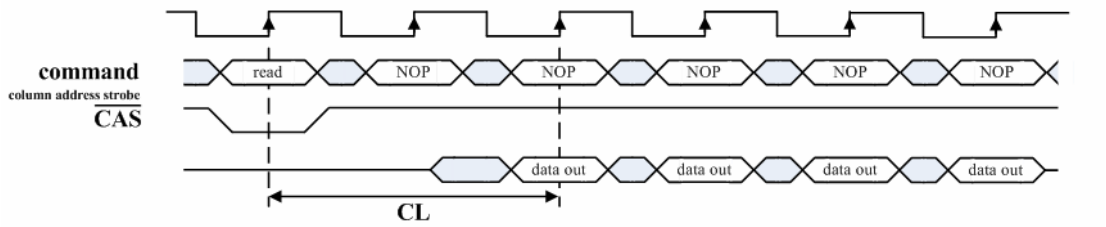


Fig. 2-51 Read timing diagram with CL = 2, burst length = 4.

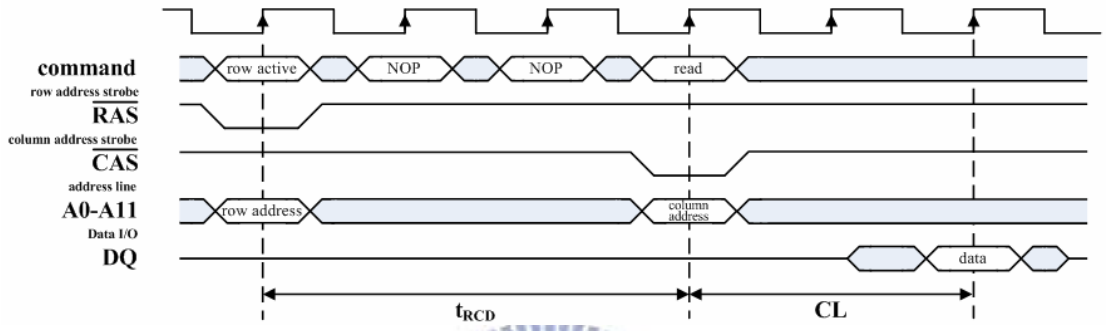


Fig. 2-52 Read operation in PSC A2V64S40CTP.

(6) Write

Row address strobe and column address strobe need to be done before a write as it does for a read. Thus, write command is issued t_{RCD} cycles after $\overline{RAS}$ has been triggered, but the write data from DQ does not need to wait for CL cycles. Write data is latched first once it is passed from DQ, then the latched value is written (or charged) to the storage capacitance by sense amplifier (Fig. 2-54). Thus write data can be issued together at the same time with $\overline{CAS}$. From the viewpoint of controller, one may say that is zero write latency. In fact, data is not charged to capacitance immediately because of capacitance charge time and transient time of transistor. Whereas the actual write needs a certain period. To ensure valid write an enough write and adjustment time is inevitable. This period is called write recovery time which is noted by t_{WR}, usually it takes more than one cycle. The timing relation between $\overline{RAS}$, $\overline{CAS}$, write, row address, column address and write data is depicted

in Fig. 2-53.

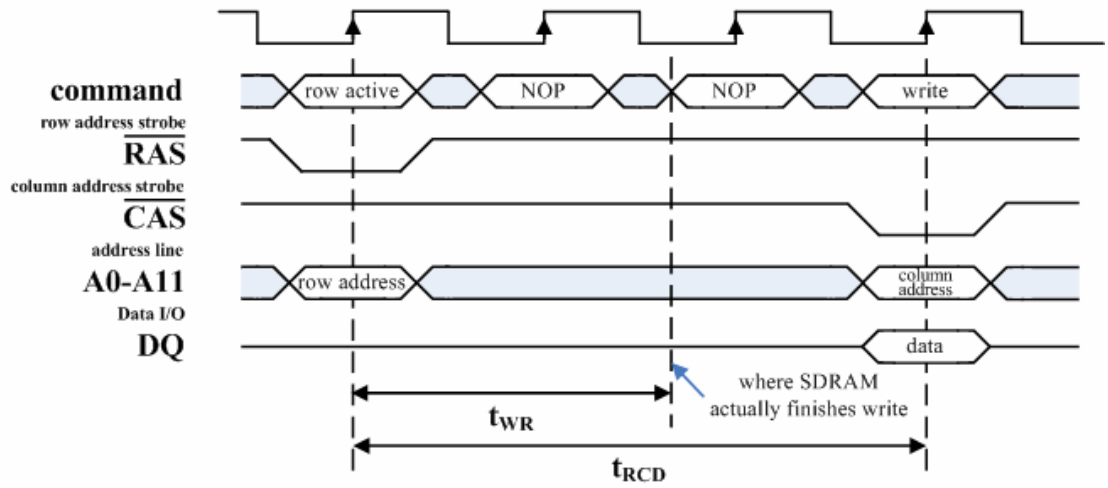


Fig. 2-53 Write operation in PSC A2V64S40CTP.

(7) Burst

SDRAM supports bust transfer. Burst transfer refers to consecutive data transfer in cells which are neighbors in the same row. The amount of cells in a consecutive transfer is called burst length. Usually, burst length can be 1, 2, 4, 8 or full page. Full page burst transfer refers to a consecutive transfer of all cells in a row of a bank. One burst transfer can be done by assigning initial address and burst length. SDRAM will automatically operate read/write on following cells in the row one by one without controller providing column address continuously. Only the first data in burst transfer takes several cycles, the following transfer takes one cycle for each data. The burst read with burst length four is shown in Fig. 2-51. A burst write is likewise to burst read the only different is that burst write need not to wait a CAS latency.

(8) Precharge

Due to the small capacitance inside of SDRAM, a memory cell's storage value must be amplified to a discriminable value. The amplification is done by sense amplifier. Observing the structure of SDRAM storage element in Fig. 2-54, if we read

logic “1” from the capacitance, the capacitance would be pull down to logic “0” after discharging. What sense amplifier does is to cache the logic value from capacitance. If the next access is in a different row, the current row is closed before another is opened. The current “row cache” is written back to whole row. This is called row precharge. The reason sense amplifier writes back to whole row is because during row address strobe the row select transistor would affect the voltage in capacitance even the cells not being selected by column address strobe. Precharge can be done by issuing a command or set the mode register to auto precharge after every read/write. Actually, it is the action to refresh the comparison voltage of sense amplifier (usually is half the voltage of capacitance). Sense amplifier compares the capacitance voltage value with its reference voltage, which helps to judge the voltage value being read out.

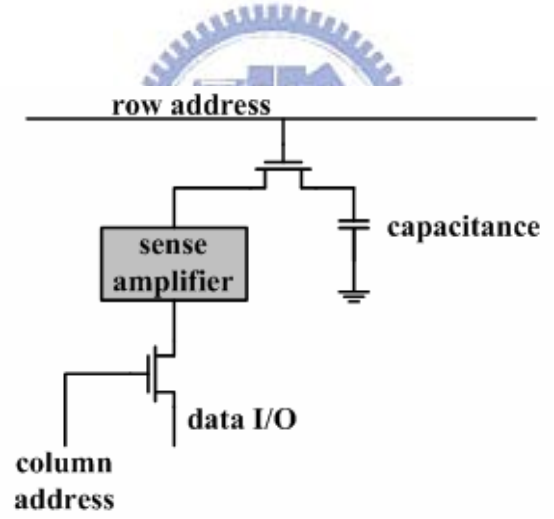


Fig. 2-54 SDRAM storage element.

Checking for the timing diagram of read in Fig. 2-55, we can find that A10 at the moment when read is issued decides whether auto precharge or not. In this case, we set a read with auto precharge by asserting A10. The precharge command is not issued by controller, it operates spontaneously. A new row active can be issued no less than t_{RP} cycles after precharge where t_{RP} is called row precharge period is the number of clock cycles needed to terminate access to an open row, and open access to the next row. The time interval between a row active command and precharge command

is known as active to precharge delay, t_{RAS}. t_{RC}, refers to the row cycle time, which determines the minimum number of clock cycles a memory row takes to complete a full cycle, from row activation up to the precharging of the active row.

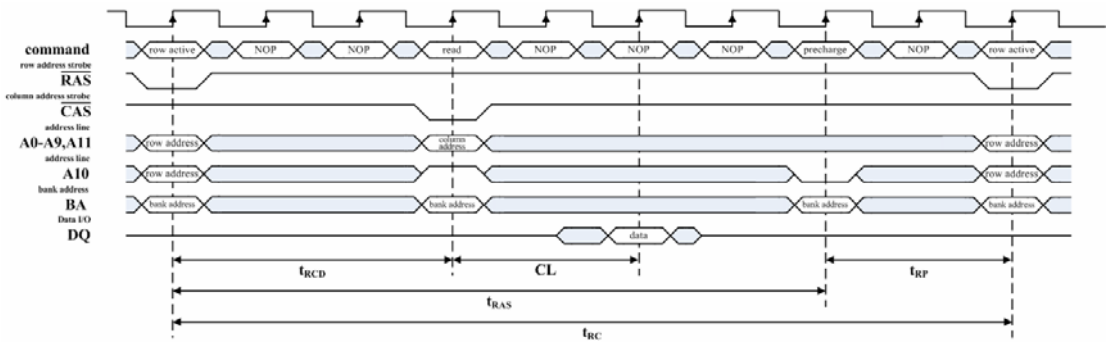


Fig. 2-55 Read with precharge.

(9) Refresh

Because the information is stored in a small capacitance suffering on leakage effects, the SDRAM cell needs to be refreshed periodically with the refresh command. Refresh does same thing as precharge, the difference between them is that precharge makes operating rows in one or all banks precharged and it does not operate periodically, whereas refresh operates periodically and sequentially to each row to maintain values in memory cells not being precharged for a while. The auto refresh in SDRAM takes no row address from controller because an internal row address counter will generate sequential row address. Refresh involves all banks thus all banks halt during auto refresh. Auto refresh takes t_{ARFC} cycles to finish which is called auto refresh cycle time. Auto refresh timing is show in Fig. 2-56

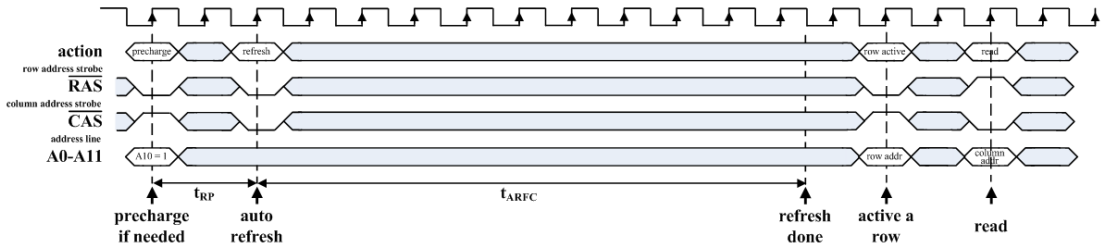
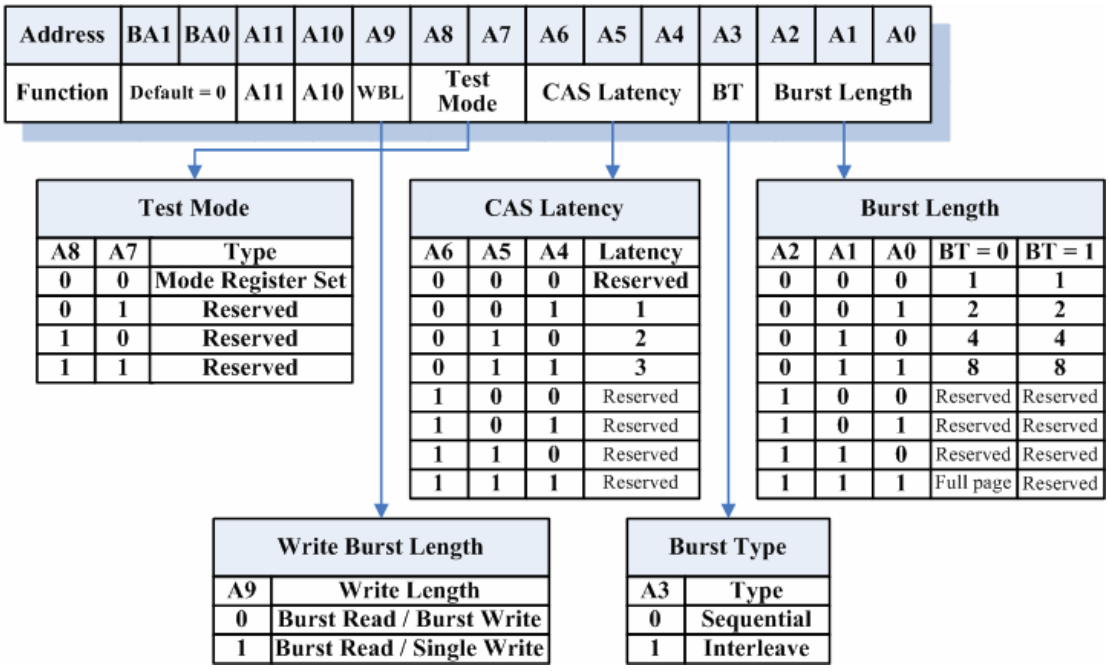


Fig. 2-56 Timing diagram of auto refresh.

2-5-4 Mode Register in SDRAM

Refer to Fig. 2-3 the mode register is the core of the control in a SDRAM. Figure 2-57 shows the mode register fields. In this design, we set burst length to be eight, burst type to be sequential, CAS latency to be three, write length to be burst read/ burst write. The reason why setting burst length to be one is that application in this work merely operate on a low data rate like VGA and ICA work under 25.2MHz and 64Hz, single burst length is enough.



Address	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Function	Default = 0		A11	A10	WBL	Test Mode		CAS Latency			BT	Burst Length		

Test Mode		
A8	A7	Type
0	0	Mode Register Set
0	1	Reserved
1	0	Reserved
1	1	Reserved

CAS Latency			
A6	A5	A4	Latency
0	0	0	Reserved
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

Burst Length				
A2	A1	A0	BT = 0	BT = 1
0	0	0	1	1
0	0	1	2	2
0	1	0	4	4
0	1	1	8	8
1	0	0	Reserved	Reserved
1	0	1	Reserved	Reserved
1	1	0	Reserved	Reserved
1	1	1	Full page	Reserved

Write Burst Length	
A9	Write Length
0	Burst Read / Burst Write
1	Burst Read / Single Write

Burst Type	
A3	Type
0	Sequential
1	Interleave

Fig. 2-57 Mode register field.

2-5-5 Timing Parameters

Before we step into the implementation of the SDRAM controller, some important timing parameter dominates electrical characteristics of SDRAM should be verified first. Most of these parameters are explained in previous section, which are list in Table 2-7.

Table 2-7 SDRAM timing parameters.

Parameter	Symbol	Value	Unit
Row active to row active delay	t_{RRD}	14	ns
RAS to CAS delay	t_{RCD}	21	ns
Row precharge time	t_{RP}	21	ns
Row active time	t_{RAS}(min)	42	ns
	t_{RAS}(max)	100	us
Row cycle time	t_{RC}	63	ns
Auto refresh cycle time	t_{ARFC}	70	ns

2-5-6 Implementation of SDRAM Controller

After knowing basic principle of SDRAM, the goal of SDRAM controller is to translate outer read/write request into internal control signals that matches timing specification. The architecture of the SDRAM controller is illustrated in Fig. 2-58.

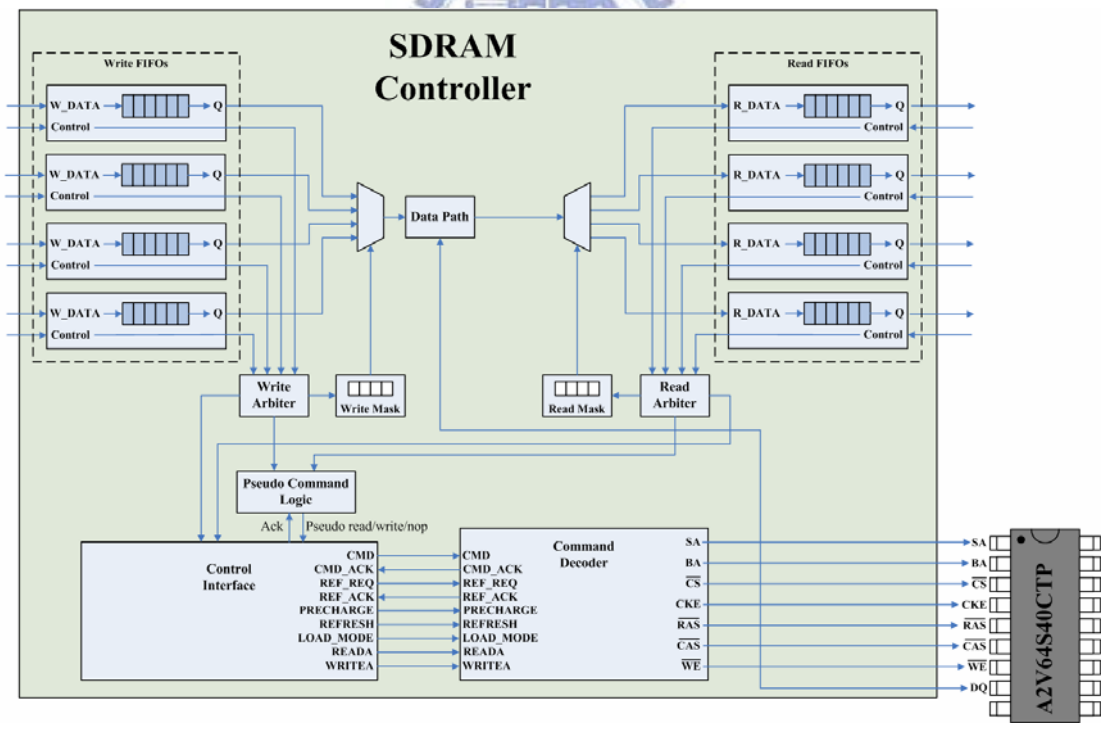


Fig. 2-58 Architecture of SDRAM controller.

We dispose four write request FIFOs and four read request FIFOs to deal with the read/write requests. The arbiter chooses one of the write FIFOs with its priority

and selects proper write data to the module "data path" by setting write mask. Where module "data path" directs write data to DQ pin of SDRAM, which is a bidirectional input/output pin. Likewise, read arbiter judges read data from DQ to proper read FIFO by setting read mask according to read priority. Also, according to the status of read/write FIFO, pseudo command control logic issues pseudo command: read, write or nop to the module “control interface”. The module "control interface" mainly deals with initial sequence, auto refresh and pseudo read/write commands. It issues commands and receive command acknowledgement to/from command decoder module. The module "command decoder" receives and decodes commands from "control interface" then generates control signals with proper timing to the SDRAM.

(1) Implementation of Arbiter and Pseudo Command Control Logic

The structure of the read/write arbiter is put together in Fig. 2-59. Threshold checker checks the depth of all read/write request FIFOs, if any one’s depth is over the threshold (burst length) it trigger a read or write signal for a burst read or burst write with eight datum. The read/write signal is sent to pseudo command control logic. However SDRAM can perform one read or one write at any instance, only one FIFO will be served at any time. Therefore a stationary priority is given for the FIFOs. Refer to FIFOs in Fig. 2-59, the FIFO atop (1st FIFO of write FIFOs) has the highest priority and the priority decreases form top to down, in other words, the 4th read FIFO has the lowest priority. Because we transfer eight data in a burst thus the starting point of the next transfer should be increased by eight. Address counter renews the address after every transfer. The threshold checker also controls the read/write mask to select correct read/write address to control interface.

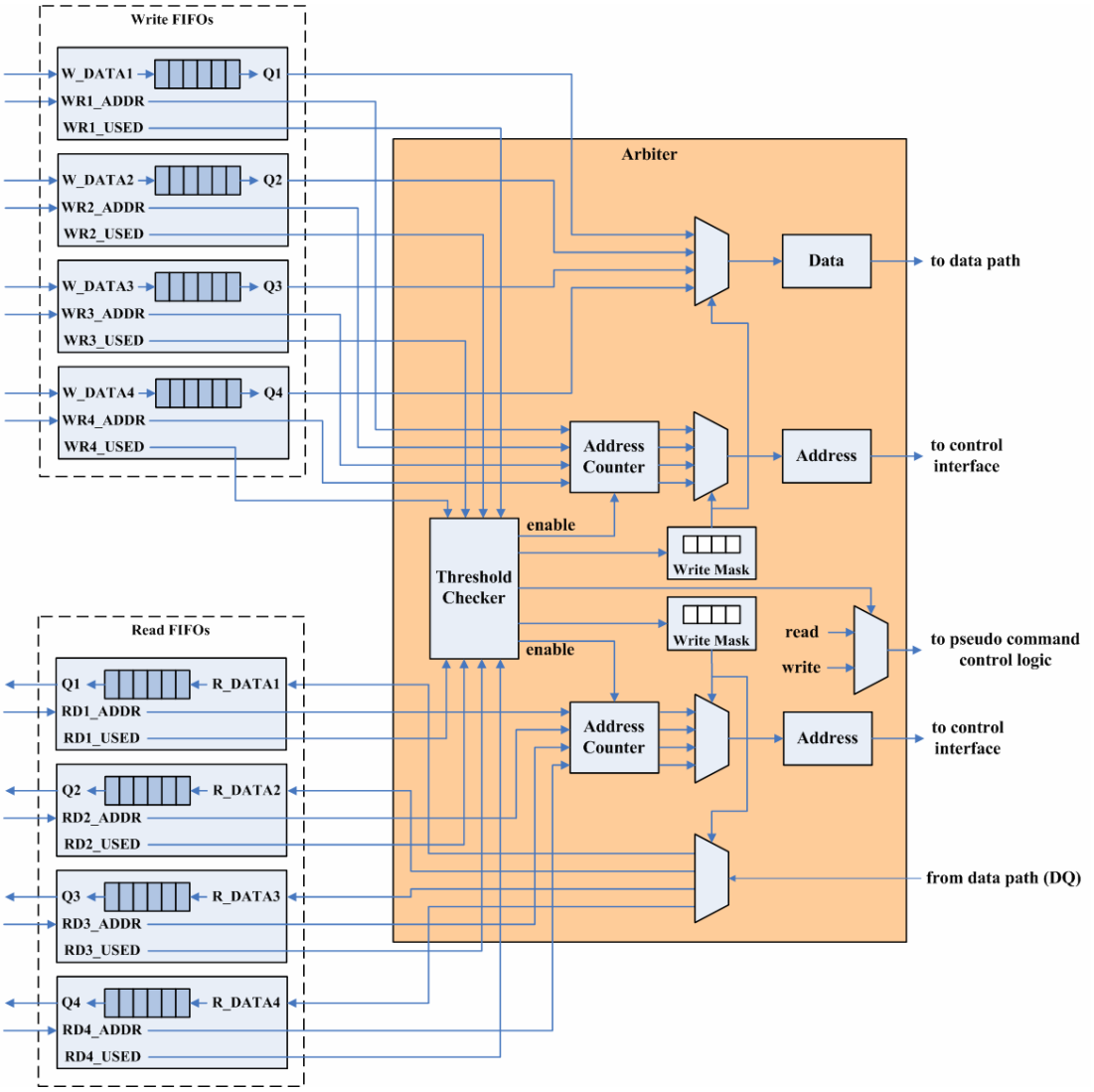


Fig. 2-59 Architecture of arbiter.

The core of the pseudo command control logic is a finite state machine, as shown in Fig. 2-60. In the IDLE stage, it waits for the read/write signal from arbiter. Once it detects the occurrence of read or write, it sends a read/write pseudo command to control interface which will be introduced next. In this module it only sends read, writ or nop to “control interface”. The control interface will responds proper action to SDRAM. After sending a pseudo command, it enters second stage to wait for the acknowledgement from the command decoder. Once it receives the acknowledgement that means the read/write procedure inside SDRAM is also started. After receiving

acknowledgement it forwards to third stage to complete a burst read/write. If the "pseudo command control logic" issued a write, it would read (get) eight datum from write FIFO and pass these datum to pin DQ through "data path" module. And vice versa, if a read was issued the pseudo command control logic would wait until first data is read out from SDRAM (through pin DQ). Once it gets data read from SDRAM, it writes (pushes) the data to read FIFO. After a burst transfer is finished, it goes back to IDLE stage.

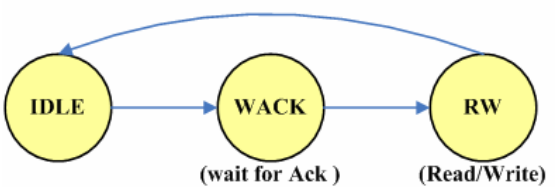


Fig. 2-60 State machine of pseudo command control logic.

(2) Implementation of Data Path

It is the module to control the only bidirectional port DQ. Where DQ takes output enable signal (which is not shown in Fig. 2-58) from module "command" to select the value to assign to DQ. During a burst write, we just directly write the data that is selected by write mask to the DQ pin. During a read, we just direct DQ to data input of a read FIFO selected by the read mask. The behavior of module "data path" is depicted in Fig. 2-61.

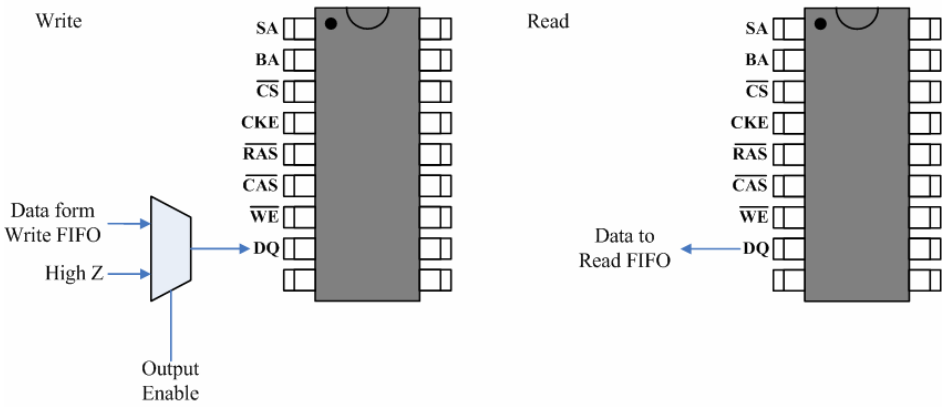


Fig. 2-61 Data path.

(3) Implementation of Control Interface

Control interface is in charge of initial sequence, auto refresh, turning the pseudo commands into read/write/nop commands and sends them to the command decoder and receiving command acknowledgement from command decoder.

There are two timers in this module. First one is initial timer which only triggered once in the beginning. The control interface sends initial request and receives its acknowledgement. After initial request hand shaking, it waits about 240us then launches refresh 8 times, in the end of initial sequence it sets mode register. The other timer is auto refresh timer, which counts down repeatedly to issue a refresh command to command decoder.

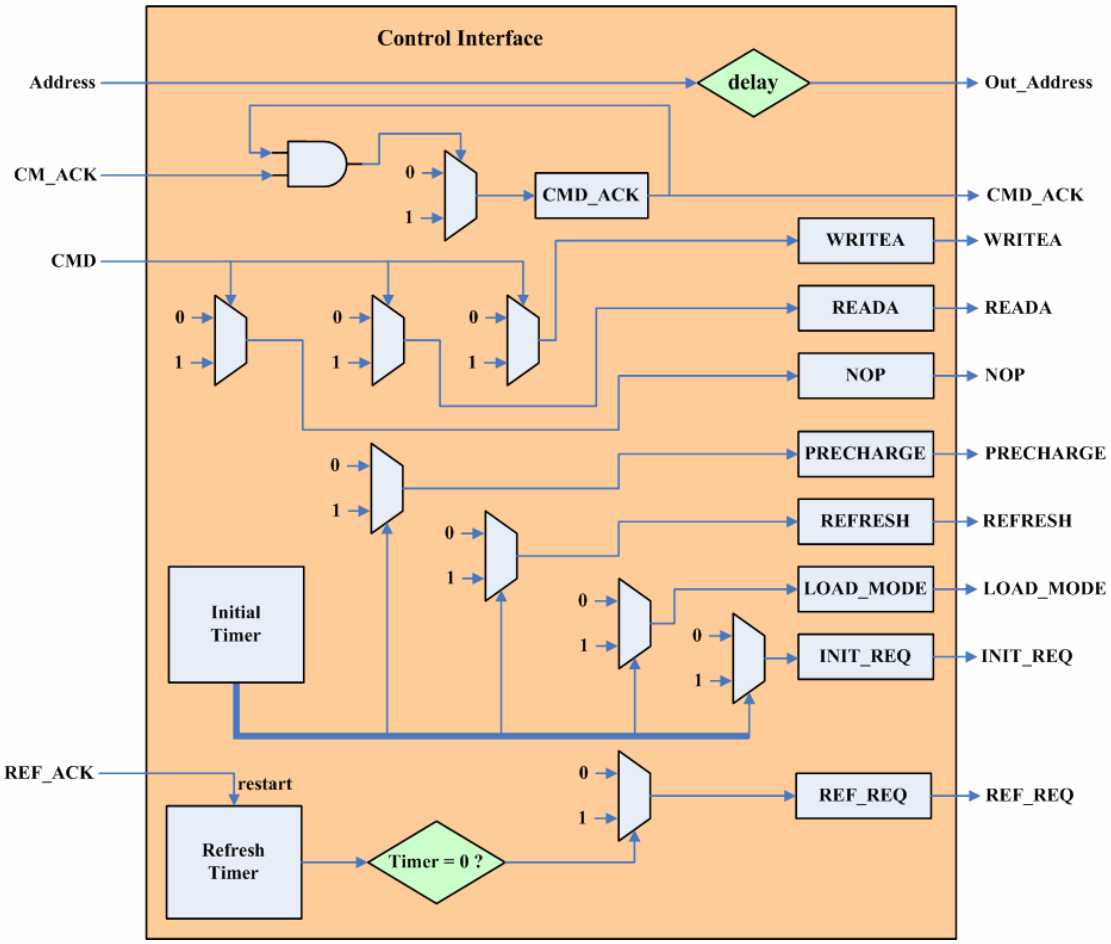


Fig. 2-62 Architecture of control interface.

After initial sequence, it will start to receive pseudo read/write. Once it gets pseudo read, it sends a read command to command decoder. In the same way, it sends a write command to command decoder for a pseudo write. Also it provides command decoder address information for decoding. Fig. 2-62 shows the architecture of control interface which reflects the logic function described above. Note that a refresh (REFRESH) and refresh request (REF_REQ) would not occur simultaneously. Command decoder does the same reaction in response for these two signals. All acknowledgements are come from command decoder. The delay of address is just for synchronization of transfer timing. Overall, what control interface does is to generate the nop, read, write, precharge and refresh commands at the right moment.

(4) Implementation of Command Decoder

Command decoder receives the command issued from control interface and generates and passes correct signals corresponding to each command. The signals are passed to SDRAM which match SDRAM timing standards. The architecture of command decoder is shown in Fig. 2-63. The commands from control interface are mutually exclusive, that is one command is valid at a moment. While receiving any command, the event generator will generate one and only one corresponding event for each command. Once an event occurs, an acknowledgement is asserted. Only refresh will cause a refresh acknowledgement (REF_ACK), the other commands cause command acknowledgement (CM_ACK). All the acknowledgements are sent to control interface. SDRAM signal timing controller direct connects with physical SDRAM. For every event the SDRAM signal timing controller generates a sequence of corresponding signals. Take command read/write for example, the first cycle should deassert $\overline{\text{RAS}}$ (active low), then after t_{RCD} cycles $\overline{\text{CAS}}$ is deasserted too. If the event do_reada or do_writea arose, $\overline{\text{RAS}}$ would be deasserted in SDRAM signal

timing controller. The do_rw register in Fig. 2-63 is asserted by command read/write which is delayed t_{RCD} cycles to pull down $\overline{CAS}$. In short, each command triggers an event, the occurrence of the event is detected then a sequence of control signals that match timing specification is input to SDRAM in timing order.

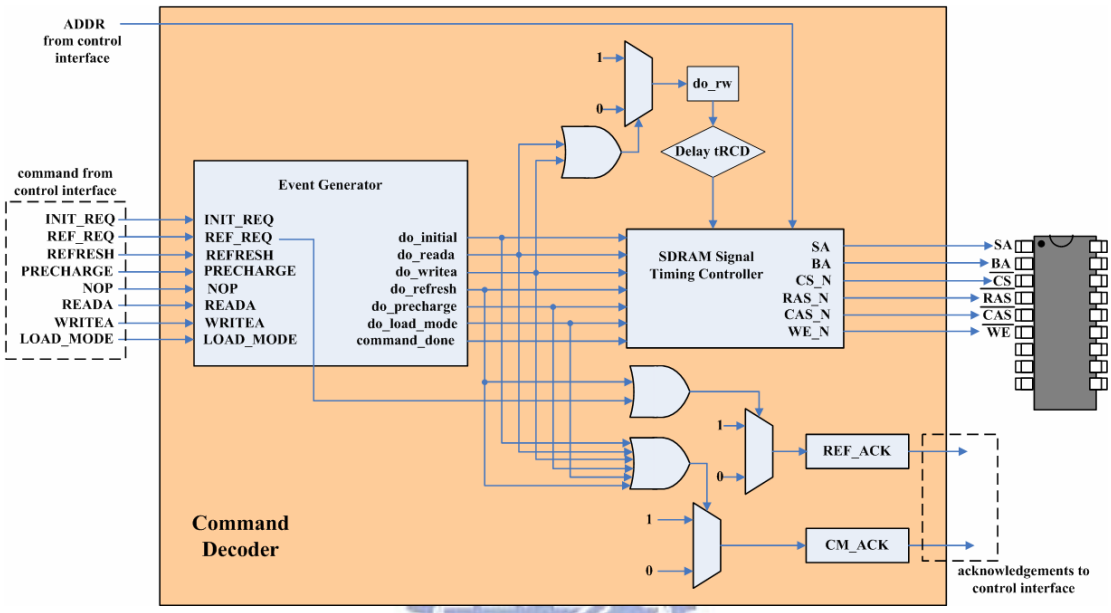


Fig. 2-63 The architecture of command decoder.

2-6 Block level simulation result

The simulation results of the designs mentioned above are shown in this section.

2-6-1 Simulation Result of VGA Controller

Figure 2-64 shows the timing diagram of post simulation of VGA controller. Look into the rim A of the middle picture and notice its partial zoom in rim E, refer to Table 2-5, when H_Count reaches H_SYNC_TOTAL that is 800, horizontal synchronization is triggered. Rim B and rim D show that horizontal synchronization is triggered periodically every 800 cycles. Same principle is applied for vertical synchronization in rim C.

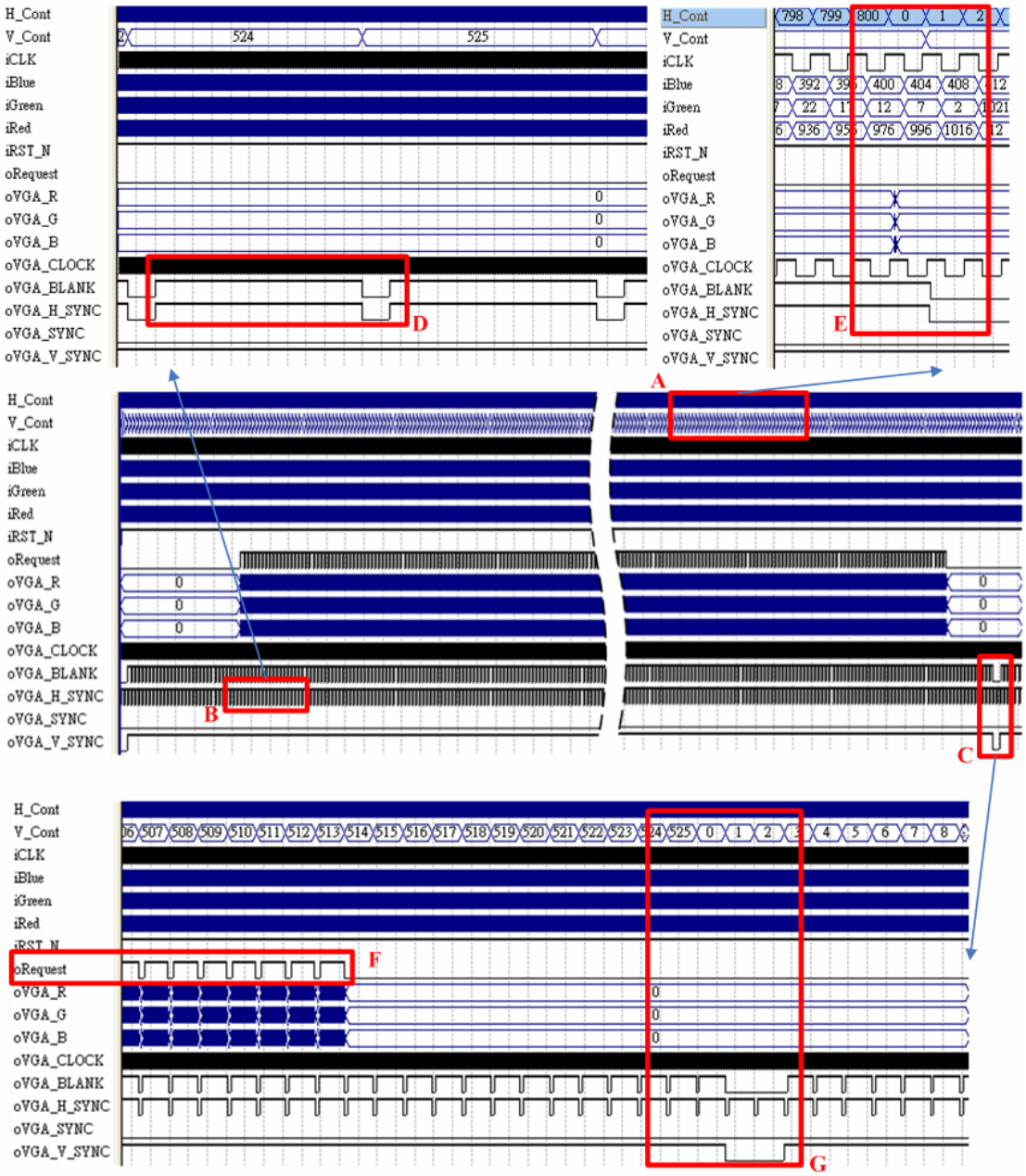


Fig. 2-64 Timing diagram of post simulation of VGA controller.

Look into rim G, when V_Count reaches V_SYNC_TOTAL that is 525, vertical synchronization is triggered. Observing that after V_Count reachs 513, controller stops sending read request to SDRAM controller (rim F) and there is no any RGB output to VGA DAC. The read request is sent for asking RGB value for each pixel. Actually, 513 is the boundary where back porch takes place for vertical synchronization. In fact, synchronization signals are active low signals. Refer to

Table 2-3 and Table 2-4, the controller matches timing specification of 640×480×60 VGA synchronization signals.

2-6-2 Simulation Result of SDRAM Controller

The simulation result of SDRAM controller is shown in Fig. 2-65. Rim A and B is the part of initial sequence. In rim A, control interface first issued an initial request (INIT_REQ) then it launched PRECHARGE, 8 times REFRESH, finally a LOAD_MODE (mode register load) to finish the initial sequence. The command decoder received the command issued from controller interface and triggered the occurrence of the events, as you can see, do_initial, do_precharge, 8 times do_refresh and finally do_load_mode are triggered in order. Note the CM_ACK and CMD_ACK are the handshaking signals between control interface and command decoder. The zoom in of initial sequence is shown in Fig. 2-66.

Look at rim D, write requests are pushed in write FIFO first then read requests are pushed in. Write requests are addressed with WR1 (in 1st write FIFO) with the values (WR1_DATA) being written. Address 20000 (WR1_ADDR) represents the first row of the third bank in SDRAM. When the amount of request in 1st write FIFO over the burst length, arbiter issued “write” (Write signal in rim D) to control interface. In rim F, controller interface resolved Write signal form arbiter and issued a WRITE command to command decoder. Notice, the handshaking (CM_ACK, CMD_ACK which are not inside rim F) between control interface and command decoder are varied almost simultaneously with WRITE command. Rim C show the control signals directly connect with physical SDRAM, they are set by command decoder. Figure 2-67 is the zoom in of rim C. The two rims denote two burst write with burst length 8. As it shows, a write was completed by first deasserting $\overline{\text{RAS}}$ (RAS_N) then deasserting $\overline{\text{CAS}}$ (CAS_N) three cycles (t_{RCD}) latter. Notice there are

eight DQ_result transients in a burst write.

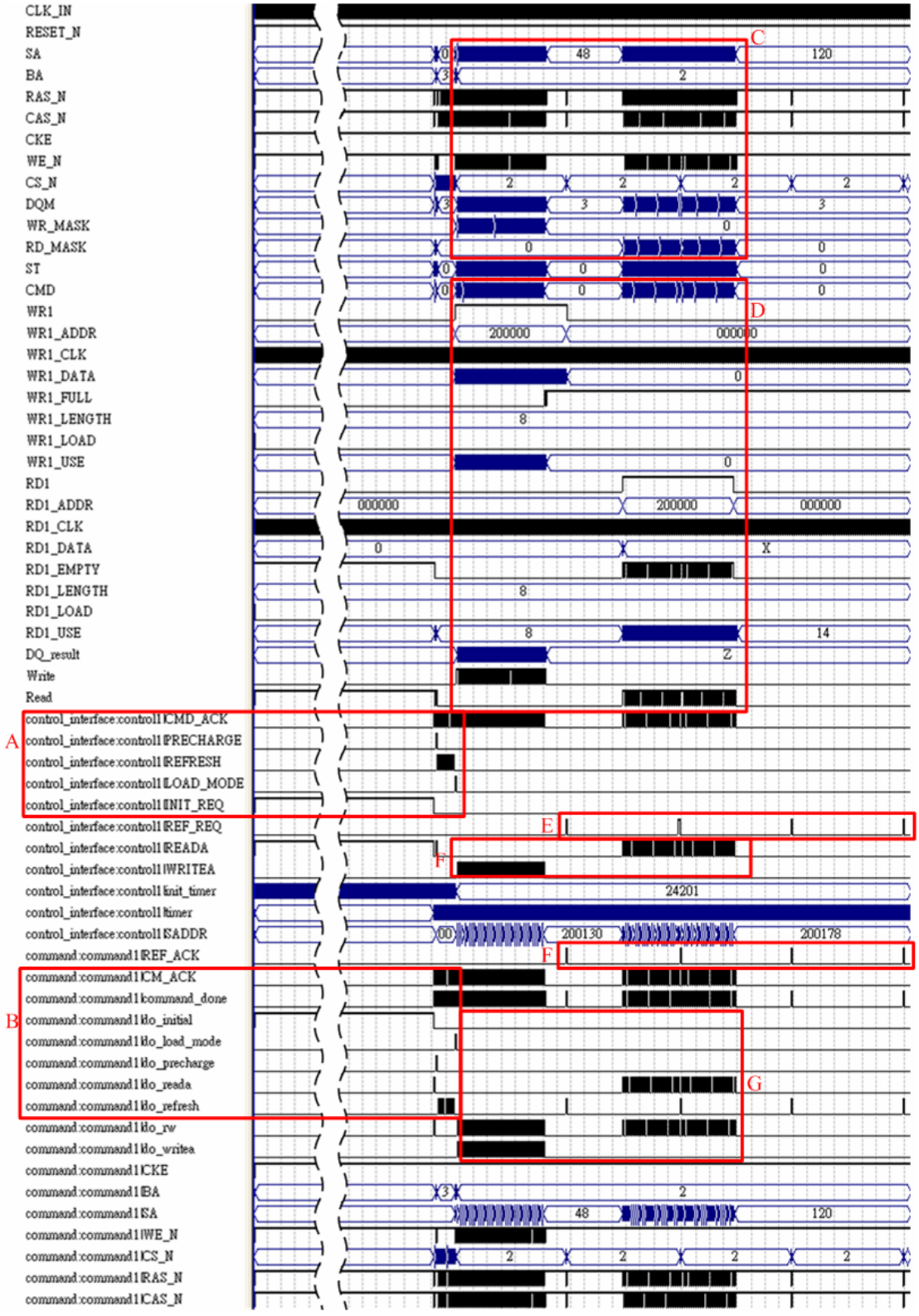


Fig. 2-65 simulation result of SDRAM controller

The simulation of read requests is similar with write request. In rim D read requests are addressed with RD1, RD1_DATA and RD1_ADDR. The value 20000 of RD1_ADDR represents the first read is started from the first row of bank three. Note the arbiter issued “read” (Read) to controller interface. In rim F, controller interface resolved Read signal form arbiter and issued a READ command to command decoder. Control signals in rim C are set by command decoder. The zoom in of these signals is shown in Fig. 2-68.

Notice, the handshaking betweens control interface and command decoder are varied almost simultaneously with WRITE and READ commands. Rim G shows the events do_writea and do_reada triggered by command WRITE and READ. Also do_rw is used to control deassertion timing of $\overline{\text{CAS}}$.

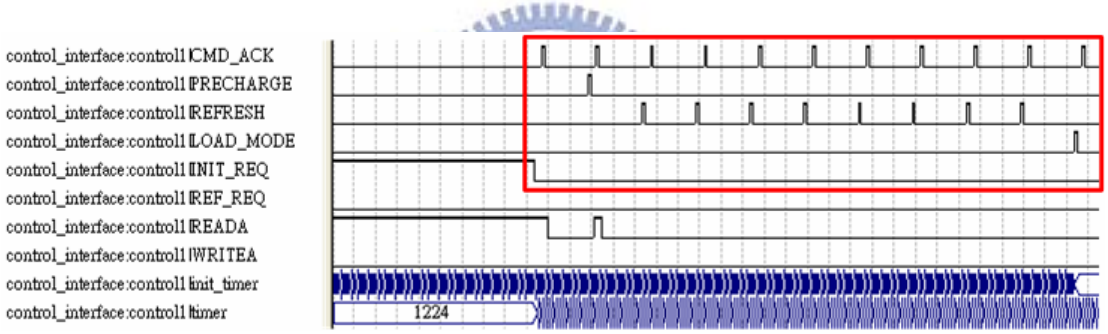


Fig. 2-66 Simulation of initial sequence in SDRAM.

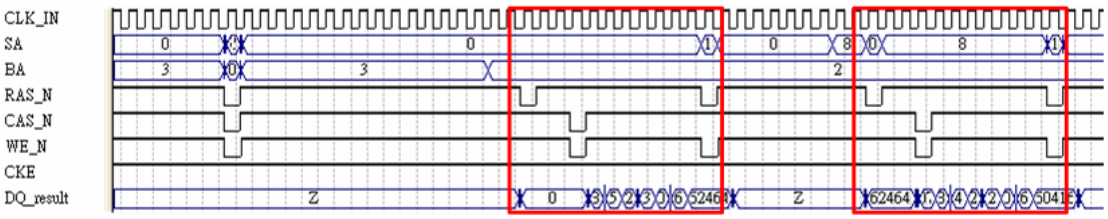


Fig. 2-67 Simulation of write in SDRAM.

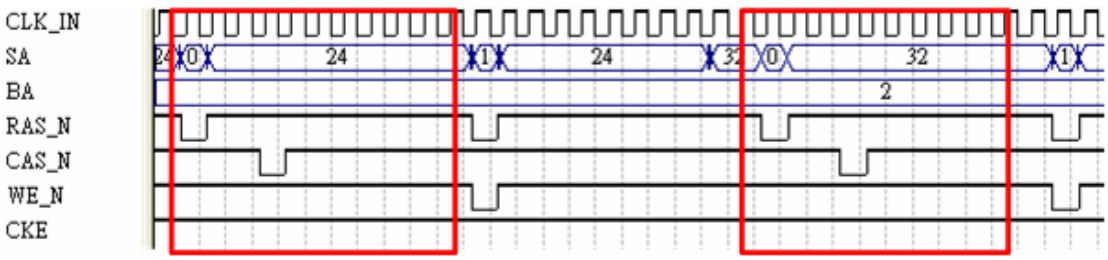


Fig. 2-68 Simulation of read in SDRAM.

Rim E shows the operation of auto refresh, auto refresh is done by asserting refresh request (REF_REQ) periodically. Rim F is the acknowledgement of refresh.

2-6-3 Simulation result of on-line ICA

Base on the algorithm of off-line ICA flow, in order to achieve real-time calculation, we divide the input of mixed sources to process real-time signals individually, and overlap the previous mixed sources to calculate new separated signals.

In this part we verify the real ICA hardware results in real-time with GUI display. The post-simulation and off-line correlation has shown in Fig. 2-69, Fig. 2-70, Fig. 2-71, and Fig. 2-72 individually. In Fig. 2-73(a) and Fig. 2-74(a) show the GUI display of four channel mixed signals. In GUI display, we set the data bandwidth are 8-bit and the header is FF. However, in Fig. 2-73(b) and Fig. 2-74(b) show the ICA result in GUI display, we can find that if the original signals are pure super-Gaussian like this, the system will has a good result. Another way, we discuss EEG signal that has less information without ICA process apparently in Fig. 2-75(a), and Fig. 2-76(a). After ICA Infomax update, the analysis signals will have more distinct information than original signals without ICA process. The ICA result of EEG signals shows in Fig. 2-75(b), and Fig. 2-76(b). In real-time ICA verification, we divided into two parts: super-Gaussian signals and EEG signal in real environment.

(1) Post-Simulation and Off-line Correlation

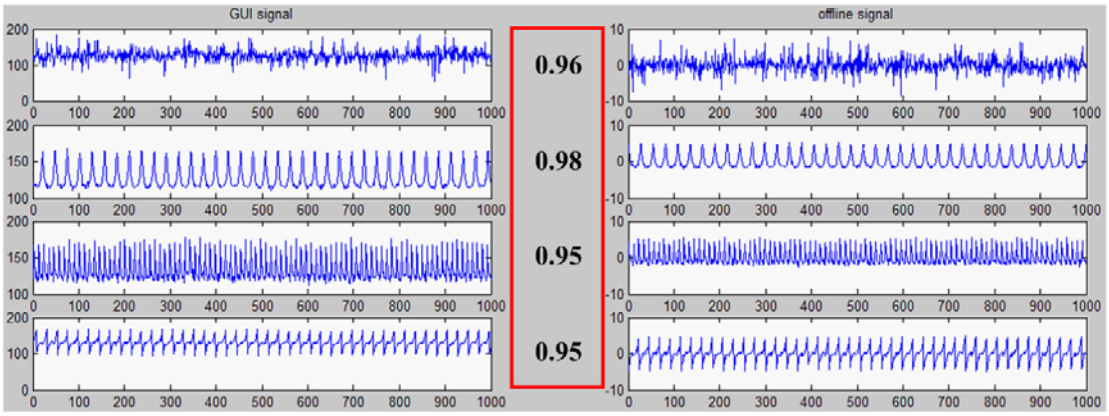


Fig. 2-69 Left is pattern one post simulation, and right is offline ICA result.

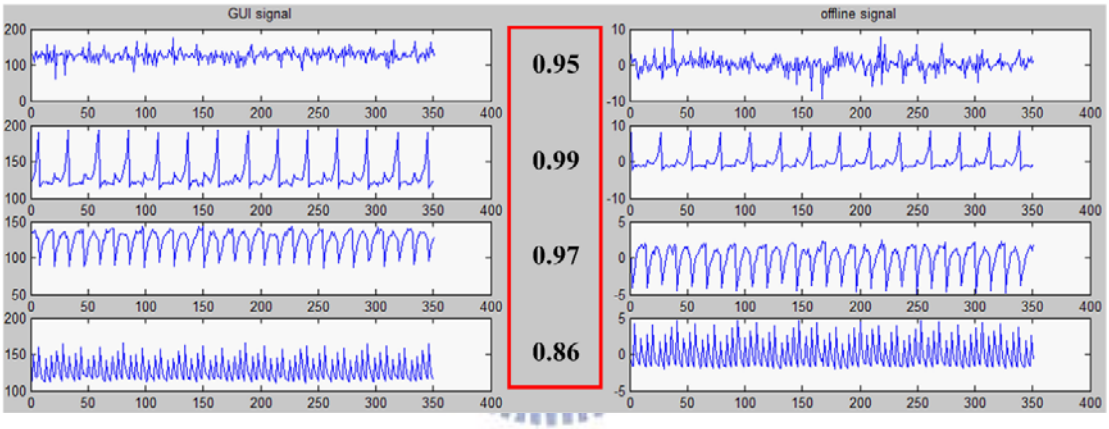


Fig. 2-70 Left is pattern one post simulation, and right is offline ICA result.

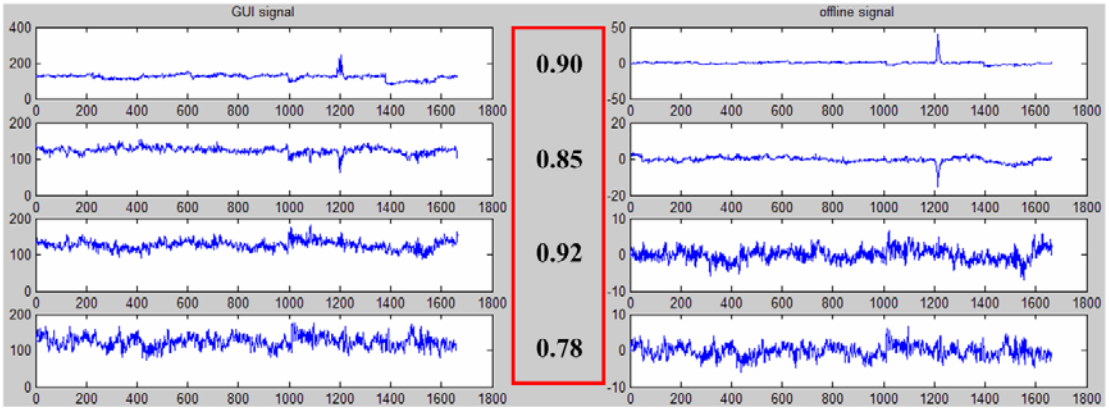


Fig. 2-71 Left is EEG1 post simulation, and right is offline ICA result.

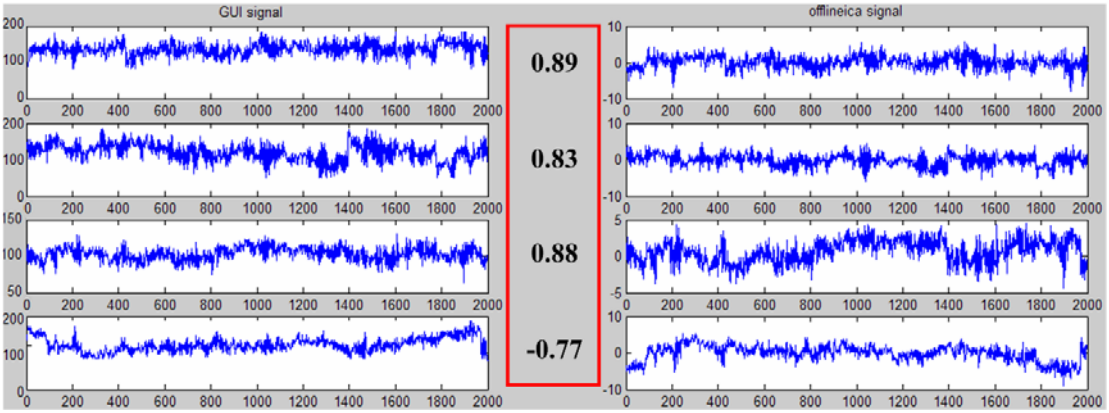
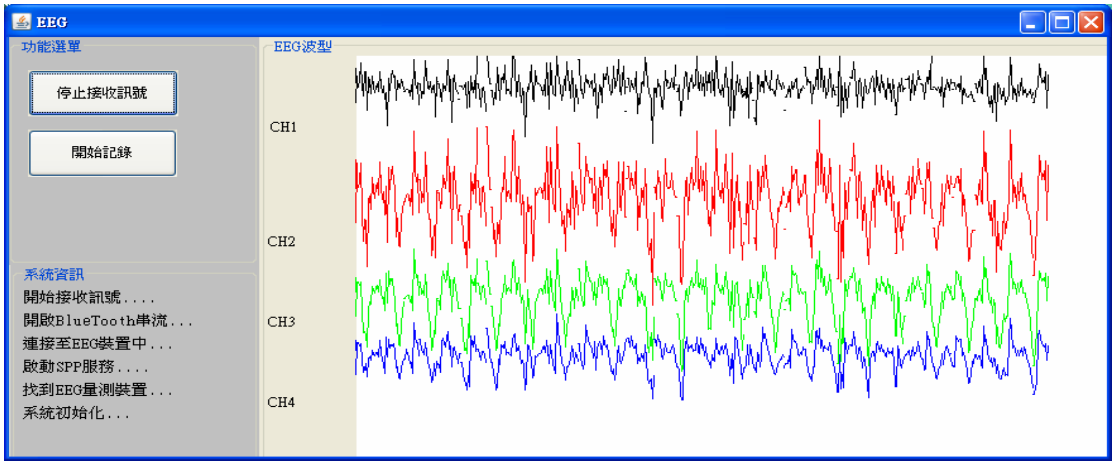


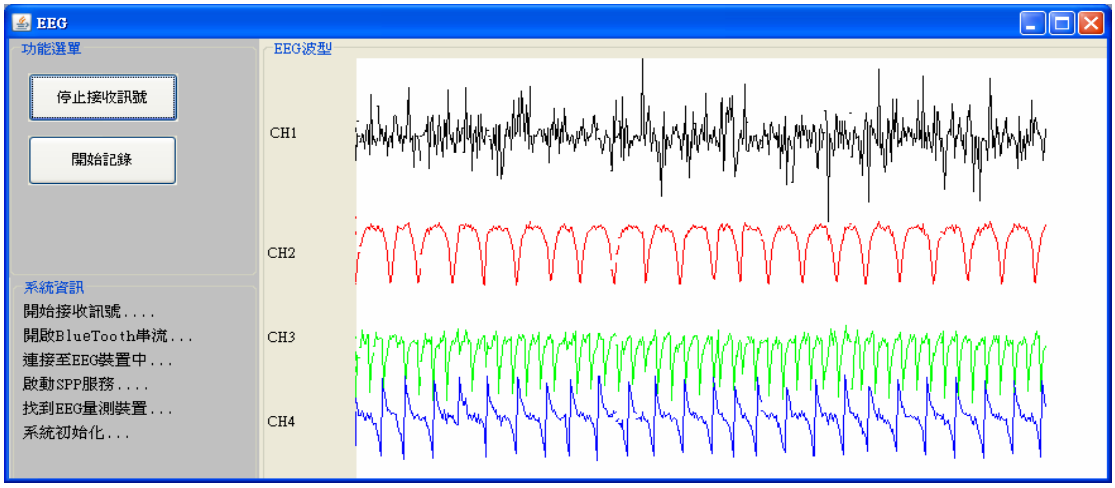
Fig. 2-72 Left is EEG2 post simulation, and right is offline ICA result.

(2) Super Gaussian in GUI

Super Gaussian Pattern One



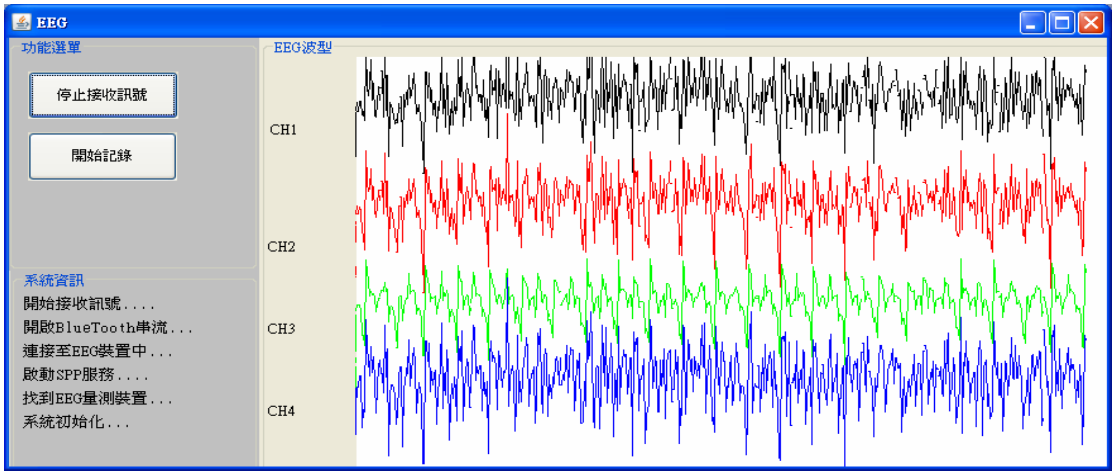
(a)



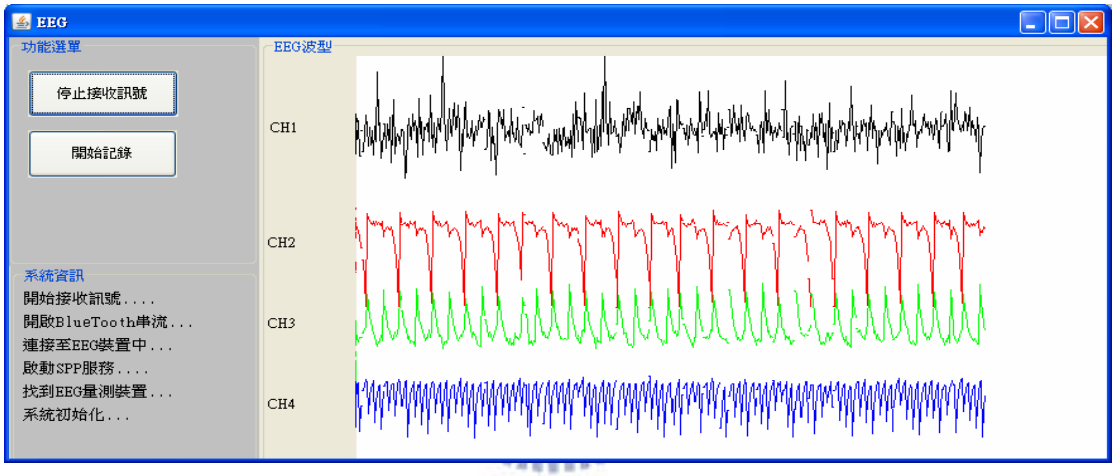
(b)

Fig. 2-73 (a) Mixed signal (b) ICA signal.

Super Gaussian Pattern Two



(a)

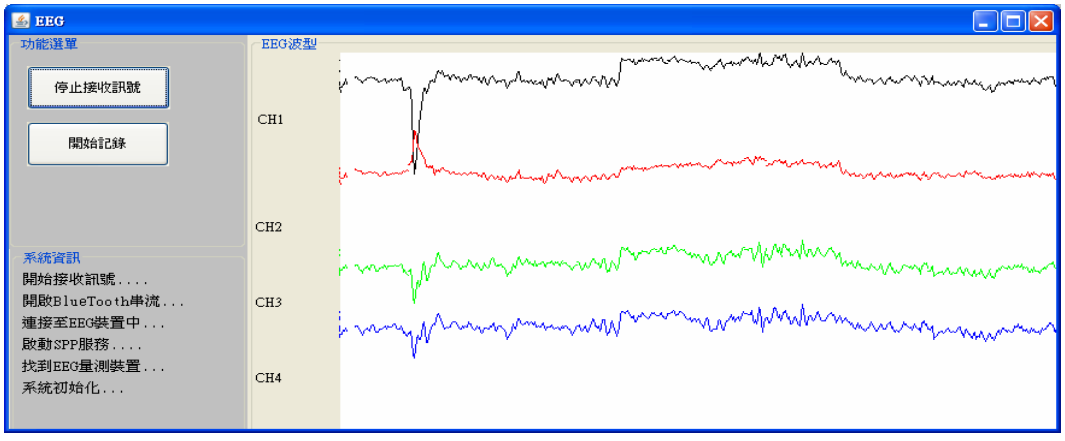


(b)

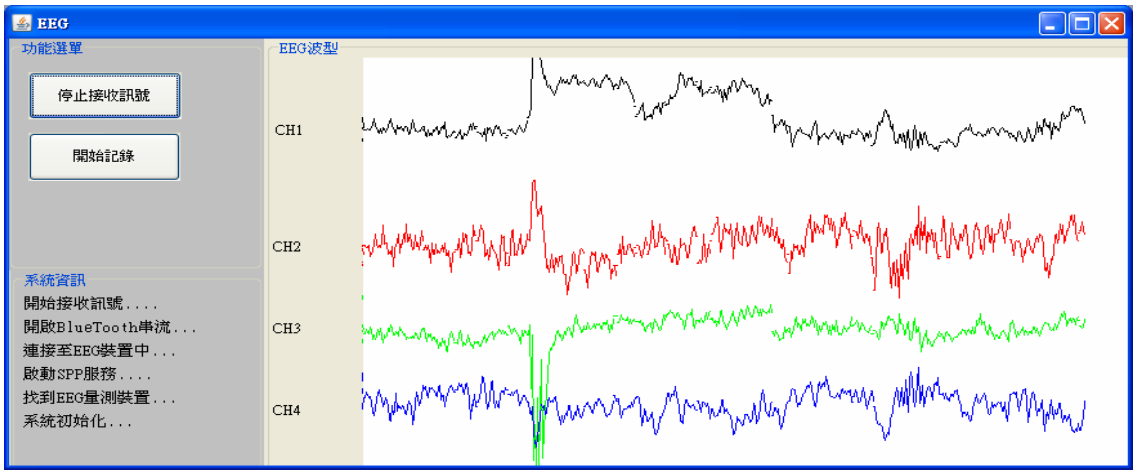
Fig. 2-74 (a) Mixed signal (b) ICA signal.

(3) EEG in GUI

EEG Pattern One



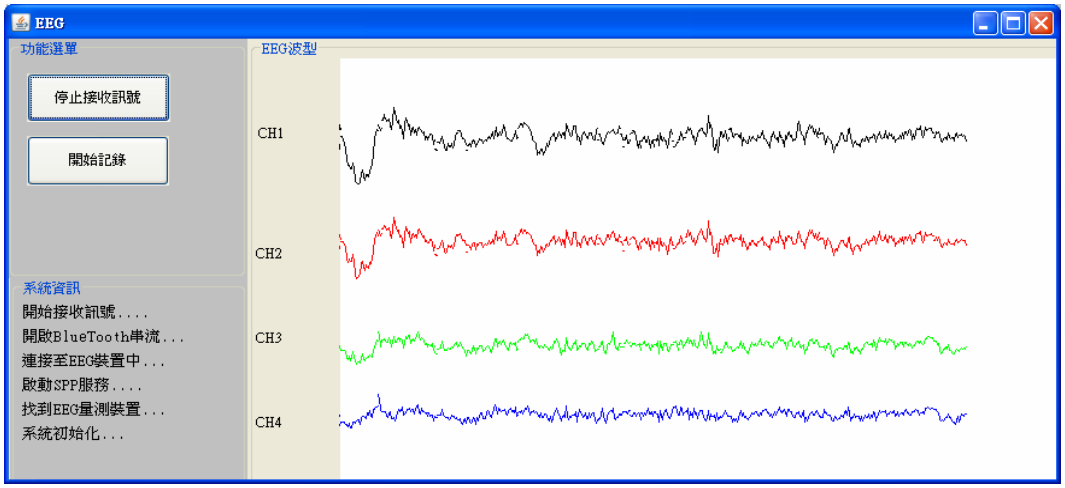
(a)



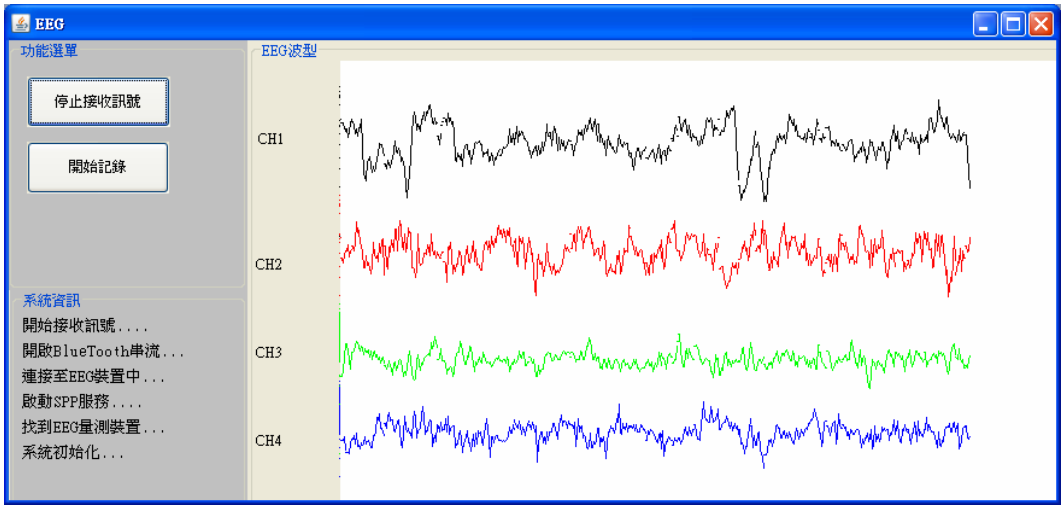
(b)

Fig. 2-75 (a) Mixed signal (b) ICA signal.

EEG Pattern 2



(a)

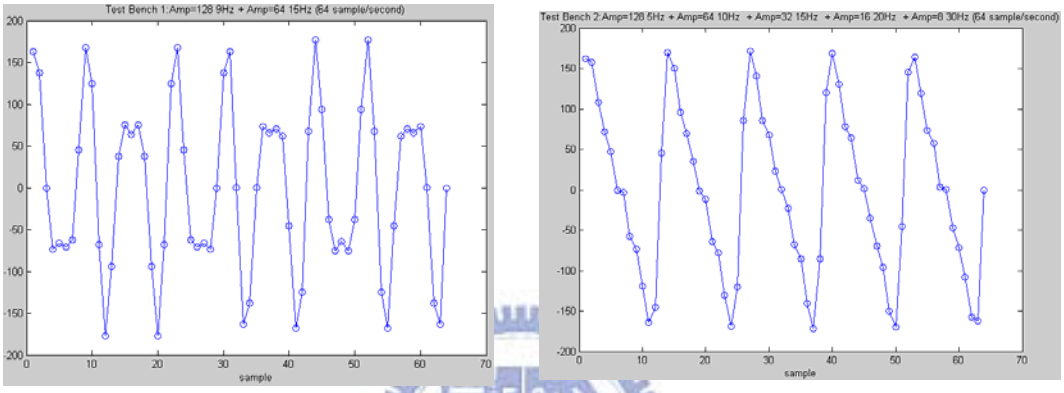


(b)

Fig. 2-76 (a) Mixed signal (b) ICA signal.

2-6-4 Simulation result of FFT

In order to verify the precision of the 64-points FFT, we use some sampled (sample rate 64Hz) combinational sinusoidal wave as our test bench. The test signal one is a 9Hz and 15Hz combinational sinusoidal wave. The test signal two is a 5Hz, 10Hz, 15Hz, 20Hz and 30Hz combinational sinusoidal wave. Test bench one and two are shown in Fig. 2-77. The circles in Fig. 2-77 are the sampled value under 64Hz sample rate.



(a) (b)

Fig. 2-77 (a) Test bench 1: $128\sin(18\pi t)+64\sin(30\pi t)$.

(b) Test bench 2:

$128\sin(10\pi t)+64\sin(20\pi t)+32\sin(30\pi t)+16\sin(40\pi t)+8\sin(60\pi t)$.

The result of hardware FFT is compared with FFT function in MATLAB library. The comparison of test bench one is shown in Fig. 2-78, in the left is the result of hardware FFT unit which features two peaks of 9Hz and 15Hz, in the right is the result of MATLAB FFT. As you can see, hardware result is quite similar with MATLAB version. The relative RMSE (root mean square error) of test bench one is only 0.330%. Figure 2-79 shows the comparison of test bench two. Note, hardware result reflects the composite 5Hz, 10Hz, 15Hz, 20Hz and 30Hz frequencies of input, also it is highly alike to the result from. The relative RMSE of test bench two is 0.436%.

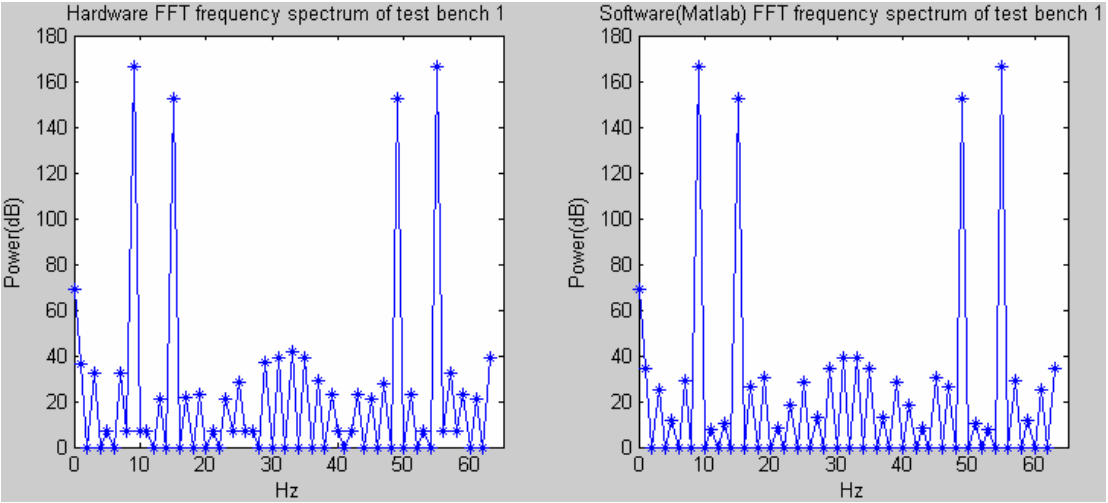


Fig. 2-78 Test bench 1 comparison of hardware and MATLAB result.

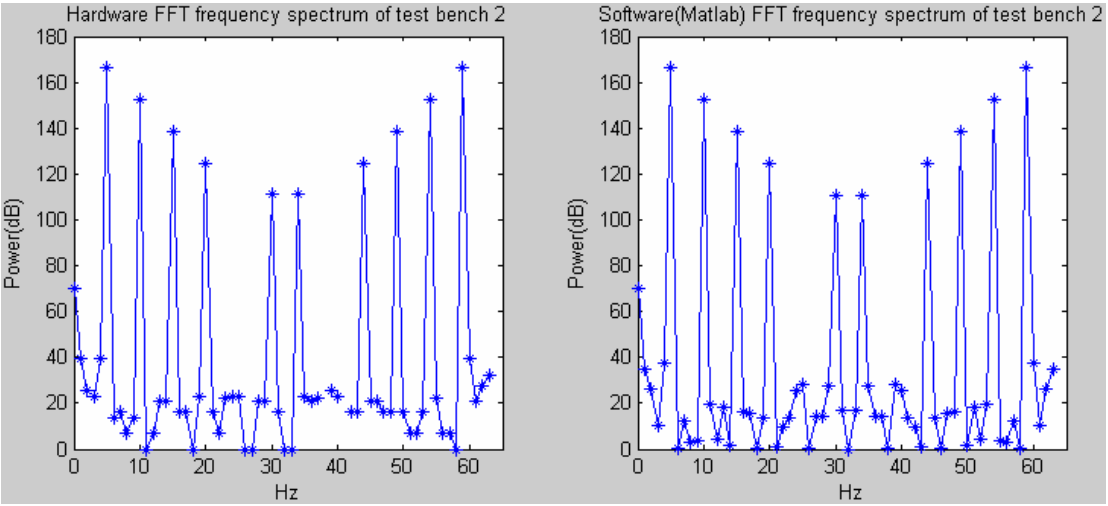


Fig. 2-79 Test bench 2 comparison of hardware and MATLAB result.

Figure 2-80 is the timing diagram of post simulation with test bench one and two. Observing the part of test bench one, rim A shows signals about input, including input datum and input control signal. Rim B includes datum of output (real part and imagine part) and control signals such as out_valid indicating validity of outputs, out_sop and out_eop implying start and end position of output stream. Fram C presents signal of stage controller. Fram D contents some internal signals about memory scheduler, they are, address, data, write enable which are sent to data RAM.

All these signals are asserted and deasserted properly. Rim E to rim H are corresponding sets of test bench two, which are correct too.

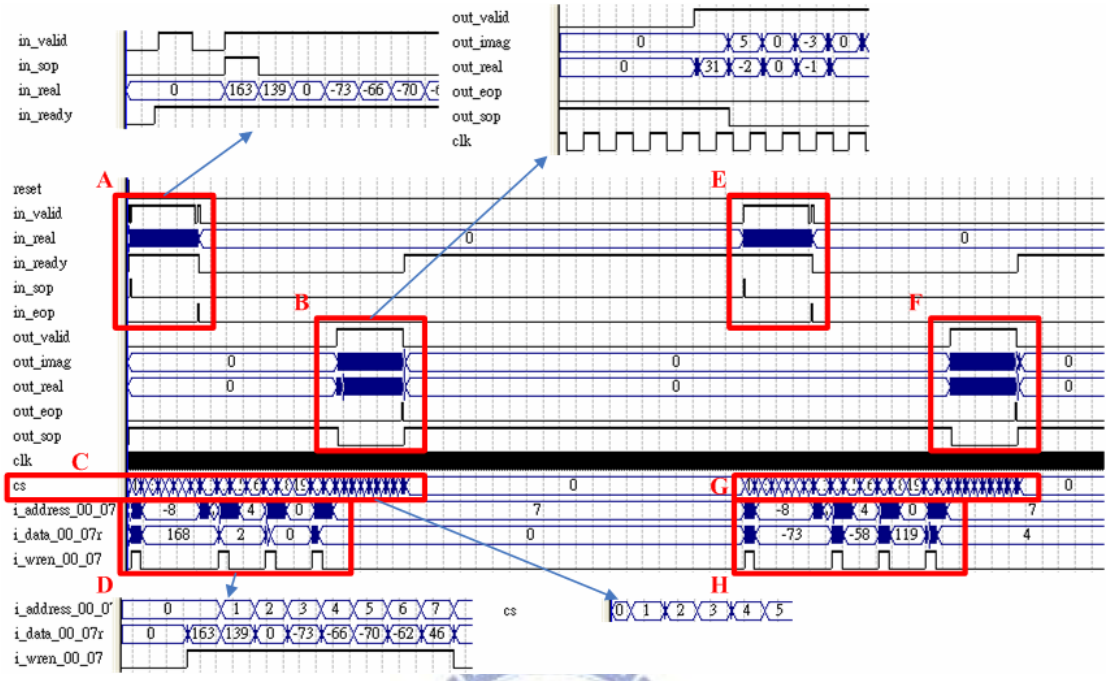


Fig. 2-80 Timing diagram of post simulation of the FFT.

Chapter3
FPGA Implementation

3-1 Introduction of Implementation Platform

In this chapter, we will explain the implementation flow and environment of the system, the architecture and specification is mentioned in previous chapter.

3-1-1 Features of Altera DE2 Board [22]

The DE2 board features a powerful Altera Cyclone™ II FPGA chip. All important components on the board are connected to the pins of this chip, allowing the user to configure the connection between the various components.

Table 3-1 Hardware features of DE2 board.

Hardware Feature	Description
512-Kbyte SRAM	Where flow control program and hardware driver is stored in.
8-Mbyte SDRAM	Half storage space is used as VGA pixel buffer, rest of tem is for customized IP.
4-Mbyte Flash memory	Where GUI pictures file is stored in.
18 toggle switches	Monitoring mode switching.
8 7-Segment Displays	Showing incoming raw data.
27 User LEDs	Stands for some inner signal.
RS-232 transceiver and 9-pin connector	The media for EEG signal to be send form PC to DE2 board.
VGA DAC (10-bit high-speed triple DACs) with VGA-out connector	Converts digital RGB value to analog output.
50-MHz and 27-MHz oscillator	Clock sources.
USB Blaster	Programs hardware design to FPGA and downloads data to memory on DE2.

Figure 3-1 depicts the layout of the DE2 board and indicates the location of the connectors and key components used in this work. In addition to FPGA chip, Table 3-1 lists hardware provided on the DE2 board and its corresponding purpose in this work. Following is more detailed information about the Cyclone™ II FPGA chip.

- 33,216 LEs (logic elements)
- 105 M4K RAM blocks
- 483,840 total RAM bits
- 70 embedded 9-bit multipliers
- 4 PLLs
- 475 user I/O pins

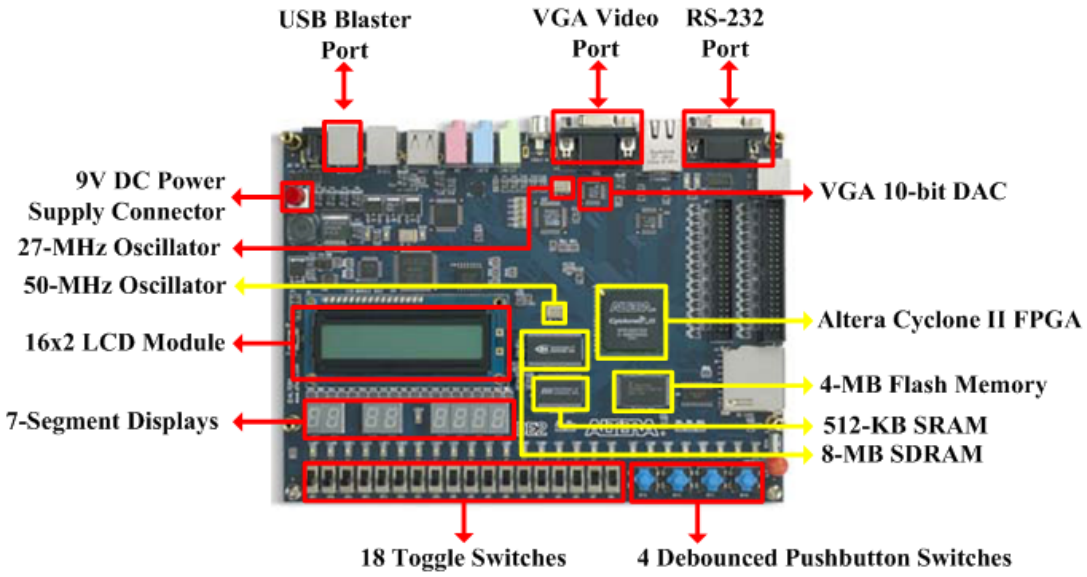


Fig. 3-1 The DE2 board.

3-2 Introduction of SOPC

The integration of the DE2 resources and customized hardware components (ICA, FFT, SDRAM controller, VGA controller) mostly is done by SOPC builder. In this section, we will briefly introduce the concept of SOPC and its tool kits SOPC builder.

3-2-1 SOPC

SOPC (System on a Programmable Chip) is a programmable system developed by Altera. Using tool kits SOPC builder is able to automate connecting soft-hardware (IP) components to create a system that runs on FPGA chips. Before the development of SOPC, application-specific integrated circuit (ASIC) technology was generally applied in SOC industry. The difference between ASIC and FPGA makes them applied for distinct purpose. ASIC is used for customized mass production, it needs a great amount of product to share out expansive design and manufacture cost. On the contrary, although FPGA has higher unit price, the programmable property is suitable for non-mass production application. SOPC is a fusion SOC technology based on programmable logic device (PLD) and ASIC, so far the cost of 65nm process ASIC is tremendous expensive, thus the integration of soft-core or hard-core CPU, DSP, memory, peripheral I/O and PLD in SOPC chip has great advantage of application flexibility and design cost.

SOPC Builder is a software tool kits developed by Altera. SOPC Builder incorporates a library of pre-made components (including the NIOS II soft processor, memory controllers, interfaces, and peripherals) and an interface for incorporating custom ones. Interconnections are made though the Avalon bus. Bus arbitration, bus width matching, and even clock domain crossing are all handled automatically when SOPC Builder generates the system. A GUI is the only thing used to configure the soft-hardware components (which often have many options) and to specify the bus topology.

Figure 3-2 shows a system A, which is composed by CPU, DSP, FPGA, I/O (RS-232, GPIO, UART…etc), and off-chip memory. Lowering power consumption, cost, and complexity is possible through proper planning, placing and routing. Components like CPU, DSP, FPGA and off-chip memory controller can be integrated

into single SOPC system, say system B.

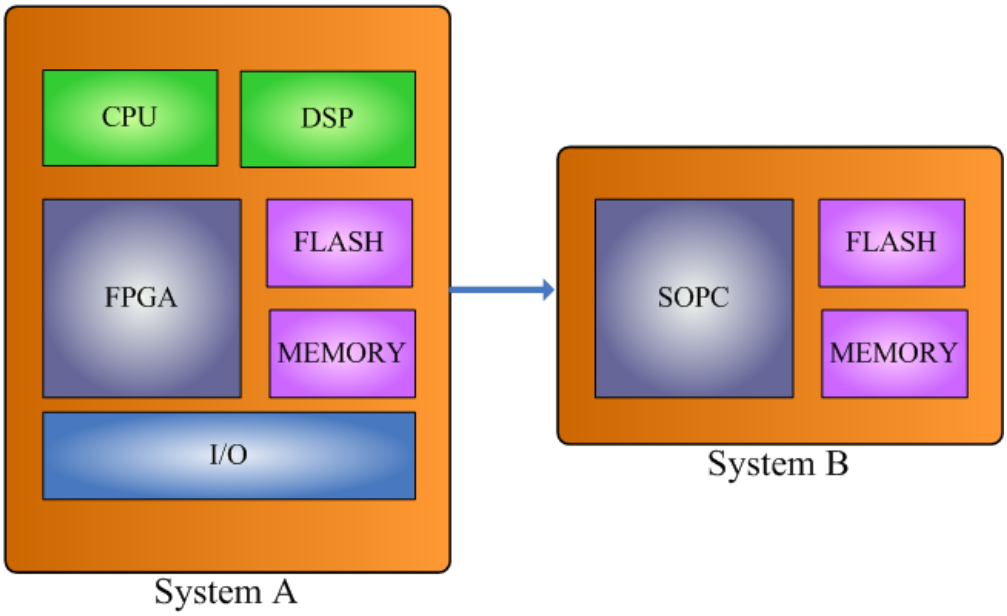


Fig. 3-2 SOPC concept chart.

3-2-2 SOPC Builder [23]

SOPC Builder is a system development tool for creating complete systems (including processors, peripherals, and memories). SOPC Builder enables one to define and generate a complete system on a programmable chip in much less time than using traditional methods. SOPC Builder is not only the tool for creating systems based on the Nios™ II processor, but more than a Nios II system builder; it is a general-purpose tool for creating systems that may or may not contain a processor.

SOPC Builder automates the task of integrating customized hardware components into a larger system. Using traditional SOC design methods, one must manually write top-level HDL files that wrap all pieces of the system together. Using SOPC Builder, you can specify the system components in a GUI (Fig. 3-3), and SOPC Builder generates the interconnect logic (Avalon bus system) automatically. SOPC Builder outputs HDL files that define all components of the system, and a top-level HDL design file that connects all the components together.

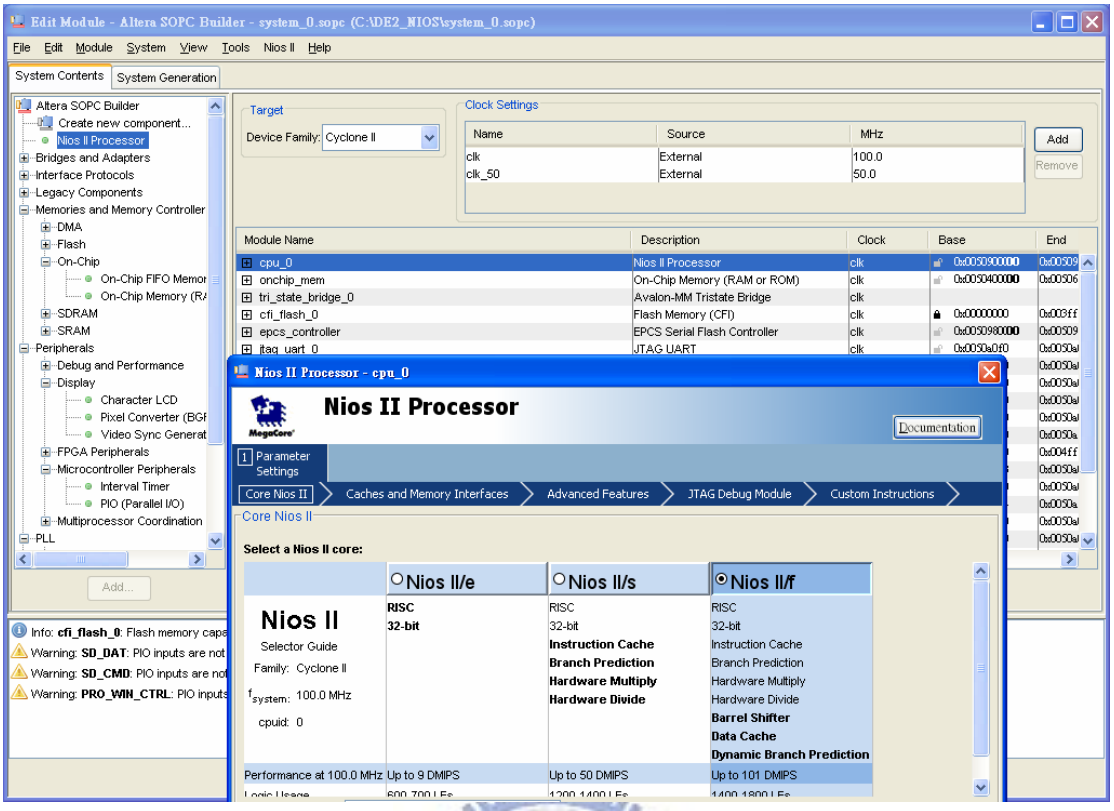


Fig. 3-3 SOPC Builder GUI.

3-3 Introduction of Avalon BUS Interface [24]

The Avalon bus is the bus system used in this work. It is a simple bus architecture for connecting on-chip processors and peripherals together into a SOPC. The Avalon bus is an interface that specifies the port connections between master and slave components, and specifies the timing by which these components communicate.

3-3-1 SOPC Builder and Generation of the Avalon Bus

SOPC Builder generates the system module, which is the on-chip circuitry that comprises the Avalon bus, master peripherals, and slave peripherals. SOPC Builder has a GUI for adding master and slave peripherals to the system module, configuring the peripherals, and then configuring the Avalon bus to connect peripherals together. The Avalon bus module is the backbone of a system module. The Avalon bus module

is the sum of all control, data and address signals and arbitration logic that connect together the peripheral components making up the system module. The Avalon bus module implements a configurable bus architecture, which changes to fit the interconnection needs of peripheral IPs in this work.

3-3-2 Avalon Peripherals

An Avalon peripheral on the Avalon bus is a logical device, either on-chip or off-chip, that performs some system-level task, and communicates with other system components through the Avalon bus. Peripherals are modular system components. Avalon peripherals can be memories and processors, as well as traditional peripheral components, such as a UART, timer or bus bridge. Any user logic can also be an Avalon peripheral, as long as it provides address, data and control signal interfaces to the Avalon bus. A peripheral connects to specific ports on the Avalon bus module allocated for that peripheral.

The peripheral may also have user-defined ports in addition to the Avalon address, data and control signals. These signals connect to custom logic external to the system module. The roles of Avalon peripherals are classified as either a master or slave. A master peripheral is a peripheral that can initiate bus transfers on the Avalon bus and has at least one master port which connects to the Avalon bus module. It may also have a slave port, which allows the peripheral to receive bus transfers initiated by other master peripherals on the Avalon bus. A slave peripheral is a peripheral that only accepts bus transfers from the Avalon bus, and cannot initiate bus transfers. Slave peripherals, such as memory devices usually have only one slave port.

3-3-3 Avalon Bus Signals

Fundamentally, the Avalon signal types are classified as either slave port signals or master port signals. Therefore, the signal types used by a peripheral are determined first and foremost by the master/slave role of the port. Each master or slave port may have up to one of each signal type. The set of signal types used by an individual master or slave port is dependent on the design of the peripheral. For example, the design for an output-only PIO slave peripheral would define only ports for write transfers (the output direction), but no ports for read transfers.

Table 3-2 shows a list of the signal types available to an Avalon slave port in this work. The signal direction is from the perspective of the peripheral. For example, the clock signal clk (listed as an input) is an input to the slave peripheral, but it is an output from the Avalon bus module.

Table 3-2 List of Avalon slave signals.

Signal Type	Width	Direction	Required Description
clk	1	in	Global clock signal for the system module and Avalon bus module. All bus transactions are synchronous to clk.
address	32	in	Address lines from the Avalon bus module.
read	1	in	Read request signal to slave.
readdata	32	out	Data lines to the Avalon bus module for read transfers.
write	1	in	Write request signal to slave.
writedata	32	in	Data lines from the Avalon bus module for write transfers.

3-3-4 Slave Read Transfers on the Avalon Bus

The slave read transfer is initiated by the Avalon bus module, and transfers one unit of data, the full width of the peripheral's data port, from the slave port to the Avalon bus module. A slave read transfers may or may not have latency.

Figure 3-4 shows an example of the read transfer with no latency. The bus transfer starts on a rising clock edge, no wait states are incurred, and the read transfer completes on the next rising clock edge. For the transfer to complete in a single bus cycle, the target peripheral must immediately and asynchronously output the contents of the addressed location to the Avalon bus module. On the first rising edge of clk, the Avalon bus passes the address and read_n signals to the target peripheral. The Avalon bus module decodes address internally. Once read is asserted, the slave port drives out its readdata as soon as it is available. Finally, the Avalon bus module captures the readdata on the next rising edge of the clock.

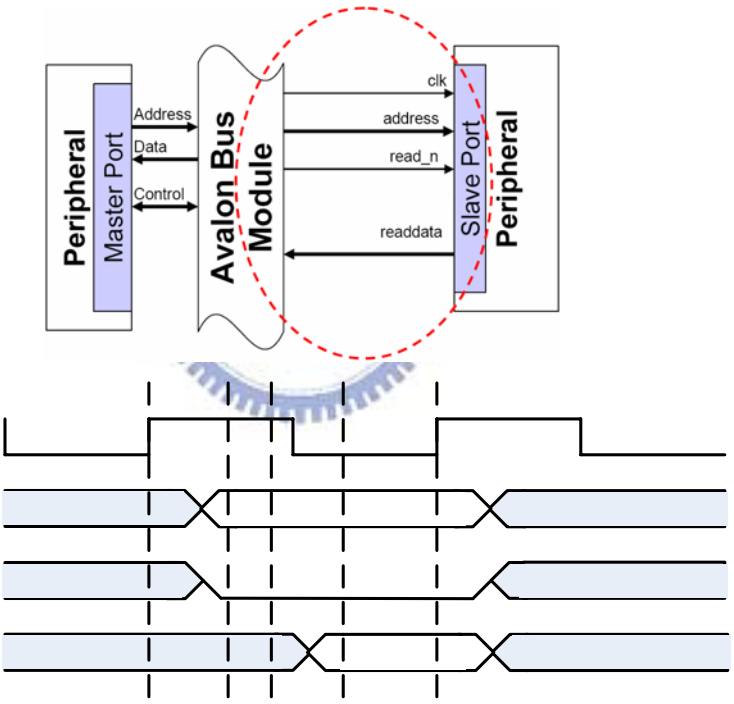


Fig. 3-4 Slave read with no latency timing.

(A) First bus cycle starts on the rising edge of clk.

(B) Registered outputs address and read_n from Avalon bus to slave are valid

(C) Avalon bus decodes address.

(D) Slave port returns valid data during the first bus cycle.

(E) Avalon bus captures readdata on the next rising edge of clk, and the read transfer ends here. The next bus cycle could be the start of another bus transfer.

3-3-5 Slave Write Transfers on the Avalon Bus

The slave write transfer is initiated by the Avalon bus module, and transfers one unit of data from the Avalon bus module to the slave port. A slave read transfers may or may not have latency. Figure 3-5 shows a slave write transfer. There are zero wait states, and no setup-time or hold-time wait states. The Avalon bus module presents address, writedata and write_n. The slave port captures the address, data and control on the next rising clock edge, and the write transfer terminates immediately. The entire transfer takes only one bus cycle. The slave peripheral may then take additional clock cycles to actually process the write data after the transfer terminates. If the peripheral cannot sustain consecutive write transfers on every bus cycle, then additional design considerations are required to generate wait states.

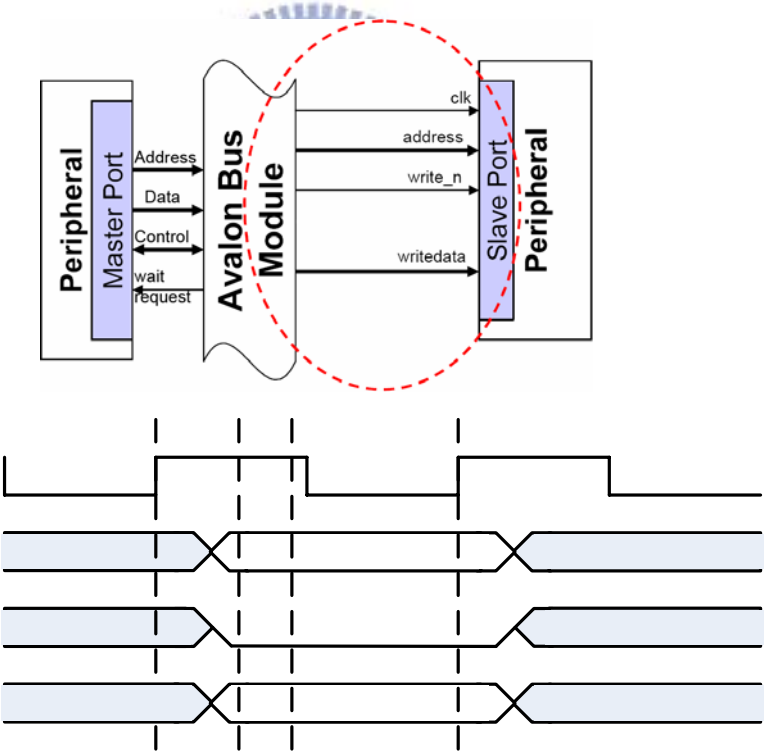


Fig. 3-5 Slave write with no latency timing.

(A) Write transfer starts on the rising edge of clk.

(B) Registered writedata, address, and write_n signals from Avalon bus are valid.

(C) Avalon bus decodes address and asserts valid chipselect to slave.

(D) Avalon bus captures writedata, address and write_n on the rising edge of clk, and the transfer terminates. Another read or write transfer may follow on the next bus cycle.

3-4 Integration of Customized IP into Avalon BUS

After introducing read/write timing specification of Avalon bus, it is evidently that an interface between custom peripheral and bus system is necessary which is as know as a wrapper. Following are the details how to design the wrappers for customized IPs.

3-4-1 Integration of VGA/SDRAM Controller into Avalon BUS

The outline of SDRAM controller is depicted in Fig. 3-6. As it shows, each pair of read/write request FIFO (port) deals with read/write of a corresponding logic bank. Note that each read/write transfers 16-bit data.

The VGA controller takes three (R, G, B respectively) 10-bit inputs at a time. In order to transfer 30-bit at a time, we connect two read ports to satisfy 30-biits bandwidth. Also, we connect two write ports to write RGB value in one write. The RGB value and internal memory storage mapping is shown in Fig. 3-7. As one can see, 10-bit R value are put in first ten bits of a memory element in the third bank, the first five bits of G are put in next five bits of the same element, we fill a “0” in the last bit (bit 15). Similarly, the second half of G (bit 5 to bit 9) are put in first five bits of a memory element in the second bank, 10-bit B value are put in following ten bits of the same element, likewise, a “0” is filled in the rest bit of the same memory element.

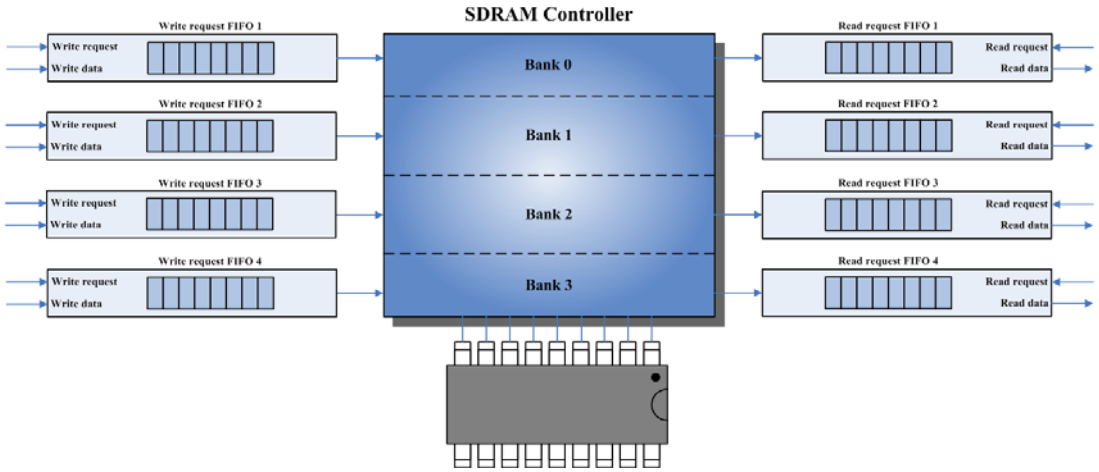


Fig. 3-6 SDRAM controller outline.

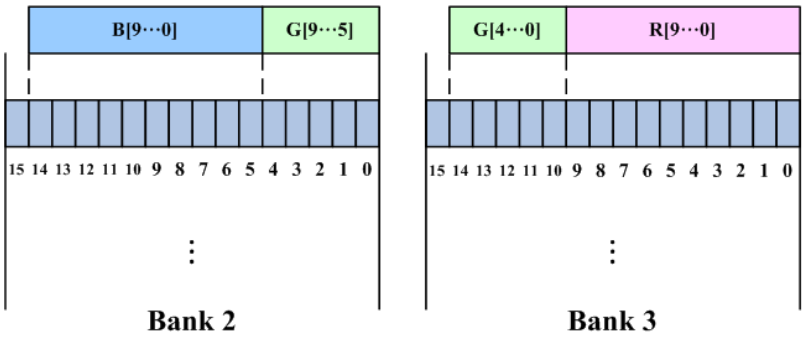


Fig. 3-7 Internal memory storage mapping of RGB with two logic banks.

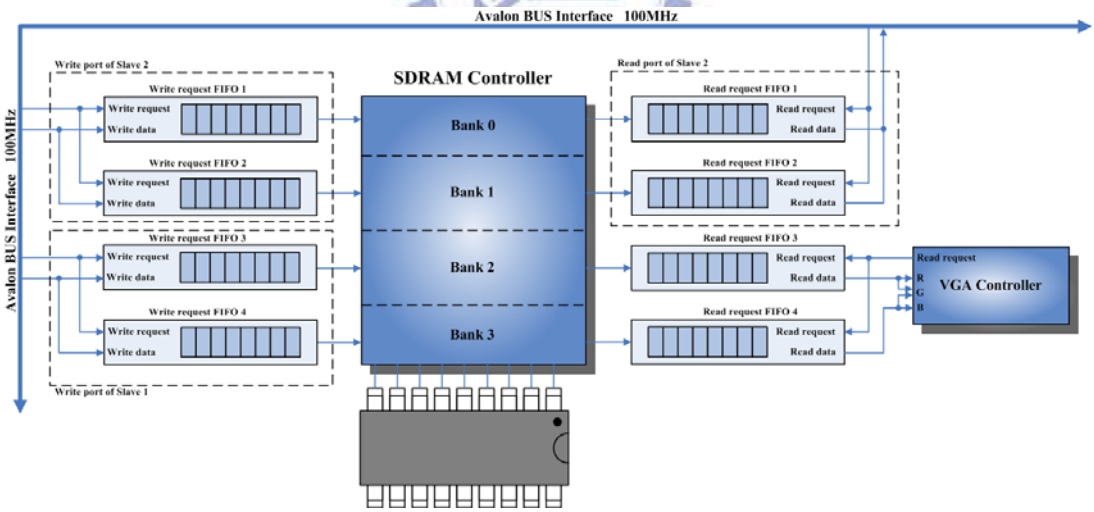


Fig. 3-8 Wrapper of combinative SDRAM and VGA controller.

The structure of the combination of SDRAM and VGA is shown in Fig. 3-8. Two read ports are connected internally providing VGA RGB value. The corresponding two write ports are combined together as a slave write port. Since the

width of data bus is 32-bit, we combine the rest other two read/write ports to be another slave read/write port.

3-4-2 Integration of ICA into Avalon BUS

The system basically works under 100MHz and ICA unit has a limited working frequency up by 60MHz. Therefore, under different operating frequency between ICA module and system bus, an interface to eliminate unmatchable clock frequency between bus interface and ICA unit is necessary. Two dual-clock FIFOs [25] are placed to block input port and output port of ICA from system bus. The structure of this ICA wrapper is shown in Fig. 3-9. Between ICA input port and system bus is a 512-entries 32-bit dual-clock FIFO with 100MHz write clock and 50 MHz read clock, the input controller controls the FIFO to send input to ICA with proper input valid signal. Although write frequency is two times faster than read frequency, FIFO overflow is not easily happens. That is because the system is applied for EEG signal with only 64Hz sample rate. Between ICA output port and system bus is a 128-entries 32-bit dual-clock FIFO with 50MHz write clock and 100MHz read clock.

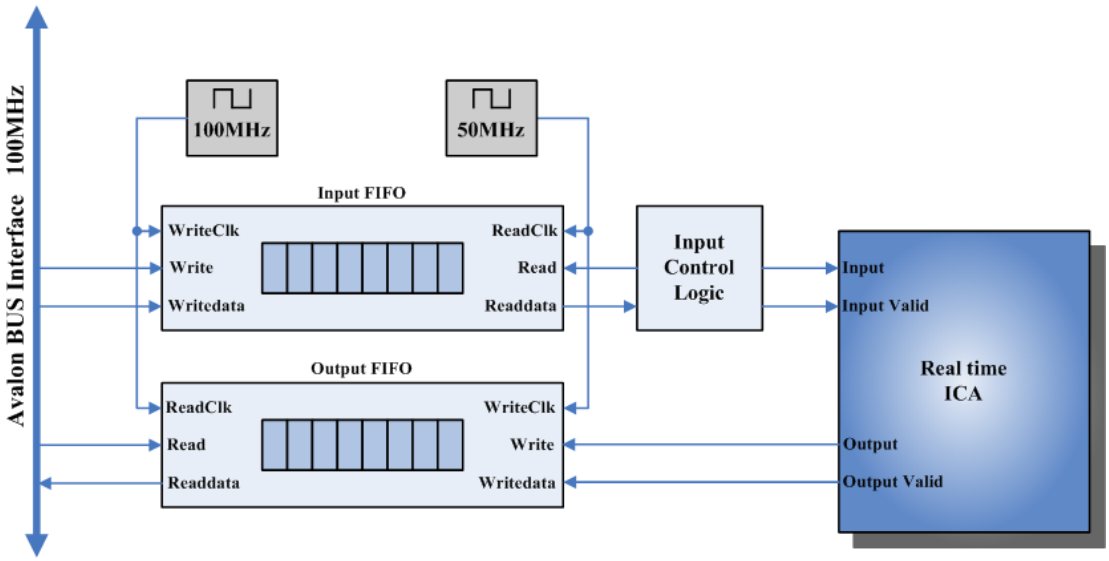


Fig. 3-9 Wrapper of ICA.

3-4-3 Integration of FFT into Avalon BUS

Frankly speaking, ICA outputs four channels 8-bit datum, that is, 32-bit at a time. It seems that four sets of FFT units should be used in this system. But considering about area and hardware cost, four channels' output should use only one FFT unit by turns. Fortunately, the system is applied on EEG signal under a slow frequency thus four channels ICA outputs can still meet timing constrain even alternatively using only one FFT unit.

The architecture of FFT wrapper is shown in Fig. 3-10. For the 32-bit width of data bus, once a write request arrives, write data is dispatched into four channels (each channel has 8-bit width). The input control logic then sends first channel to the input of FFT. The computation result of first channel is pushed into its corresponding output FIFO. Until FFT hardware is no longer being used, input control logic sends next channel as FFT's input. The control of the rest channels is as well as first and second channels. The four output channels are treated as eight independent slaves (four channels real part and four channels imaginary part).

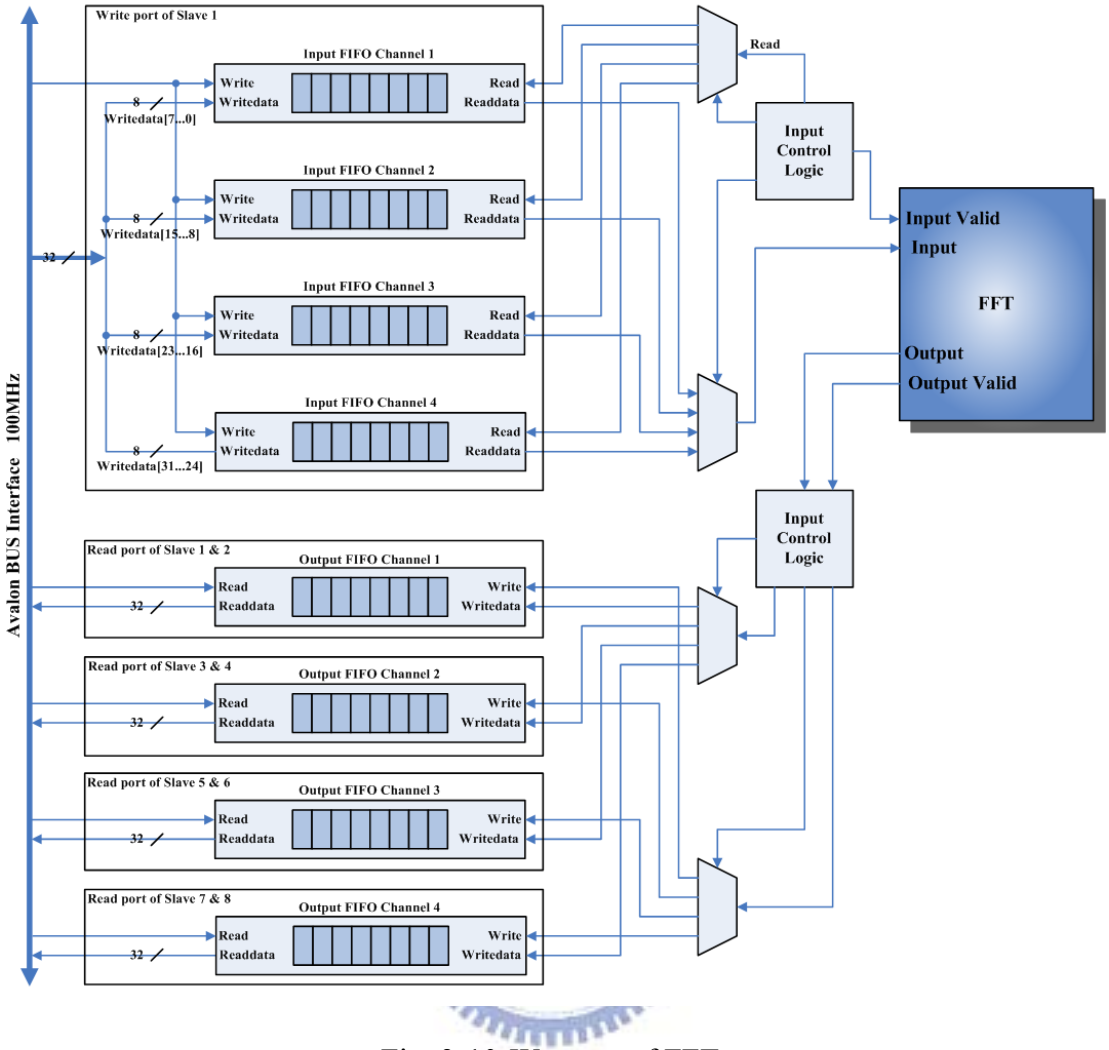


Fig. 3-10 Wrapper of FFT.

3-5 Simulation Result of Wrappers

In this section we survey the action of wrappers through the results of post simulation.

3-5-1 Simulation Result of VGA/SDRAM Controller Wrapper

The simulation of integrated VGA/SDRAM controller wrapper is shown in Fig. 3-11. After initial sequence, we sequentially wrote a RGB value of a pixel in a 640×480 frame. The write request is transfer through system bus, write request (avs_s1_write) together with write data (avs_s1_writedata) are shown in rim A. Once

SDRAM get the write requests, internal controller generated corresponding control signals which are shown in rim C. In rim D, after reset signal, VGA controller just acted spontaneously, that is, regularly generated horizontal or vertical synchronization signals (VGA_VS, VGA_HS). Also, once the horizontal counter and vertical counter are both under the scope of display (that means they are not in the back porch or front porch) VGA controller generated read request (oRequest) to SDRAM controller. Then SDRAM controller responded corresponding controller for the read requests, which are shown in rim C.

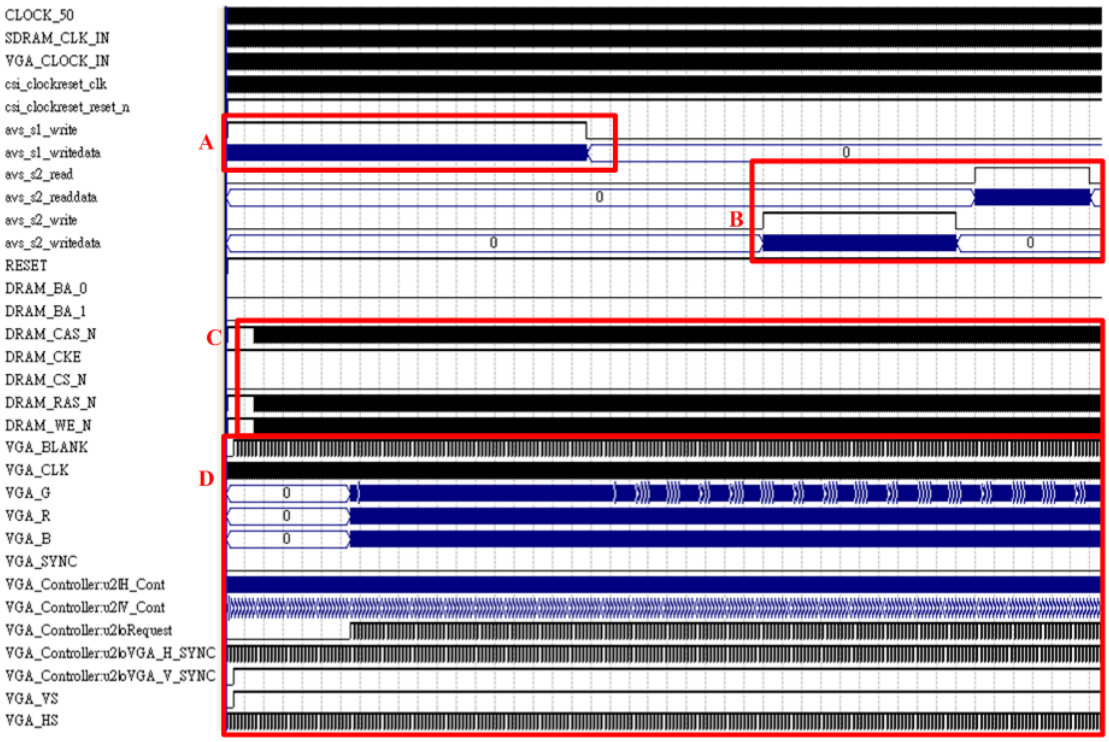


Fig. 3-11 Simulation of integrated VGA/SDRAM controller wrapper.

3-5-2 Simulation Result of ICA Wrapper

The simulation result of ICA wrapper is shown in Fig. 3-12. Rim A shows that simulation started with 512 consecutive writes from Avalon bus under 100MHz. Rim B and rim c shows that input control logic (Fig. 3-9) sent inputs with proper input valid signal to ICA as input FIFO began not to be empty anymore. The inputs sent to

ICA were under 50MHz transfer frequency. After ICA finished its training and computing (training process is showed in rim D), the output valid signal ACT_RAM_ENABLE (rim E) was pulled up to push outputs to output FIFO with 50MHz write frequency. Note the 100MHz clock was generated by feeding 50MHz clock to PLL to double it. Rim F shows read request from system bus and read data outputs with two cycle wait state. The zoom in of 512 consecutive write requests and inputs data with valid signal to ICA in the beginning is shown in Fig. 3-13. Figure 3-14 shows output datum with output valid signal were written to output FIFO under write frequency 50 MHz. The read requests from system bus read output FIFO with 100MHz bus frequency which is shown in Fig. 3-15. A two cycles read latency of a read request should be noticed.

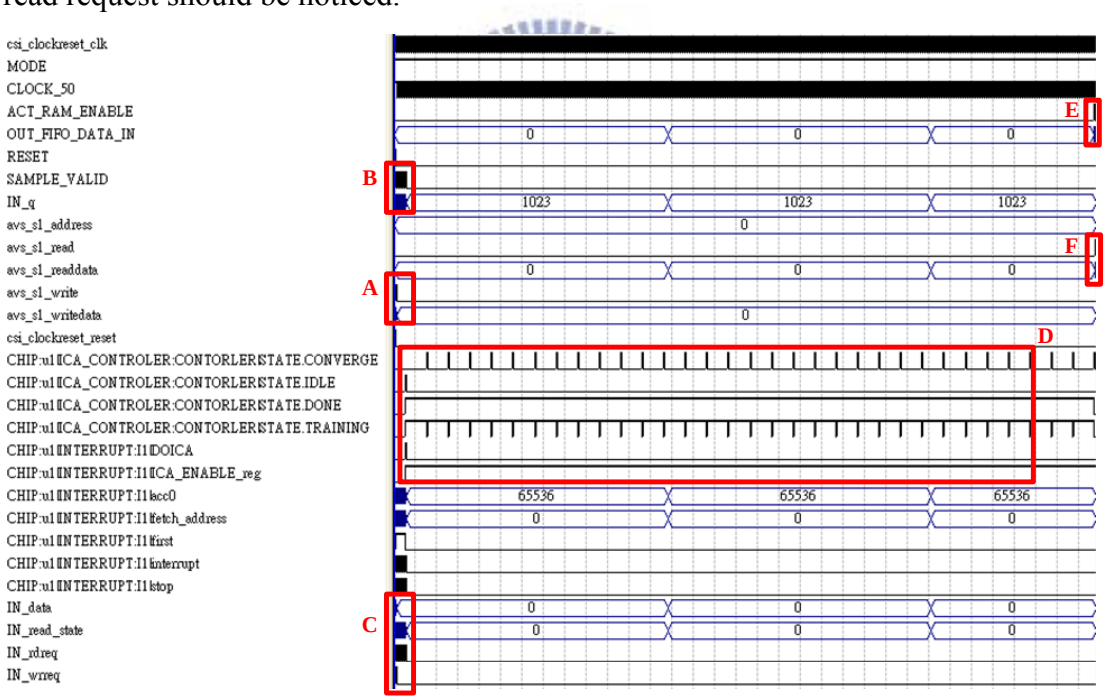


Fig. 3-12 Simulation of wrapper of ICA unit.

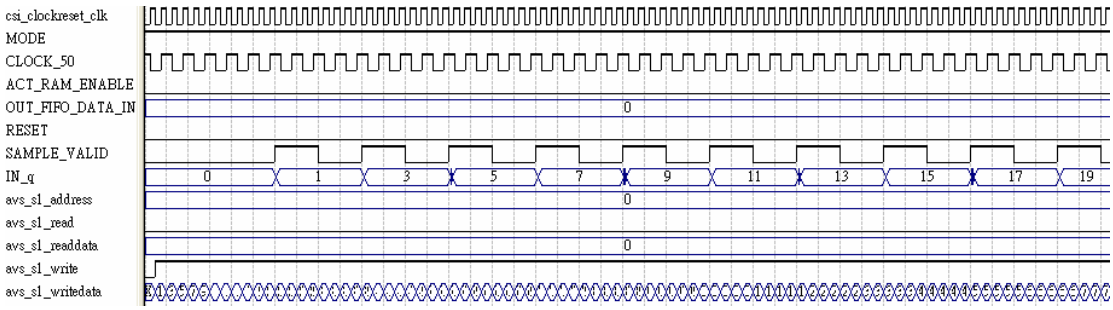


Fig. 3-13 Zoom in of rim B.

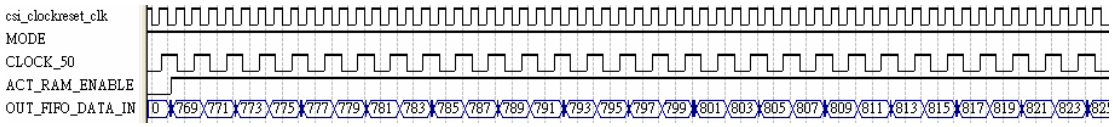


Fig. 3-14 Zoom in of rim E.

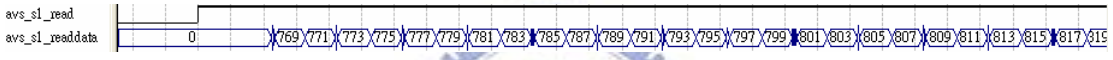


Fig. 3-15 Zoom in of rim F.

3-5-3 Simulation Result of FFT Wrapper

The simulation result of FFT wrapper is shown in Fig. 3-16. Look at rim B, in the beginning a sequence of write requests is driven from bus to input FIFOs (Fig. 3-10). In rim A, after gathered 64 inputs, input control logic sent inputs with proper control (sop, eop) signals of channel one to FFT unit, once FFT started to generate output (out_r, out_i) inputs and control signal of next channel was sent to FFT. The inputs of four channels were sent one channel after another.

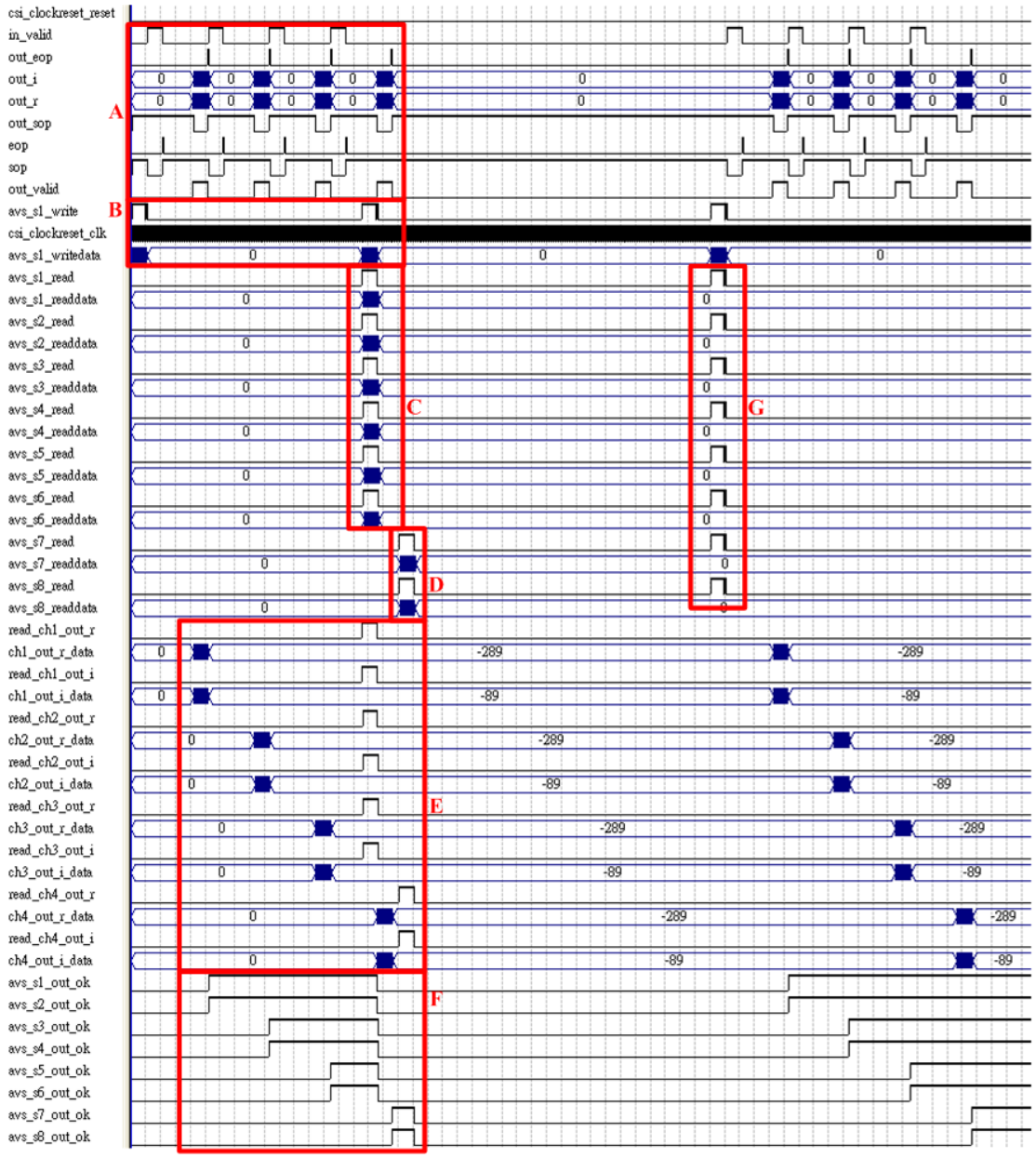


Fig. 3-16 Simulation of wrapper of FFT unit.

Rim E shows outputs from FFT were transferred to output FIFOs by turn, then avs_sn_ok (where n represents which slave) signals were asserted to denote output FIFOs were ready for read request. Rim F shows the assertion of avs_sn_ok signals. Rim C and D show the read datum avs_sn_readdata were responded for read requests, yet read requests were valid only when avs_sn_ok signals were asserted. Rim G just shows if a avs_sn_ok was not pulled up, read request would get nothing. The zoom in

of rim A to rim E are shown in Fig. 3-17 through Fig. 3-20. Note Fig. 3-18 and Fig. 3-19 show there is a five cycle read latency for the read requests.

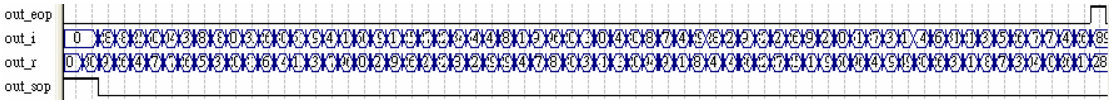


Fig. 3-17 Zoom in of rim A.

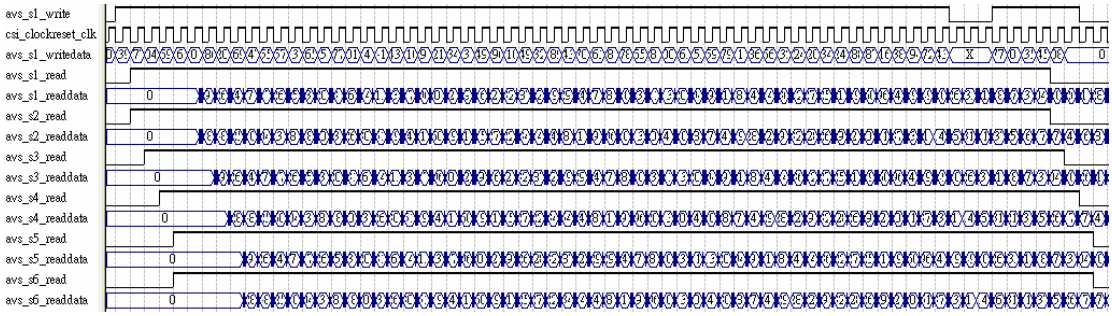


Fig. 3-18 Zoom in of rim C.

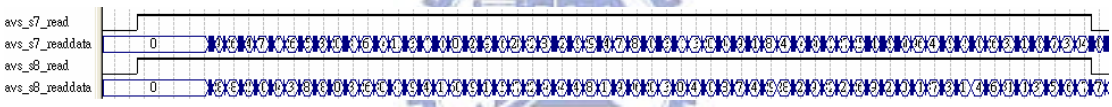


Fig. 3-19 Zoom in of rim D.

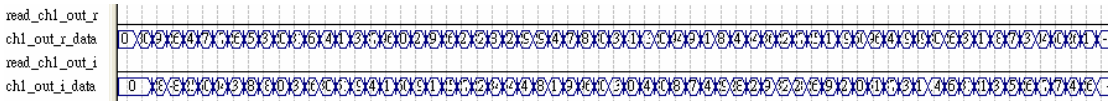


Fig. 3-20 Partial zoom in of rim E.

3-5-4 Base Address Definition of System Components

So far, all components are ready to connect on system bus. We use SOPC to automatically generate interconnection of system bus with system components. Figure 3-21 shows the configuration of whole system and base address of all components. The definition of base address provides higher level software an entrance to access a certain component.

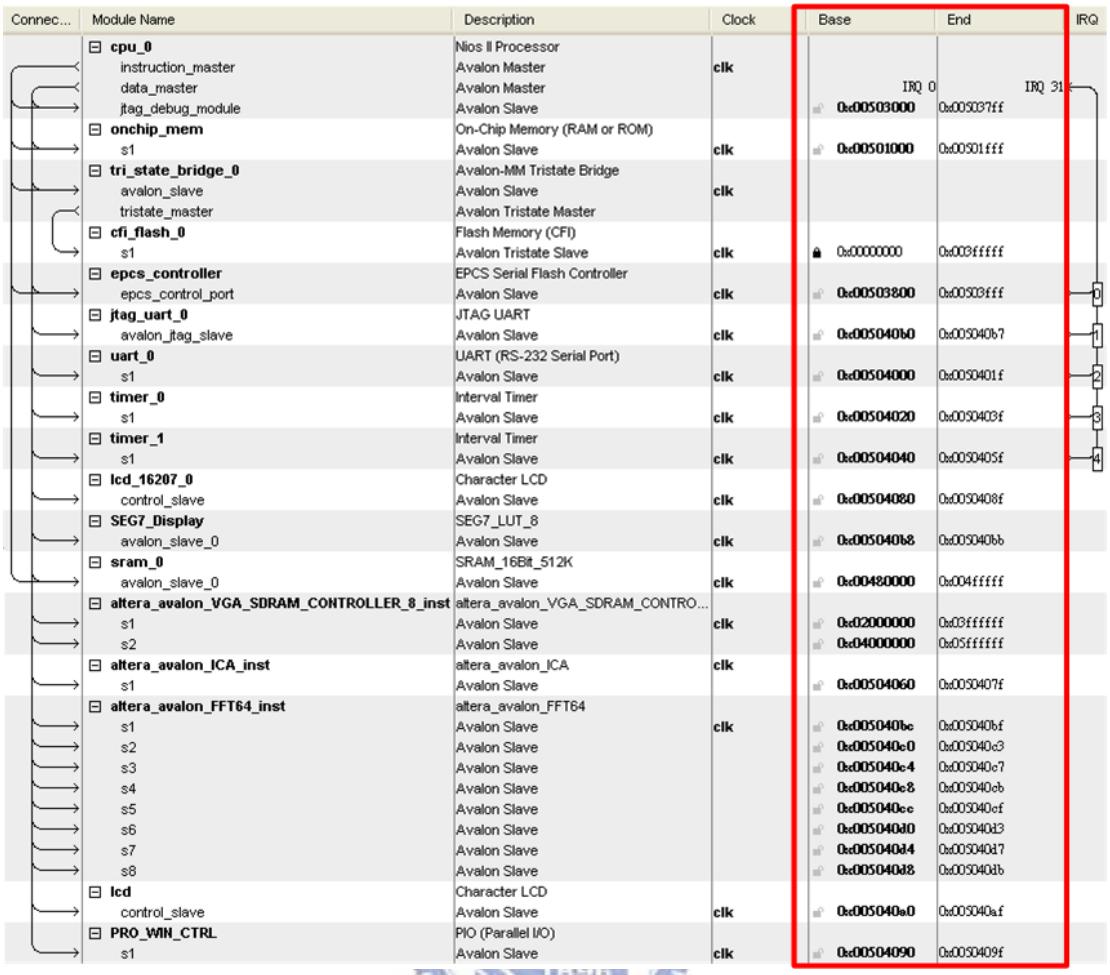


Fig. 3-21 Base address definition of system components.

3-6 Summary of Hardware Resources after Synthesis

In this section, we scrutinize the hardware resource configuration of each part of system.

3-6-1 Synthesis Result of VGA/SDRAM Controller Wrapper

The synthesis result of VGA/SDRAM controller wrapper is shown in Fig. 3-22, which takes 1,405 logic elements on the FPGA. Other details are depicted in synthesis report. Timing report is shown in Fig. 3-23. The speed of VGA/SDRAM controller wrapper is up to 8139 MHz actually.

Top-level Entity Name	altera_avalon_VGA_SDRAM_CONTROLLER_8
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	1,405 / 33,216 (4 %)
Total combinational functions	1,125 / 33,216 (3 %)
Dedicated logic registers	1,007 / 33,216 (3 %)
Total registers	1007
Total pins	258 / 475 (54 %)
Total virtual pins	0
Total memory bits	64,512 / 483,840 (13 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Fig. 3-22 Synthesis report of VGA/SDRAM controller wrapper.

Type	Slack	Required Time	Actual Time
Worst-case tsu	N/A	None	6.915 ns
Worst-case tco	N/A	None	13.697 ns
Worst-case tpd	N/A	None	4.643 ns
Worst-case th	N/A	None	-3.045 ns
Clock Setup: 'SDRAM_CLK_IN'	N/A	None	139.30 MHz (period = 7.179 ns)
Clock Setup: 'csi_clockreset_clk'	N/A	None	152.42 MHz (period = 6.561 ns)
Clock Setup: 'VGA_CLOCK_IN'	N/A	None	Restricted to 210.08 MHz (period = 4.760 ns)
Clock Setup: 'CLOCK_50'	N/A	None	266.74 MHz (period = 3.749 ns)
Total number of failed paths			

Fig. 3-23 Timing report of VGA/SDRAM controller wrapper.

3-6-2 Synthesis Result of ICA Wrapper

The synthesis result of ICA wrapper is shown in Fig. 3-24, which takes 15,015 logic elements on the FPGA. Other details are depicted in synthesis report. Timing report is shown in Fig. 3-25. In order to achieve real-time operation in real environment, we need to over-design the ICA including its wrapper. The speed of ICA wrapper is up to about 60 MHz.

Top-level Entity Name	altera_avalon_ICA
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	No
Total logic elements	15,015 / 33,216 (45 %)
Total combinational functions	13,888 / 33,216 (42 %)
Dedicated logic registers	5,954 / 33,216 (18 %)
Total registers	5954
Total pins	144 / 475 (30 %)
Total virtual pins	0
Total memory bits	36,864 / 483,840 (8 %)

Fig. 3-24 Synthesis report of ICA wrapper.

	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	9.353 ns
2	Worst-case tco	N/A	None	10.585 ns
3	Worst-case tpd	N/A	None	13.650 ns
4	Worst-case th	N/A	None	-3.355 ns
5	Clock Setup: 'PLL:P1|altpll:altpll_component|_clk0'	3.141 ns	50.00 MHz (period = 20.000 ns)	59.32 MHz (period = 16.859 ns)
6	Clock Setup: 'csi_clockreset_clk'	4.223 ns	100.00 MHz (period = 10.000 ns)	173.10 MHz (period = 5.777 ns)
7	Clock Hold: 'csi_clockreset_clk'	0.167 ns	100.00 MHz (period = 10.000 ns)	N/A
8	Clock Hold: 'PLL:P1|altpll:altpll_component|_clk0'	0.391 ns	50.00 MHz (period = 20.000 ns)	N/A
9	Total number of failed paths			

Fig. 3-25 Timing report of ICA wrapper.

3-6-3 Synthesis Result of FFT Wrapper

The synthesis result of FFT wrapper is shown in Fig. 3-26, which takes 6,625 logic elements on the FPGA. Other details are depicted in synthesis report. Timing report is shown in Fig. 3-27. The speed of FFT wrapper is up to about 120 MHz.

Top-level Entity Name	altera_avalon_FFT64
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	6,652 / 33,216 (20 %)
Total combinational functions	6,526 / 33,216 (20 %)
Dedicated logic registers	2,903 / 33,216 (9 %)
Total registers	2903
Total pins	355 / 475 (75 %)
Total virtual pins	0
Total memory bits	22,528 / 483,840 (5 %)
Embedded Multiplier 9-bit elements	64 / 70 (91 %)
Total PLLs	0 / 4 (0 %)

Fig. 3-26 Synthesis report of FFT wrapper.

	Type	Slack	Required Time	Actual Time
1	Worst-case tsu	N/A	None	10.101 ns
2	Worst-case tco	N/A	None	11.823 ns
3	Worst-case th	N/A	None	-3.341 ns
4	Clock Setup: 'csi_clockreset_clk'	N/A	None	119.32 MHz (period = 8.381 ns)
5	Total number of failed paths			

Fig. 3-27 Timing report of FFT wrapper.

3-7 Software Development

After defining base address of whole system, all components in the system can be accessed by specifying hardware base address with C-language macros. Through these C-language macros, one can directly read/write a customized IP which is connected with system bus (Avalon bus). The C-language macros provide us an foundation to transfer data from software to hardware, that is also the foundation forming the drivers of custom IPs in this work.

3-7-1 Main Program Procedural Flow

Due to lack of an operating system ported on this system, in other words, none is in charge of thread management and thread scheduling. Therefore, the main program in this work is a single thread procedure instead of multithread procedure.

In the beginning, we initialize the UART port and continuously read from UART in a nonblocking manner. The data we read from UART actually is a prerecorded EEG pattern used to simulate the electrodes connected directly on a human being. Once we gather a valid ICA input, we write the data through ICA driver, following, we monitor the output from ICA by continuously reading through ICA driver. Then we write the ICA output to 64-points FFT through FFT driver, since the wrapper automatically split ICA 32-bit input into 4-channels 8-bit input, FFT module will have four channels outputs which are buffered in four different slaves.

Thus we have to read four slaves by turn after a write to FFT. Once ICA result and FFT result are gathered, the procedure monitors some control switches form FPGA to select the display mode of GUI. The implementation of GUI will be introduced following.

3-7-2 Flash Programmer and Zipped File System

Due to lack of hard drive in this system, the flash memory is used as the role of hard drive. Actually, all elements of the graphic interface frameworks are stored in this system “hard drive”, which is read when every time necessary.

Altera provides a read-only zip file system for use with the hardware abstraction layer system library. The read-only zip file system provides access to a simple file system stored in flash memory. One can access the zip file subsystem using the ANSI C standard library I/O functions, such as fopen() and fread().The zip file must be uncompressed. The read-only zip file system uses the zip format only for bundling files together; it does not provide any file decompression features.

The flash programmer is part of the SOPC development tools, and is a convenient say to program this memory. The purpose of the flash programmer is to program data into a flash memory device connected to an Altera FPGA. It sends file contents over a download cable, such as the USB Blaster, to a SOPC system running on the FPGA.

3-7-3 GUI Implementation

Actually, a 320×640 32-bit frame buffer is maintained by main procedure in form of an integer array, which keeps the RGB value of the 320×640 display area in any instance. Any time, if one wants to renew the display area, the only thing needs to do is a consecutive 320×640 writes of this frame buffer to SDRAM through the aid of

VGA/SDRAM controller dirver. VGA controller is internally connected to SDRAM and is always continuously reading same section of SDRAM as its outputs to display area.

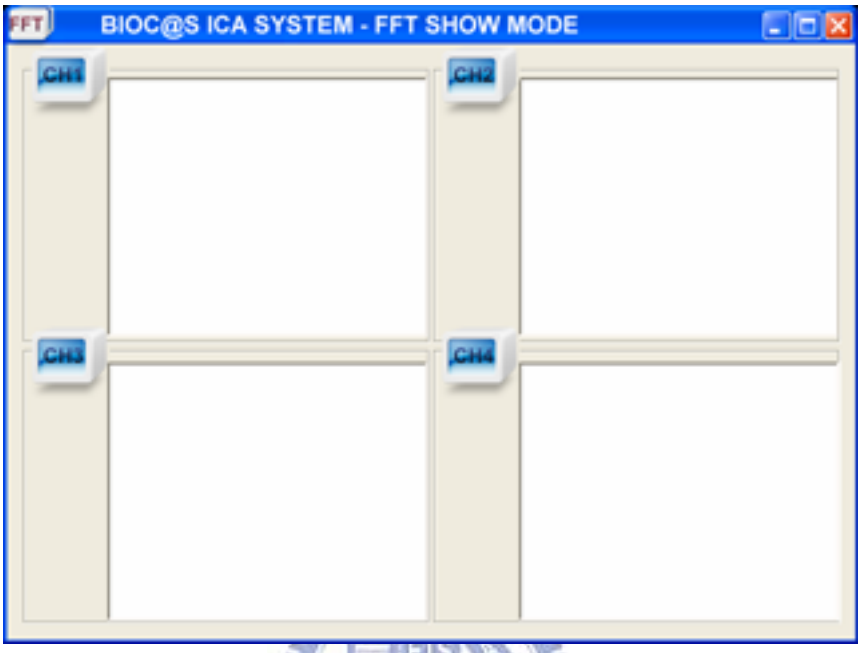


Fig. 3-28 Four channel mode GUI.

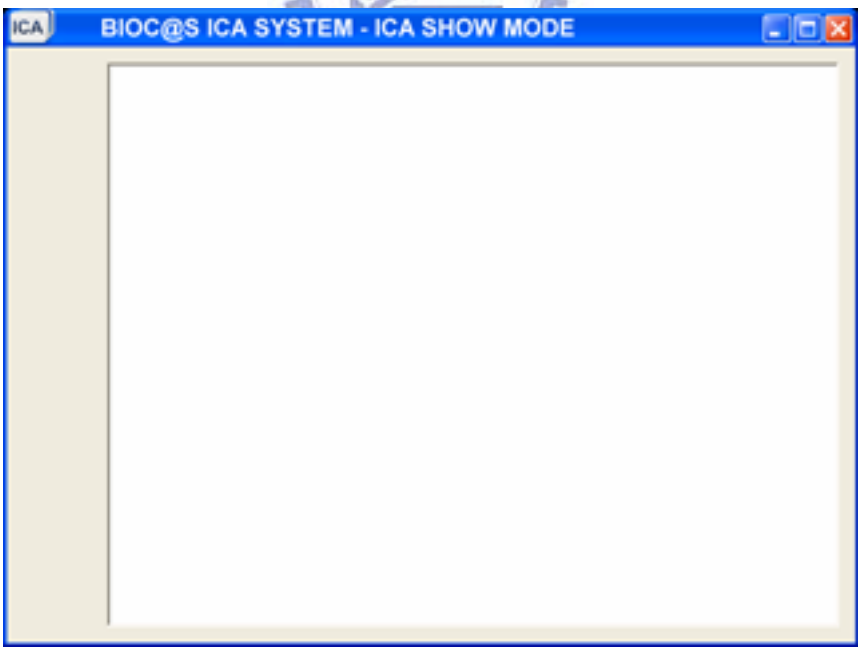


Fig. 3-29 One channel mode GUI.

Basically, the GUI in this work has tow modes, one is four channel mode and the other is one channel mode which are shown in Fig. 3-28 and Fig. 3-29. Both these

mode can show ICA result or FFT result which depends on the mode selection switch on development board. In one channel mode, extra channel selection switch can used to select specific channel to display.

As you can see, the blank area in GUI is the place showing moving waves. The first step of showing a wave in GUI is to scale the value of ICA or FFT result (which is divided by the height of blank area) to the “altitude” of blank area. Next, if a new value arrives, the right most column pixels in blank area must be discarded. And new arrival value should be shown in the left most column. Usually, two consecutive pixels will have a random value difference between them, that is, there may be a gap between the altitudes of two consecutive pixels in the blank area. Therefore, in order to reduce the roughness of two consecutive pixels in GUI, the connection of tow pixel is necessary. The connection means we fill up all pixels between the gap of two consecutive pixels (pixels in neighbor columns).

Chapter4
Experimental Results

In this section, we will show the real-time experimental result in GUI and comparison with other system.

4-1 Synthesis Result of the System

Before showing the experimental result, we check the synthesis result of whole system first.

Top-level Entity Name	DE2_NIOS
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	29,640 / 33,216 (89 %)
Total combinational functions	27,767 / 33,216 (84 %)
Dedicated logic registers	11,461 / 33,216 (35 %)
Total registers	11510
Total pins	348 / 475 (73 %)
Total virtual pins	0
Total memory bits	233,344 / 483,840 (48 %)
Embedded Multiplier 9-bit elements	70 / 70 (100 %)
Total PLLs	2 / 4 (50 %)

Fig. 4-1 Synthesis report of the system.

Type	Slack	Required Time	Actual Time
Worst-case tsu	N/A	None	10.457 ns
Worst-case tco	N/A	None	16.373 ns
Worst-case tpd	N/A	None	2.612 ns
Worst-case th	N/A	None	2.581 ns
Clock Setup: 'SDRAM_CPU_VGA_PLL:PLL1|altpll:altpll_component|_clk1'	0.964 ns	100.00 MHz (period = 10.000 ns)	110.67 MHz (period = 9.036 ns)
Clock Setup: 'CLOCK_50'	4.329 ns	50.00 MHz (period = 20.000 ns)	63.81 MHz (period = 15.671 ns)
Clock Setup: 'VGA_PLL:PLL2|altpll:altpll_component|_clk1'	33.697 ns	25.31 MHz (period = 39.506 ns)	172.15 MHz (period = 5.809 ns)
Clock Setup: 'VGA_PLL:PLL2|altpll:altpll_component|_clk0'	48.884 ns	18.41 MHz (period = 54.320 ns)	183.96 MHz (period = 5.436 ns)
Clock Setup: 'altera_internal_jtag~TCKUTAP'	N/A	None	126.49 MHz (period = 7.906 ns)
Clock Hold: 'SDRAM_CPU_VGA_PLL:PLL1|altpll:altpll_component|_clk1'	0.391 ns	100.00 MHz (period = 10.000 ns)	N/A
Clock Hold: 'VGA_PLL:PLL2|altpll:altpll_component|_clk0'	0.391 ns	18.41 MHz (period = 54.320 ns)	N/A
Clock Hold: 'VGA_PLL:PLL2|altpll:altpll_component|_clk1'	0.391 ns	25.31 MHz (period = 39.506 ns)	N/A
Clock Hold: 'CLOCK_50'	0.391 ns	50.00 MHz (period = 20.000 ns)	N/A
Total number of failed paths			

Fig. 4-2 Timing report of the system.

Synthesis report indicates that the system uses about 29,640 logic elements. And timing report shows that the system can work on a maximum frequency 110MHz.

4-2 Power Analysis of the System

The result of power estimation is depicted in Fig. 4-3 and Fig. 4-4. The power estimation is tested under room temperature, also we choose two constrains to reflect the case in real world. In Fig. 4-3 we choose a theoretical worst case which gives us a total power consumption 0.911W. A 0.865W total power consumption in Fig. 4-4 is given which is under a typical case constrain.

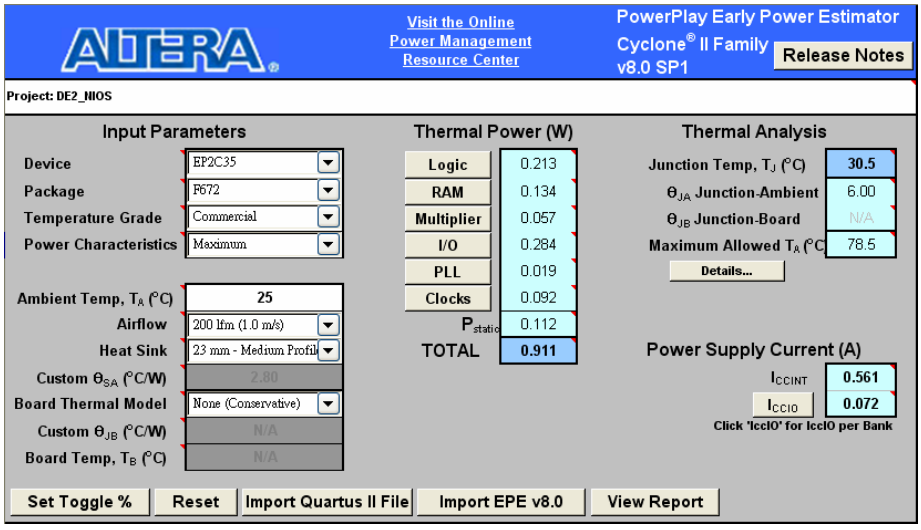


Fig. 4-3 System power report under worst case.

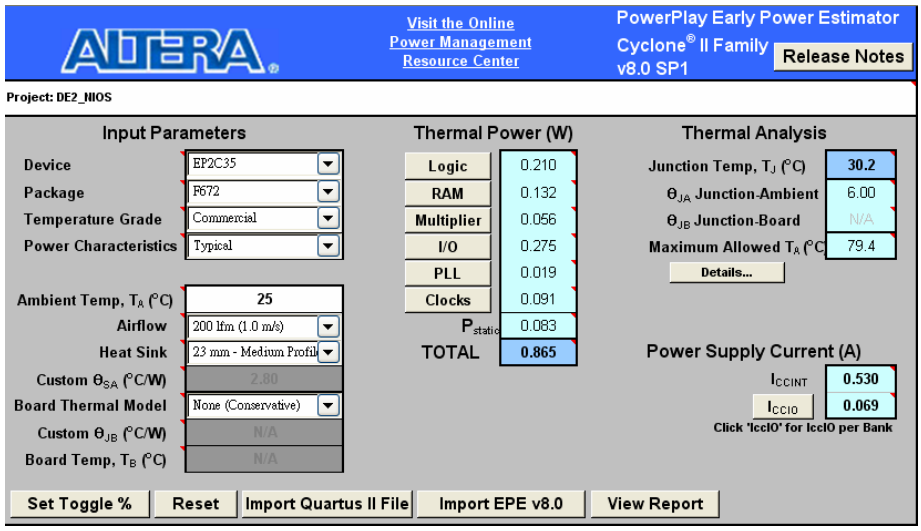


Fig. 4-4 System power report under typical case.

4-3 Device for Demonstration

The device for demonstration of the system is shown in Fig. 4-5. As it indicates, we program our design to FPGA through USB blaster, UART keep on receiving prerecorded EEG signal form PC to simulate the realistic case, VGA keep on outputting frame information to the display device to show the GUI for the user.

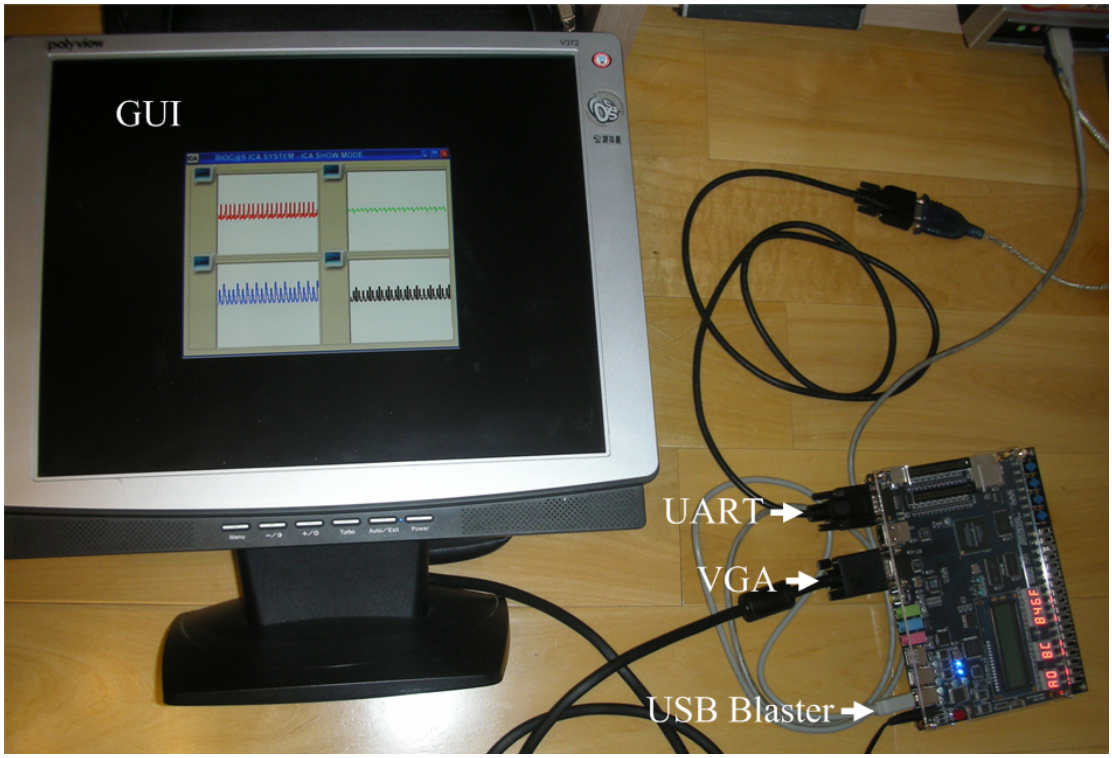


Fig. 4-5 Demonstration prototype of the system.

4-4 Experimental Result of Super-Gaussian Signal

Figure 4-8 through Fig. 4-17 show the experimental result of this work applied on Super-Gaussian signals. Figure 4-6 is the test pattern we transferred to our biomedical system which are mixed with a random proportion of four super-Gaussian signals which are shown in Fig. 4-7. In this case, we can see the four channel BSS signals is separated by ICA unit and they are shown respectively on different channels. Figure 4-8 shows the result of four-channel ICA result which is displayed

on a four channel monitoring mode. Figure 4-9 through Fig. 4-12 show the ICA result but in a single channel monitoring mode. Further, after doing frequency spectrum analysis by FFT unit, we can see there frequency compositions are shown on FFT showing window respectively. Figure 4-13 shows the result of four-channel FFT result which is displayed on a four channel spectrum analysis mode. In four channels mode, we only show the spectrum from 1Hz to 32Hz, each bar in our GUI represents one Hz. Figure 4-14 through Fig. 4-17 show the spectrum analysis result but in a single channel monitoring mode respectively. In one channel mode, 64 bars represent all computation result of 64-point FFT, usually we take only first 32 data since the rest of them are symmetric.

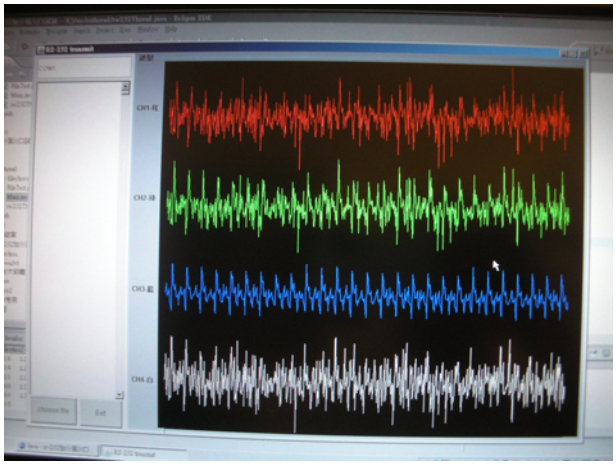


Fig. 4-6　Mixed signals of Super-Gaussian test pattern.

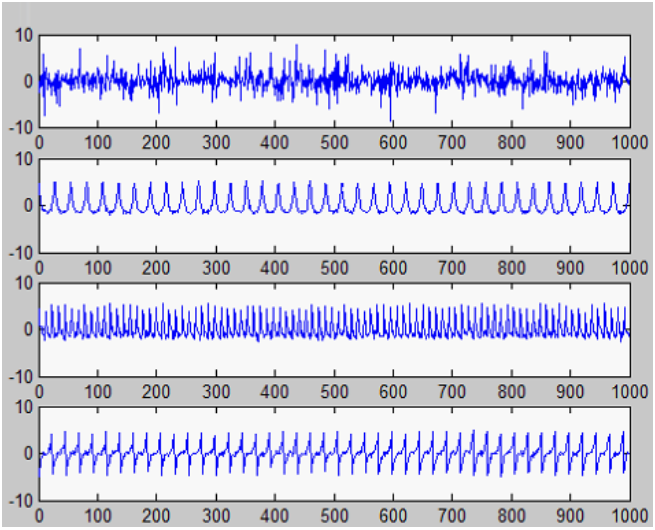


Fig. 4-7　Super-Gaussian mixture source.

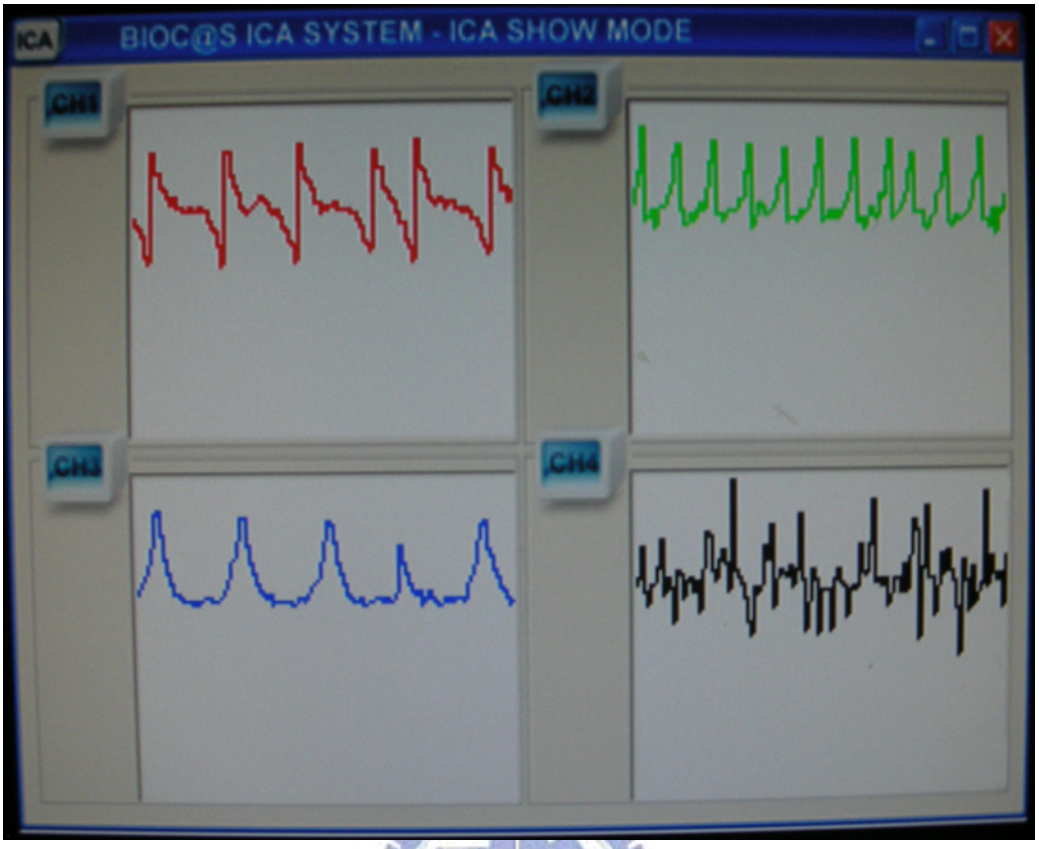


Fig. 4-8 Four channel GUI with ICA result of EEG signal.

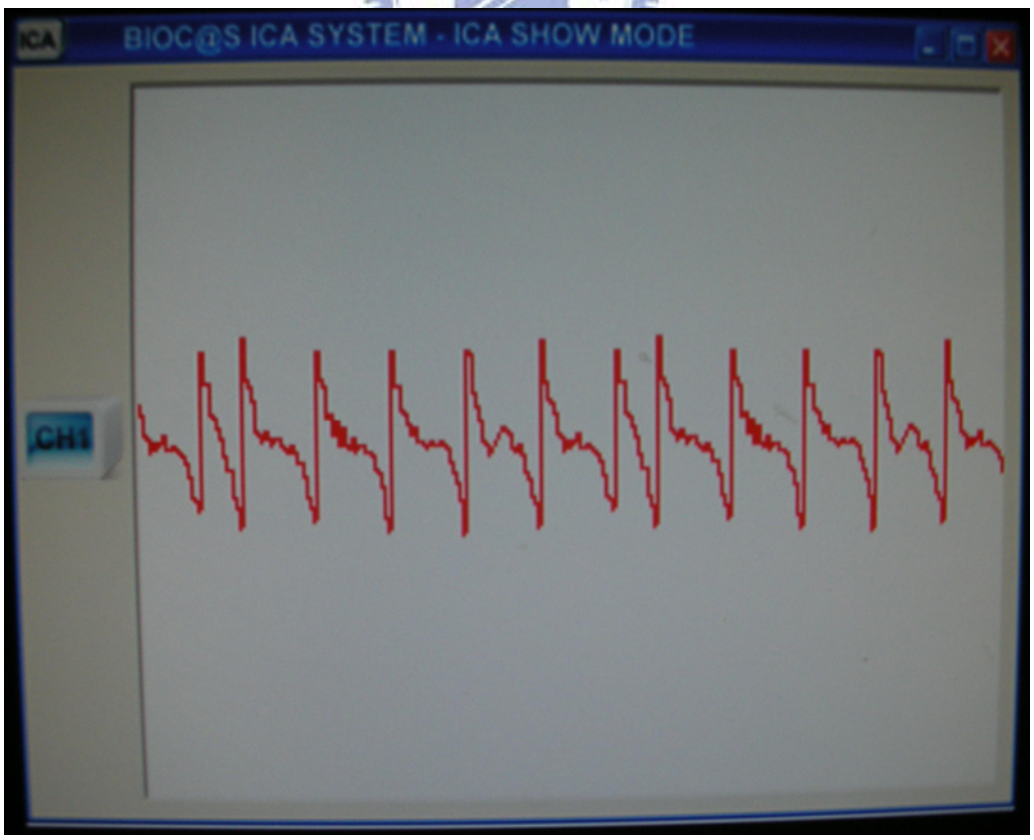


Fig. 4-9 One channel GUI with ICA result of EEG signal channel one.

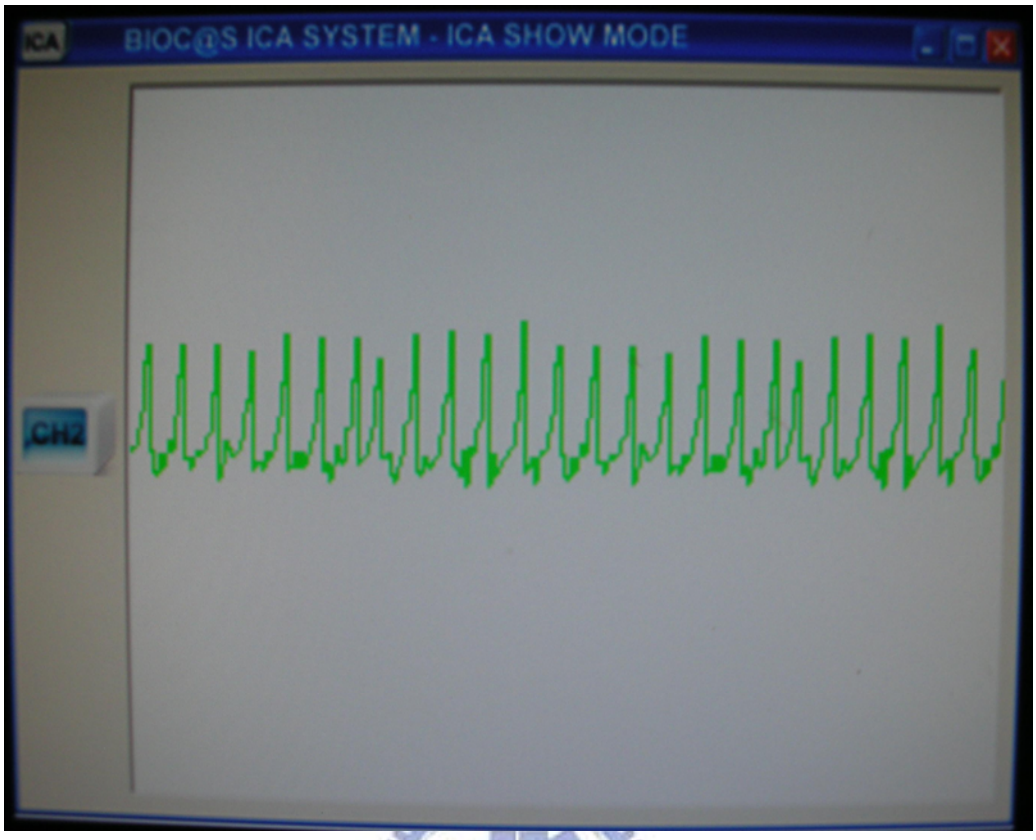


Fig. 4-10 One channel GUI with ICA result of EEG signal channel two.

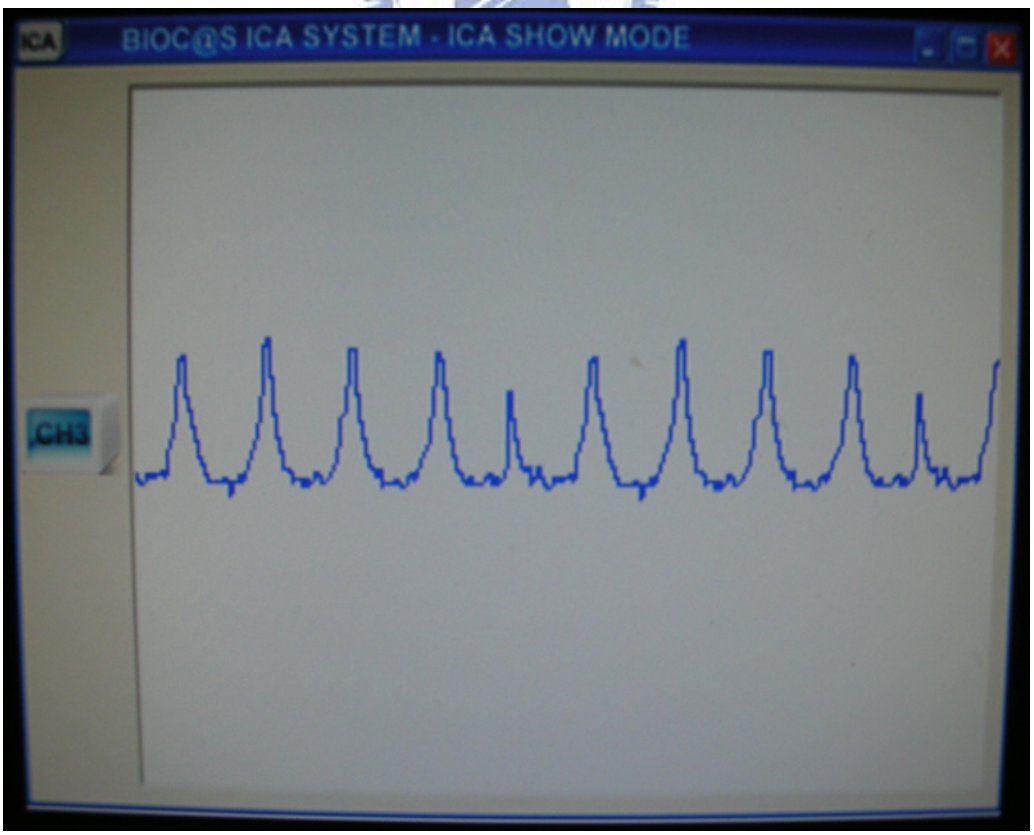


Fig. 4-11 One channel GUI with ICA result of EEG signal channel three.

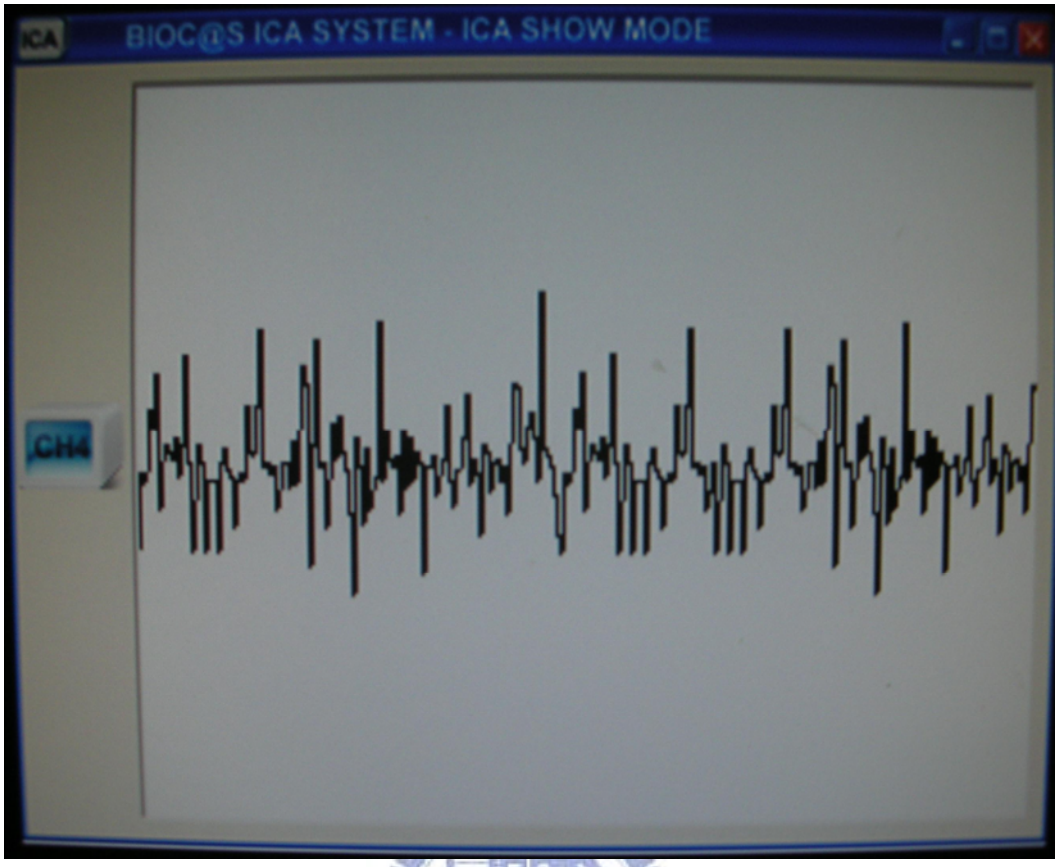


Fig. 4-12 One channel GUI with ICA result of EEG signal channel four.

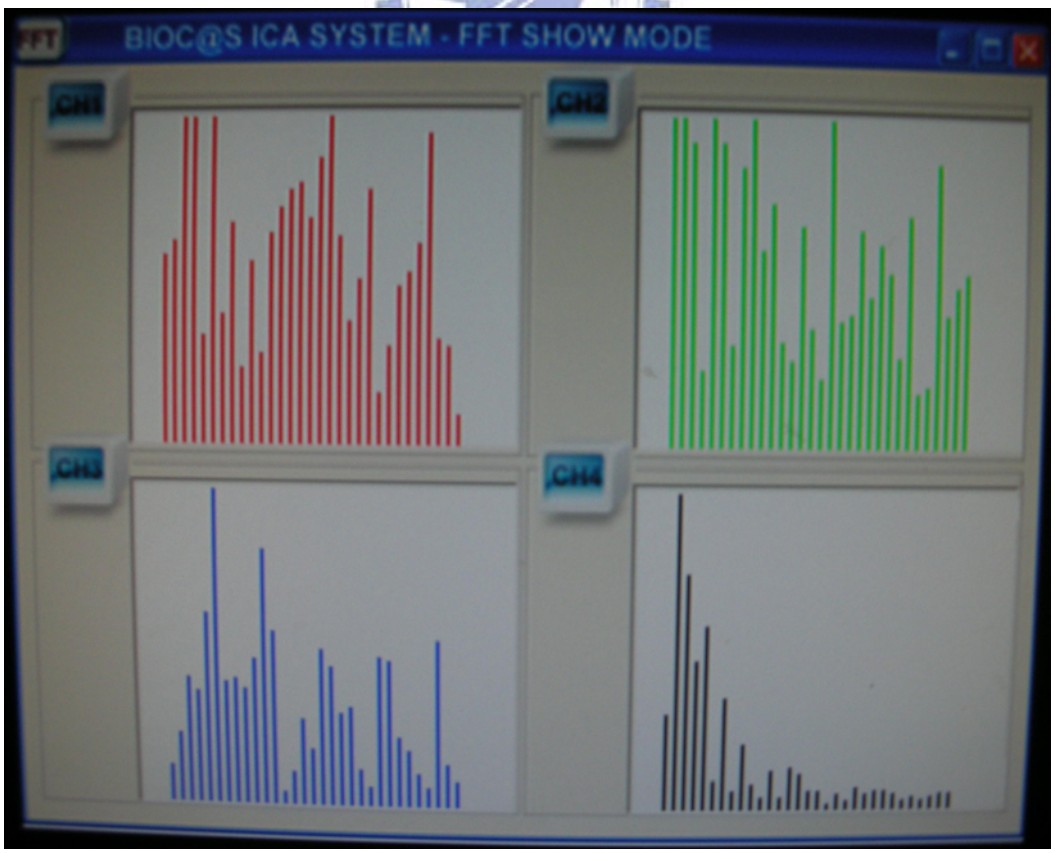


Fig. 4-13 Four channel GUI with ICA result of FFT signal.

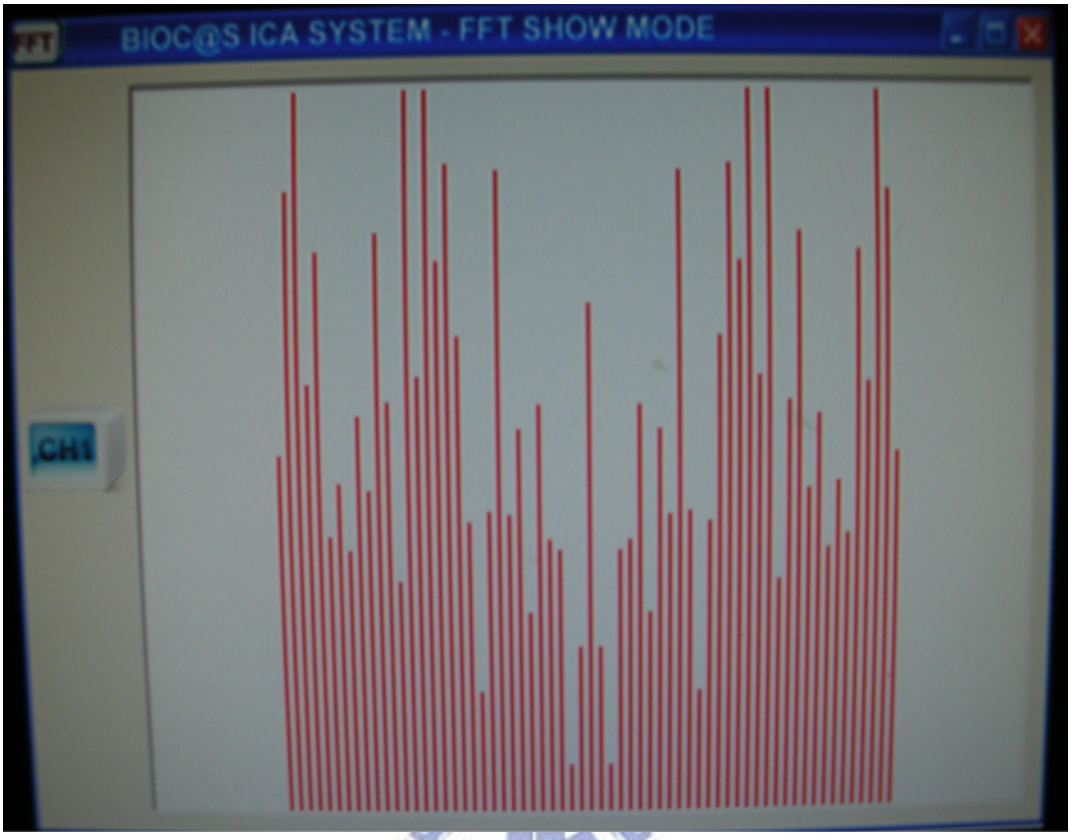


Fig. 4-14 One channel GUI with FFT result of EEG signal channel one.

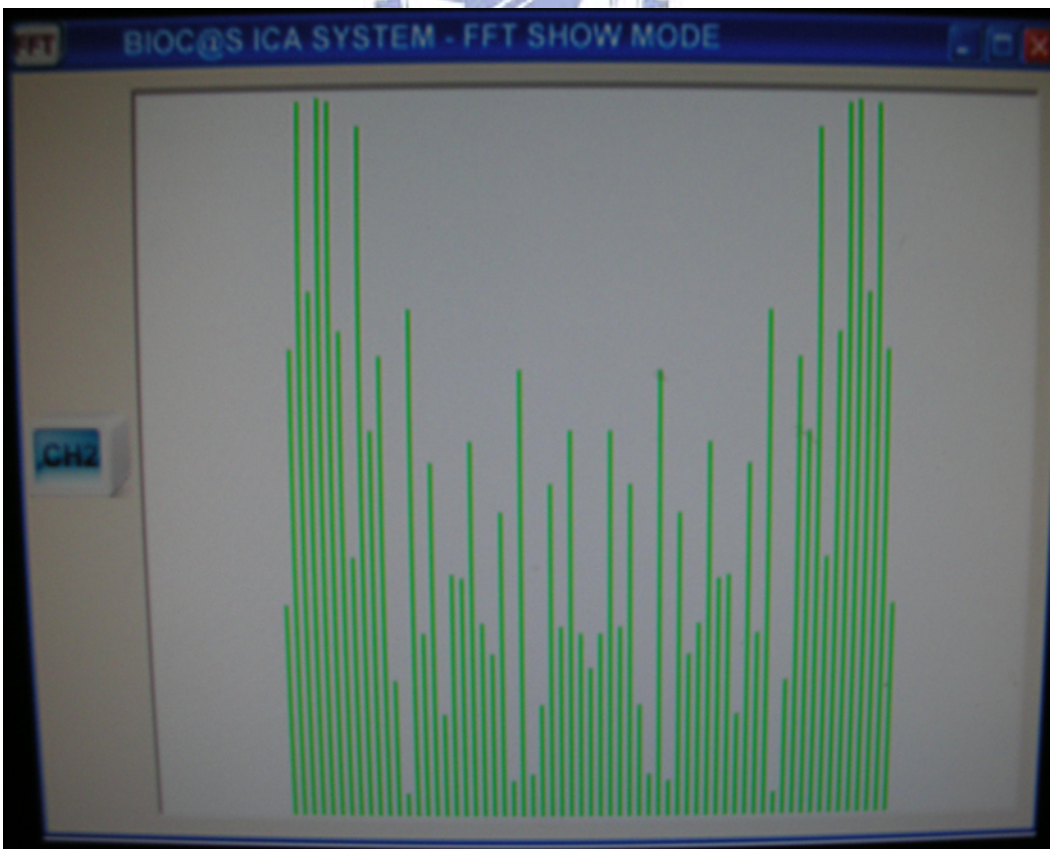


Fig. 4-15 One channel GUI with FFT result of EEG signal channel two.

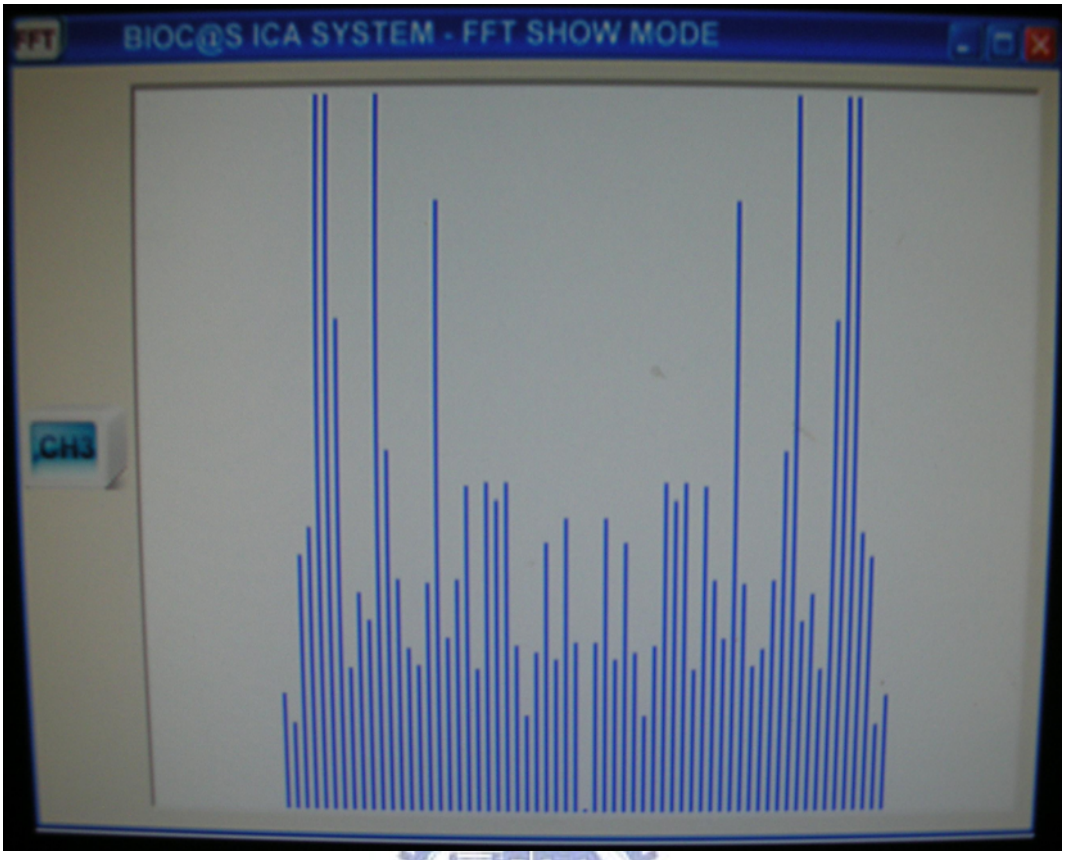


Fig. 4-16 One channel GUI with FFT result of EEG signal channel two.

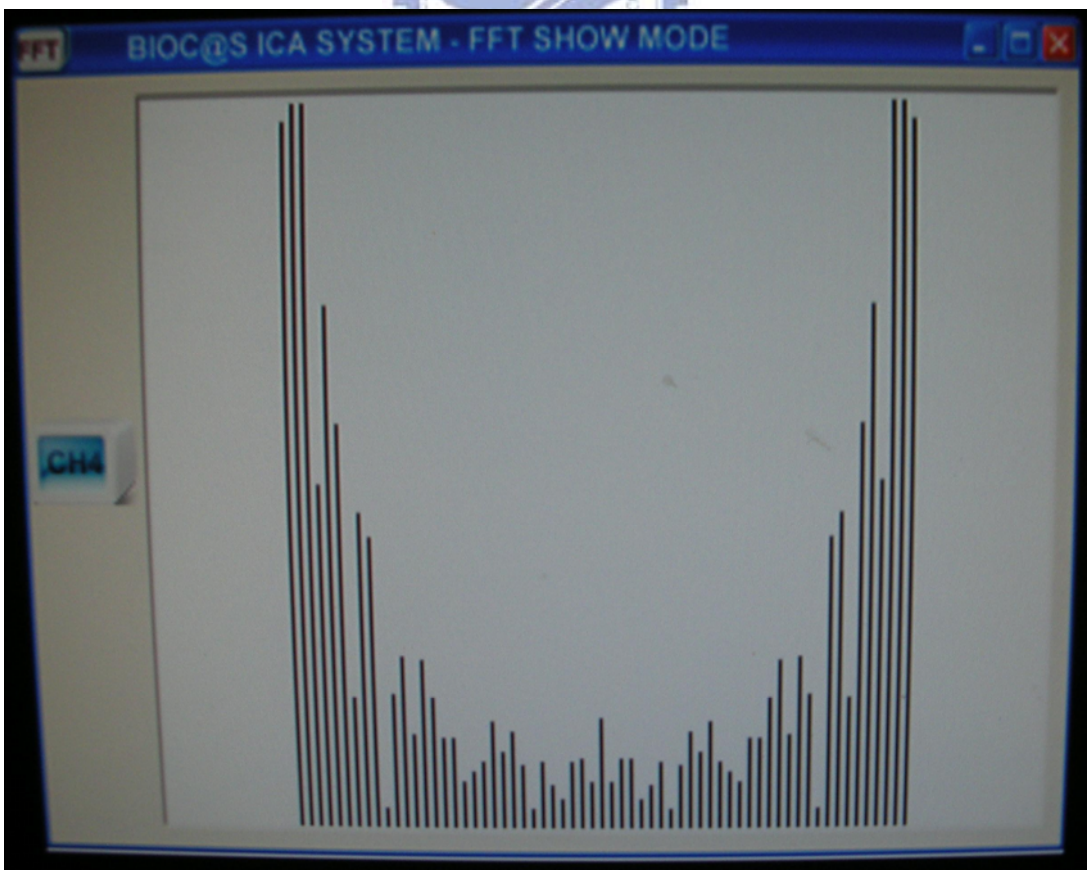


Fig. 4-17 One channel GUI with FFT result of EEG signal channel two.

4-5 Comparison with other System

There have been few studies about the design or implementation of biomedical system applied in EEG signal. The difference between our proposed and others is listed in Table 4-1. In 2008, C.T Lin, Y. C Chen, T. Y Huang, and Hung-Yi Hsieh [27] implemented a novel brain–computer interface (BCI) system that can acquire and analyze EEG signals in real-time to monitor human physiological as well as cognitive states. In 2006, J. R Chang Chien and C. C Tai [28] proposed a design which adopts a Nintendo palm-type Game Boy Advanced (GBA) color video player and incorporates self-made ECG measurement circuits to develop a portable color ECG measurement system. In 2008, Chia-Ta Lai [26] published a wireless embedded biomedical system which is mainly applied in EEG signal analysis. In Lai's study, a software version ICA together with FFT was implemented on the BF533, which is the processing core of the system. In comparison with Lai's study, software version ICA can do at most five stage training, however, a customized ICA hardware in this work can reach 128 stage trainings. Although our FFT is a 64-point version, higher resolution FFT can be achieved by configuring FFT output to next stage input in CPU. Moreover, since we have some DSP units such as ICA and FFT our system does not need to work on a high clock frequency (such as 600MHz), instead, the system operates on 100MHz. Consequently, this work presents a biomedical SoC with competitiveness in area and the power consumption of this work is relatively the least among listed platforms.

Table 4-1 Comparison with other ICA design

Platform	This work	BF533 WP1.[26]	BF533 Stamp	OMAP1510 OSK1.[27]	GBA SP1.[28]
Chipset	NIOS II	ADI BF533	ADI BF533	TI OMAP1510	ARM + Z80
Speed (CLK)	100MHz	600MHz	600MHz	ARM 168MHz DSP 192MHz	16.78MHz
Operating Voltage	3.3V	1.2V	1.2V	1.1V	-
SDRAM	8MB	16MB	128MB	32MB	384KB
Design	SoC	Embedded system	Embedded system	Embedded system	Embedded system
Power	0.911W	0.99W	1.584W	1.25W	-

Chapter5
Conclusions and Future Works

5-1 Conclusions

In this thesis, we had implemented a MCU/DSP biomedical system with 29,640 logic elements under about 110MHz operating frequency. We use NIOS II as its micro controller, also, two customized DSP units (ICA and FFT) are integrated into system. We implement a hardware ICA which operates under 50MHz but can reach 128 trainings which outperforms software version ICA 56 times. Also, we design hardware 64-points FFT with the relative RMSE no more than 1% and slightly performance improvement. Furthermore, implementation of SDRAM controller which provides the guarantee of correct read/write operations. The design of VGA controller provides correct timing of 640×480×60 display. The integration of these hardware make them successfully accessed by NIOS II CPU through system bus. A GUI application is executed on the system with EEG signal as testing pattern is verified on the system, showing the integration of software and hardware is correct and success. In comparison with the implementations of other related researches, this work achieves relative smaller power consumption about 0.91W. The SOC design manner also provides smaller area in comparison with embedded manner.

5-2 Future Works

In the future, the design can be developed for following parts:

- Replacing UART data transfer media with wireless interface.

- Another set of data transfer media can be added as feedback or warning to user or surveillant.
- Improving the precision of hardware ICA and FFT, using floating representation or more bits for fix-point representation.
- Porting operating system, implementing multithread process to provide higher concurrency of application.
- Changing display as LCM to increase portability.
- Integrating SD card as writable storage to record long term user physical status data.
- Taping out the SOC design and integrating this work and ICs mentioned in this thesis into a PCB.

References

[1] K. Ashwin Whitchurch, B. Han Ashok, R. Vinod Kumaar and K. Sarukesi, and K. Vijay Varadan, “Wireless system for long term EEG monitoring of Absence Epilepsy,” *Biomedical Applications of Micro- and Nanoengineering, Proceedings of SPIE*, Vol. 4937, pp. 343-349, Nov. 2002.

[2] P. Comon, “Independent component analysis, a new concept?’’ *Signal Process*., vol.36, no. 3, pp. 287–314, Apr. 1994.

[3] T-W Lee, "Independent Component Analysis - Theory and Applications", Kluwer Academic Publishers, 1998.

[4] C. M. Kim and S. Y. Lee, “A digital chip for robust speech recognition in noisy environment,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 1089–1092, 2001.

[5] T. P. Jung, S. Makeig, T. W. Lee, M. J. Mckeown, G. Brown, A. J. Bell, and T. J. Dejnowski, “Independent Component Analysis of Biomedical Signals,” in *Proceedings of the 2nd International Workshop on Independent Component Analysis and Blind Signal Separation*, pp. 633-44, 2000.

[6] N.J. Hill, T.N. Lal, K. Bierig, N. Birbaumer, and B. Scholkopf, “ Attentional modulation of auditory event-related potentials in a brain-computer interface,’’ in *Proc. IEEE Int. Workshop Biomedical Circuits and Systems, Singapore*, pp.17–20, *2004*.

[7] B. Kamousi, Z. Liu, and B. He, “Classification of Motor Imagery Tasks for Brain-Computer Interface Applications by Means of Two Equivalent Dipoles Analysis,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 13, No. 2, pp. 166-171, Jun. 2005.

[8] C. J. James and S. Wang, “Blind source separation in single-channel EEG analysis: An application to BCI,” in *Proc. 28th Annu. Int.Conf. IEEE Engineering in Medicine and Biology Society*, New York, USA, pp. 6544-6547, Aug. 2006.

[9] Cooley J W & Tukey J W, “An algorithm for the machine calculation of complex Fourier series,” *Math Comput*, Vol.19, pp.297-301, 1965.

[10] ISSI IS61LV25626 Data Sheet, Intergrated Silicon Solution, Inc. http://www.chipdocs.com/datasheets/datasheet-pdf/ISSI/IS61LV256-10.html

[11] A2V64S40CTP SDRAM Specification, Powerchip Semiconductor, Corp. http://code.google.com/p/minimig/downloads/detail?name=A2V64S40CTP.pdf

[12] S29AL032D Data Sheet, SPANSION Inc. http://www.spansion.com/datasheets/s29al032d_00_a8_e.pdf

[13] ADV7123 Data Sheet, Analog Devices Inc. http://www.ic37.com/htm_pdf/550634_915000-pdf.htm

[14] MAX232 Data Sheet, Texas Instruments I Inc. http://rocky.digikey.com/WebLib/Texas%20Instruments/Web%20data/MAX232,232I.pdf

[15] Saruwatari, H., Kawamura, T., Sawai, K.; Kaminuma, A., Sakata, M, “Blind source separation based on fast-convergence algorithm using ICA and beamforming for real convolutive mixture,” *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 1, pp. 13-17, 2002.

[16] Ristaniemi, T.and Joutsensalo, J. “Advanced ICA-based receivers for DS-CDMA systems,” *Personal, Indoor and Mobile Radio Communications*, Vol. 1, pp. 276 -281, 2000.

[17] C. Jutten and J. Herault, “Blind Separation of Sources 1. An Adaptive Algorithm Based on Neuromimetic Architecture,” *Signal Processing*, Vol. 24, pp 1-10, 1991.

[18] J.F. Cardoso, A. Souloumiac, “Blind Beamforming for Non-Gaussian Signals,” *IEE Proc.F,* Vol. 140, pp. 362-370, 1993.

[19] A. Hyvarinen and E. Oja. “A Fast Fixed-Point Algorithm for Independent Component Analysis,” *Neural Computation*, Vol. 9, pp. 1483-1492, 1997.

[20] Shousheng He and Mats Torkelson, “A New Approach to Pipeline FFT Processor,” *Proceedings of the* IPPS, pp.766 – 770, 1996.

[21] E.H. Wold and A.M. Despain, “Pipeline and parallel-pipeline FFT processors for VLSI implementation,” *IEEE Trans. Comput*, pp.414–426, May 1984.

[22] DE2 Development and Education Board User Manual, Version 1.31, ALTERA Corp.
http://www.ee.ryerson.ca/~courses/coe608/labs/DE2_UserManual.pdf

[23] Quartus II Version 7.2 Handbook Volume 4: SOPC Builder, ALTERA Corp.
http://www.altera.com/literature/hb/qts/qts_qii5v4.pdf

[24] Avalon Interface Specifications Version 1.0, ALTERA Corp.
http://www.ict.kth.se/courses/IL2206/0506/docs/mnl_avalon_spec.pdf

[25] Single- and Dual-Clock FIFO Megafunction User Guide Version 7.1, ALTERA Corp.
http://www.altera.com/literature/ug/ug_fifo.pdf

[26] 賴家達，“無線嵌入式生醫分析平台”，國立交通大學碩士論文，民國九十七年

[27] Chin-Teng Lin, Fellow, IEEE, Yu-Chieh Chen, Teng-Yi Huang, and Hung-Yi Hsieh, ” Development of Wireless Brain Computer Interface With Embedded Multitask Scheduling and it Application on Real-Time Driver’s Drowsiness Detection and Warning,” *Biomedical Engineering, IEEE Transactions,* , Vol.55, Issue 5, pp.1582 – 1591, May 2008

[28] Jia-Ren Chang Chien and Cheng-Chi Tai, ” The Design of a Portable ECG Measurement Instrument Based on a GBA Embedded System,” *Industrial Technology, 2006. ICIT 2006. IEEE International Conferenc*e, pp.1782 – 1787, Dec. 2006