

國立交通大學

機械工程研究所

碩士論文

以 USB 介面實現 AD/DA 訊號處理

**USB AD/DA Signal Processing**

研究生：鄭凱文

指導教授：成維華 博士

中華民國九十三年六月

以 USB 介面實現 AD/DA 訊號處理

USB AD/DA Signal Processing

研究生：鄭凱文

Student : Kevin Cheng

指導教授：成維華博士

Advisor : Dr. Wei-Hua Chieng

國立交通大學

機械工程學系

碩士論文

A Thesis

Submitted to Department of Mechanical Engineering

College of Electrical Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Mechanical Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

# 以 USB 介面實現 AD/DA 訊號處理

學生：鄭凱文

指導教授：成維華博士

國立交通大學機械工程研究所

## 摘要

本篇論文主要利用Cypress 公司生產的一顆USB Version1.1 的晶片---EZ-USB FX。因此晶片具有增強型的 8051 微處理器，其韌體可經由USB介面直接傳輸燒錄並可重新裝置列舉 (Renumeration<sup>TM</sup>)，因此增加了發展時的便利性。本論文目的在利用目前USB的普及性及HID(Human Interface Device)使用上的便利，以HID裝置設計一個不需使用者提供驅動程式及不需外接電源的方便性來實現類比/數位訊號擷取和測試類比/數位的擷取訊號於USB 1.1 介面中的效能。本論文的架構分為硬體，韌體，軟體三大部分，硬體即使用EZ-USB，韌體使用Keil C 編譯器來撰寫，軟體部分則使用Microsoft Visual C++ 6.0 來開發視窗介面的程式。

**關鍵字：USB，HID，通用序列匯流排，類比數位訊號處理**

# USB AD/DA Signal Processing

Student: Kevin Cheng

Advisor: Dr. Wei – Hua Chieng

**Institute of Mechanical Engineering  
National Chiao Tung University**

## Abstract

The thesis is chiefly on utilizing the chip USB Version 1.1, EZ-USB FX, which is manufactured by Cypress Semiconductor Corporation. This chip possesses an enhanced 8051 microprocessor. Its firmware can directly download by USB and use Renumeration™ technology to enumerate automatically. It has improved the convenience on development. The purpose of the thesis is to capitalize on the popularization of USB and the utility of HID (Human Interface Device), to capture and process the Analog-to-digital signal and Digital-to-analog signal by means of HID (Human Interface Device) within the utilization of USB 1.1 interface. The device has supported users to capture signals without the help of drivers and the necessity of electricity. The structure of the thesis is divided into 3 parts: Hardware, firmware, and software. EZ-USB is put to use as the hardware, the firmware is edited by Keil C Compiler and the software is primarily compiled by Microsoft Visual C++ 6.0 to develop the program of windows interface.

**Keyword: USB, HID, Analog-Digital signal.**

## 誌謝

生我者父母，育我者交大。於此優秀學府七百餘日，時日非長，唯同窗好友秉霖，彬佐，瑗焄，朝群等朝夕相處，終日為伍，教學相長，育樂相邀，感情甚深。實驗室乃地靈人傑匯居之地，各方學長齊聚一堂：時龍，淳彬，永成，嘉豐，晉偉，旭生等。於此做學，如醍醐灌頂，收事半功倍之效。心中崇敬之意，尤以成教授維華先生為鉅，弟子蒙先生指導，循循善誘，如沐春風。感謝之意，無引長江之水不能宣，無登台北一零一不能示。

除此，尚有多年莫逆，相互砥礪支持，感謝之意，言語難表。

行文至此，尚懷羞愧之心，跪於父母親大人膝下，感謝家父家母供於無慮之生活，溫暖之親情，使敝人責無旁貸，專於拙作。

# Contents

摘要.....	I
<b>ABSTRACT .....</b>	<b>II</b>
誌謝.....	III
<b>CONTENTS.....</b>	<b>IV</b>
<b>TABLE CONTENTS.....</b>	<b>VI</b>
<b>FIGURE CONTENTS.....</b>	<b>VII</b>
<b>CHAPTER 1 INTRODUCTION.....</b>	<b>1</b>
1.1 USB BACKGROUND .....	1
1.2 USB 1.1.....	4
1.2.1 Cable .....	4
1.2.2 Electrical.....	8
1.3 USB 2.0.....	10
1.4 USB COMMUNICATIONS.....	13
1.5 USB TRANSFERS AND PACKETS .....	16
<b>CHAPTER 2 USB CHIP---EZ-USB .....</b>	<b>19</b>
2.1 INTRODUCTION .....	19
2.2 FEATURES .....	20
<b>CHAPTER 3 HARDWARE DEVELOPMENT .....</b>	<b>24</b>
3.1 EZ-USB .....	24
3.2 AD CONVERTER/ DA CONVERTER.....	25
<b>CHAPTER 4 FIRMWARE DEVELOPMENT .....</b>	<b>32</b>
4.1 FIRMWARE CONFIGURATION .....	32
4.1.1 USB Descriptor .....	34
4.1.2 Peripheral Circular Program .....	37
4.2 HID CLASS .....	38
<b>CHAPTER 5 HOST APPLICATION.....</b>	<b>41</b>
5.1 WINDOWS APPLICATION.....	42
5.2 HARDWARE DRIVER.....	50
<b>CHAPTER 6 CONCLUSION.....</b>	<b>52</b>
6.1 TRANSFER RATE CALCULATION .....	52
6.2 EXPERIMENT RESULT.....	53
6.3 CONCLUSION .....	59



# Table Contents

TABLE 1-1 CHIRP STATE .....	10
TABLE 1-2 SYNCHRONIZATION TYPES .....	14
TABLE 1-3 TRANSFER TYPES .....	15
TABLE 1-4 USB PID .....	17
TABLE 4-1 FILES THAT MAKING A EZ-USB FIRMWARE NEEDS .....	32
TABLE 5-1 USEFUL WINDOWS API FOR HID.....	50





# Figure Contents

FIGURE 1-1 STANDARD HIGH/FULL SPEED HARDWIRED CABLE ASSEMBLY .....	5
FIGURE 1-2 STANDARD LOW SPEED HARDWIRED CABLE ASSEMBLY .....	6
FIGURE 1-3 CABLE CONSTRUCTION OF HIGH/FULL SPEED .....	7
FIGURE 1-4 FULL-SPEED DEVICE CABLE AND RESISTOR CONNECTIONS .....	8
FIGURE 1-5 LOW-SPEED DEVICE CABLE AND RESISTOR CONNECTIONS .....	9
FIGURE 1-6 COMMUNICATION FLOW IN A USB SYSTEM .....	16
FIGURE 1-7 THE PACKET'S ELEMENT.....	18
FIGURE 2-1 CY7C64613 (128 PIN) SIMPLIFIED BLOCK DIAGRAM.....	20
FIGURE 2-2 FINAL SYSTEM DIAGRAM OF EZ-USB FX.....	22
FIGURE 2-3 ENUMERATION & RENUMERATION.....	23
FIGURE 3-1 EZ-USB 8051-BASED CPU .....	24
FIGURE 3-2 EZ-USB FX 128 PIN .....	25
FIGURE 3-3 MAIN SCHEMATIC .....	27
FIGURE 3-4 ANALOG TO DIGITAL INPUT SCHEMATIC .....	28
FIGURE 3-5 DIGITAL TO ANALOG OUTPUT .....	29
FIGURE 3-6 ADC0809 TIME DIAGRAM .....	30
FIGURE 3-7 PHOTO OF THE AD CONVERTER .....	30
FIGURE 3-8 PROTOTYPE OF THE HID AD/DA PROCESSING SYSTEM .....	31
FIGURE 4-1 FLOW CHART OF THE EZ-USB DRIVER FUNCTION .....	33
FIGURE 4-2 THE ORDER OF USB DESCRIPTORS.....	35
FIGURE 4-3 DESCRIPTORS INFORMATION .....	35
FIGURE 4-4 DESCRIPTORS IN USB.....	36
FIGURE 4-5 THE FLOWCHART OF THE SIGNAL BETWEEN HOST AND USB .....	38
FIGURE 4-6 REPORT DESCRIPTOR.....	40
FIGURE 5-1 THE FLOW CHART OF USB SOFTWARE DEVELOPMENT.....	41
FIGURE 5-2 PID AND VID .....	42
FIGURE 5-3 THE FLOW CHART OF HID DEVICE CHECK .....	47
FIGURE 5-4 THE FLOW CHART OF THE PROGRAM .....	48
FIGURE 5-5 SNAPSHOT OF THE PROGRAM.....	49
FIGURE 5-6 HID DEVICE SHOWS UP ON DEVICE MANAGER.....	51
FIGURE 6-1 100HZ SIGNAL.....	53
FIGURE 6-2 100HZ SIGNAL (2).....	54
FIGURE 6-3 1 KHZ SIGNAL.....	54
FIGURE 6-4 1 KHZ SIGNAL (2).....	55
FIGURE 6-5 2 KHZ SIGNAL.....	55
FIGURE 6-6 2 KHZ SIGNAL (2).....	56
FIGURE 6-7 3 KHZ SIGNAL.....	56
FIGURE 6-8 3 KHZ SIGNAL (2).....	57

FIGURE 6-9 DIGITAL-TO-ANALOG OUTPUT TEST (1)..... 58  
FIGURE 6-10 DIGITAL-TO-ANALOG OUTPUT TEST (2) ..... 58



# Chapter 1

## Introduction

### 1.1 USB Background

Universal Serial Bus (USB) was invented in 1995 by Universal Serial Bus Implementers Forum (USB-IF), which the group included computer manufacturers and peripherals vendors. A major purpose of USB is to replace most of traditional interface ports on personal computer with one user-friendly way. The original motivation for the Universal Serial Bus (USB) came from three interrelated considerations:

- **Connection of the PC to the telephone**

It is well understood that the merge of computing and communication will be the basis for the next generation of productive applications. The movement of machine-oriented and human-oriented data types from one location or environment to another depends on ubiquity and low-priced connectivity. Unfortunately, the computing and communication industries have evolved independently. The USB provides a ubiquitous link that can be used across a wide range of PC-to-telephone interconnection.

- **Ease-of-use**

The lack of flexibility in reconfiguring the PC has been acknowledged as the Achilles' heel to its further deployment. The combination of user-friendly graphical interfaces, the hardware and software mechanisms associating with new-generation bus architectures has made computers less confrontational and easier to reconfigure. However, from the end user's point of view, the PC's I/O interfaces, such as serial/parallel ports, keyboard/mouse/joystick interfaces, etc., do not have the attributes of plug-and-play.

- **Port expansion**

The addition of external peripherals continues to be constrained by port availability. The lack of a bidirectional, low-cost, low-to-mid speed peripheral bus has held back the creative proliferation of peripherals such as telephone/fax/modem adapters, answering machines, scanners, PDA's, keyboards, mice, etc. Existing interconnections are optimized for one or two point products. As each new function or capability is added to the PC, a new interface has been defined to address this need.

### **Goals for the Universal Serial Bus**

The USB is specified to be an industry-standard extension to the PC architecture with a focus on PC peripherals that enable consumer and business applications. The following criteria were applied in defining the

architecture for the USB:

- Ease-of-use for PC peripheral expansion
- Low-cost solution that supports transfer rates up to 480 Mb/s
- Full support for real-time data for voice, audio, and video
- Protocol flexibility for mixed-mode isochronous data transfers and asynchronous messaging
- Integration in commodity device technology
- Comprehension of various PC configurations and form factors
- Provision of a standard interface capable of quick diffusion into product
- Enabling new classes of devices that augment the PC's capability
- Full backward compatibility of USB 2.0 for devices built to previous versions of the specification.



The USB has three speeds which can be used from low-speed, full-speed to hi-speed. In USB 1.1, there are only the first two speeds, low-speed and full-speed. In USB2.0, the third speed, hi-speed, is added. Transmission rate of three speeds are: 1.5 Mb/s, 12 Mb/s and 480 Mb/s. The USB2.0 is full compatible with USB1.1.

## **1.2 USB 1.1**

The USB1.1 was invented in September 23, 1998. Its speeds support 1.5Mb/s and 12Mb/s. It is the most popular USB version that we used in the market. It includes many applications such as: mouse, scanner, printer, flash memory and the MP3 player etc. These devices do not need a fast speed rate, thus the USB1.1 can be used to implement the devices. The details of USB1.1 will be described below.

### **1.2.1 Cable**

The USB cable consists of four conductors, two power conductors, and two signal conductors, the recommended colors are white, grey, or black.

Figure1-1 illustrates a standard high-/full- speed hardwired cable assembly.

Figure 1-2 illustrates a standard low-speed hardwired cable assembly.

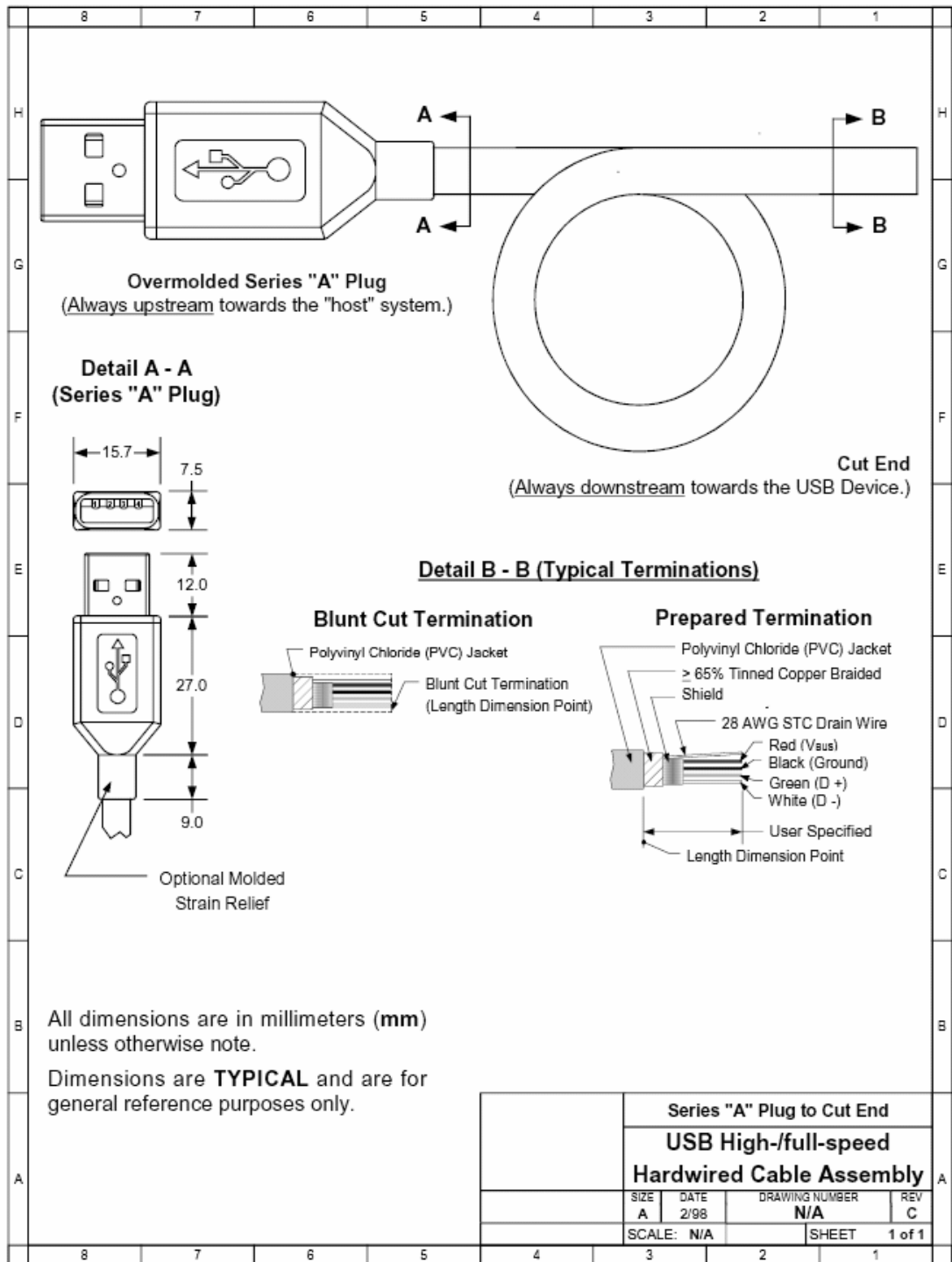


Figure 1-1 Standard high/full speed hardwired cable assembly

Reference: USB Specification V2.0

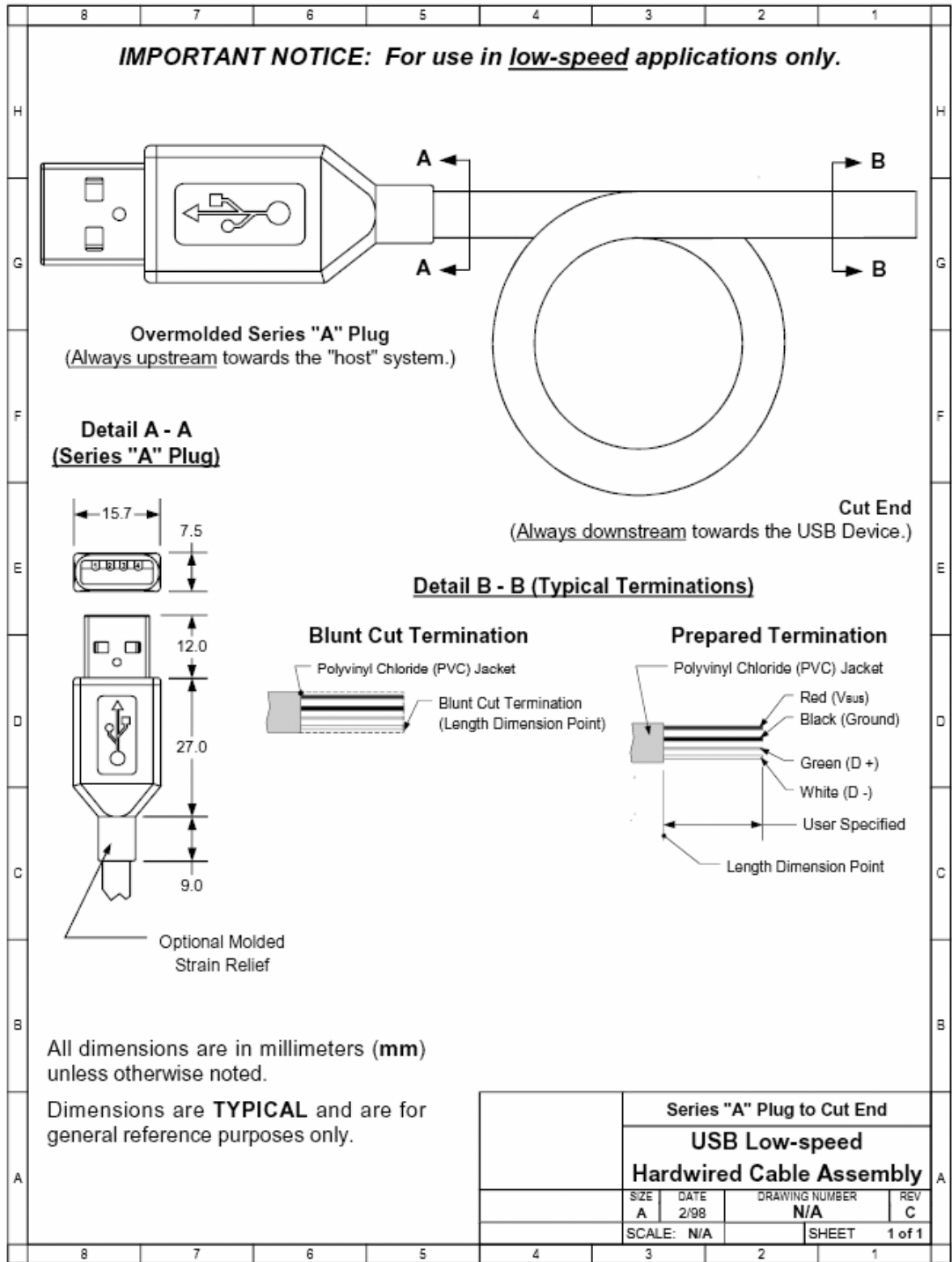


Figure 1-2 Standard low speed hardwired cable assembly

Reference: USB Specification V2.0



High-/full-speed cable consists of a signaling twisted pair, VBUS, GND, and an overall shield. High-/full- speed cable must be marked to indicate suitability for USB usage. High-/full-speed cable can be used with either low-speed, full-speed, or high-speed devices. Low-speed is recommended, but does not require using the cable with twisted signaling conductors. The maximum allowable cable length is determined by signal pair attenuation and propagation delay, usually the limited length is less than 5.0 meters for high-/full- speed and 3.0 meters for low-speed. Figure 1-3 shows the cable construction of high/full speed.

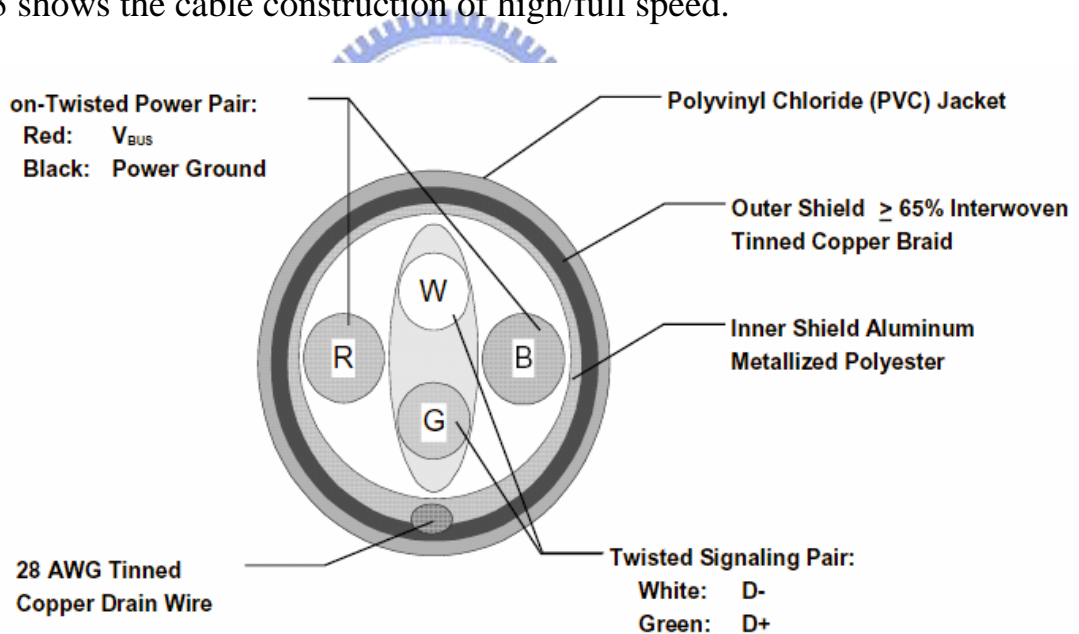


Figure 1-3 Cable construction of high/full speed

Reference: USB Specification V2.0

## 1.2.2 Electrical

This electrical utility of USB contains signaling, power distribution, and physical layer specifications. We will put the focus on the major difference between USB1.1 and USB2.0; other details will not be described so much here.

### Devices detection

The USB hub should monitor each port to check if there is connection or disconnection. Two pull-down resistors on the D+ and D- lines of the hub ensure that both data lines are ground. The USB device should include a pull-up resistor on either D+ or D- to trigger connect detection. Figure 1-4 and Figure 1-5 show the difference between full-speed device and low-speed device.

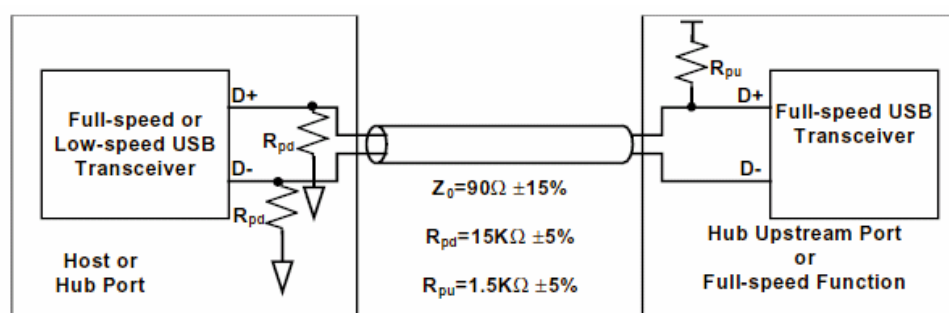


Figure 1-4 Full-speed device cable and resistor connections

Reference: USB Specification V2.0

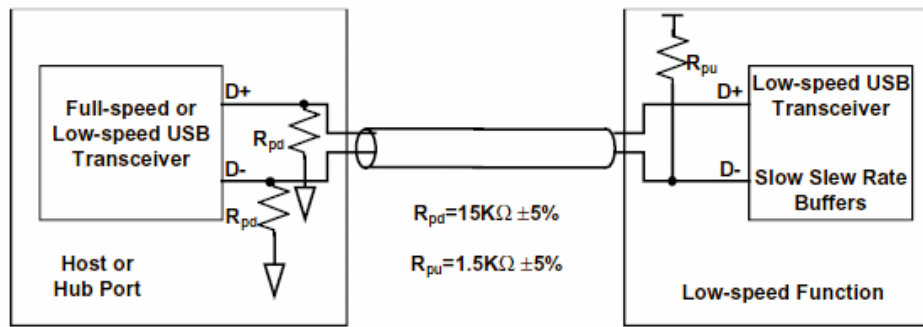


Figure 1-5 Low-speed device cable and resistor connections

Reference: USB Specification V2.0

## Power distribution

All USB ports provide power for devices which are attached to them. The peripheral devices can be designed for either using the power that the USB hub supplies or using their own power supply. The Cable power is 5 Volt dc and can be used to generate the devices. The self-powered hub can provide current up to a maximum of 500ma and at a minimum of 100ma for the devices. For example, when the external 4-ports hub is used without the AC adepter, each port can provide a maximum current of 100ma. However, when it is used with AC adepter, each port can provide a maximum current of 500ma. The power setup of the device is defined in the Configuration Descriptor.

## 1.3 USB 2.0

In April 2000, the new specification of USB was released, as known as USB 2.0. The transmission rate of USB 2.0 is 480Mbps, which is 40 times faster than USB 1.1. In the best case, while the bus is idling, a high-speed bulk transfer can move data at 53Mbps, using 90% of the bus' bandwidth. For the compatibility, USB 2.0 device should be enumerated as a full-speed device that can communicate to USB 1.1 hub. In other words, USB 2.0 hub supports three speeds and allows full-speed and low-speed while transferring the data.

### High-speed Signaling Levels

The high-speed signaling voltage specification had defined “Chirp K” state and “Chirp J” state. Table 1-1 shows these two states' definition.

Table 1-1 Chirp state

<b>Chirp J State</b> (differential voltage; applies only during reset <b>time</b> when both hub and device are high-speed capable)	<b>DC Levels:</b> $V_{CHIRPJ}(\min) \leq (D+-D-) \leq V_{CHIRPJ}(\max)$
<b>Chirp K State</b> (differential voltage; applies only during reset <b>time</b> when both hub and device are high-speed capable)	<b>DC Levels:</b> $V_{CHIRPK}(\min) \leq (D- - D+) \leq V_{CHIRPK}(\max)$

## Devices detection

The high-speed Reset and Detection mechanisms follow the behavioral model for low-/full-speed. When reset is completed, the link must **to** be operating in its appropriate signaling mode (low-speed, full-speed, or high-speed as governed by the preceding usage rules), and the speed indication bits in the port status register will report this mode correctly. High-speed capable devices initially attach as full-speed devices. After the initial attachment, high-speed capable transceivers engage in a low level protocol during reset to establish a high-speed link and to indicate high-speed operation in the appropriate port status register.

High-speed Detection Handshake is used to detect whether the device is at high-speed or not. The procedure is listed as below.

1. The high-speed device leaves the D+ pull-up resistor connected, leaves the high-speed terminations disabled, and drives the high-speed signaling current into the D- line. This creates a Chirp K on the bus. The device chirp must last no less than 1.0 ms (TUCH) and must end no more than 7.0 ms (TUCHEND) after high-speed Reset time T0.
2. The hub must detect the device chirp after the assertion of the Chirp K **is seen** for no less than 2.5  $\mu$ s.
3. No more than 100  $\mu$ s (TWTDC) after the bus leaves the Chirp K state,

the hub must begin to send an alternating sequence of Chirp K's and Chirp J's. There must be no idle status on the bus between the J's and K's. This sequence must continue to a time (TDCHSE0) no more than 500  $\mu\text{s}$  and no less than 100  $\mu\text{s}$  before the end of Reset. (This will guarantee that the bus remains active and prevent the device from entering the high-speed Suspend state.) Each individual Chirp K and Chirp J must last no less than 40  $\mu\text{s}$  and no more than 60  $\mu\text{s}$  (TDCHBIT).

4. After completing the hub chirp sequence, the hub asserts SE0 until the end of Reset. At the end of reset, the hub must switch to the high-speed Enabled state.
5. After the device completes its chirp, it looks for the high-speed hub chirp. At a minimum, the device is required to see the sequence Chirp K-J-K-J-K-J in order to detect a valid hub chirp. Each individual Chirp K and Chirp J must be detected for no less than 2.5  $\mu\text{s}$  (TFILT).
  - A) If the device detects the sequence Chirp K-J-K-J-K-J, then no more than 500  $\mu\text{s}$  (TWTHS) after detection **that** the device is required to disconnect the D+ pull-up resistor, enable the high-speed terminations, and enter the high-speed Default state.
  - B) If the device has not detected the sequence Chirp K-J-K-J-K-J by a

time no less than 1.0 ms and no more than 2.5 ms (TWTFS) after completing its own chirp, then the device is required to revert to the full-speed Default state and wait for the end of Reset.

## 1.4 USB Communications

USB supports four transmission types: control, interrupt, bulk, and isochronous.

Control transfer--- control transfer is used to transfer specific data of USB device. Control transfer is commonly used during device configuration.

Interrupt transfer--- interrupt transfer is used for devices that must be polled periodically to see if the device has data to transfer. Applications can be applied such as mice and keyboards.

Bulk transfer --- a bulk transfer is used for large blocks of data. There is no periodic or transfer rate required. Applications can be applied such as printer or scanner.

Isochronous transfer --- isochronous transfer is used to the transfer which requires a constant delivery rate. Applications can be applied such as microphone, speaker. There is a problem with isochronous transfers which is synchronization. In USB, we have feedback and feed forwarding

solution. Table 1-2 lists the synchronization types for both source and sink.

Table 1-3 contains more details for these four transfers. Figure 1-6 is the communication flow in a USB system

Table 1-2 Synchronization types

	Source	Sink
Asynchronous	Free running source clock Provides implicit feed forward. The data rate is carried implicitly in the data stream based on the number of samples it produces in a frame	Free running sink clock Provides explicit feed back via a synchronous pipe. The endpoint sends feedback to the host to indicate its data rate. This feedback info is relative to the frame (SOF) timing.
Synchronous	Source clock lock to USB clock Uses implicit feedback. The feedback is supplied via the SOF packet. The endpoint slaves its sample clock to the SOF via a PLL.	Sink clock lock to USB clock Uses implicit feedback. The feedback is supplied via the SOF packet. The endpoint slaves its sample clock to the SOF via a PLL.
Adaptive	Source clock lock to sink Uses explicit feedback via an isochronous pipe to determine the desired frequency of the sink. The feedback info is relative to the frame (SOF) timing.	Sink locked to data flow Uses implicit free forwarding. The data rate is carried implicitly in the data stream based on the number of samples it produces in a frame. The adaptive endpoint synchronizes its sample clock to the data stream rate.



Table 1-3 Transfer types

Type	Control	Interrupt	Isochronous	Bulk
Transfer rate of high-speed(Byte/1ms)	15872	24576	24576	53248
Transfer rate of full-speed(Byte/1ms)	832	64	1023	1216
Transfer rate of low-speed(Byte/1ms)	24	0.8 (8Byte/10ms)	Not allow	Not allow
Bandwidth	10%	90% in USB 1.1		No
	20%	80% in USB 2.0		
CRC Check	Yes	Yes	No	Yes
Guaranteed delivery time	No	No	Yes	No
Guaranteed delay time	No	Yes	No	No

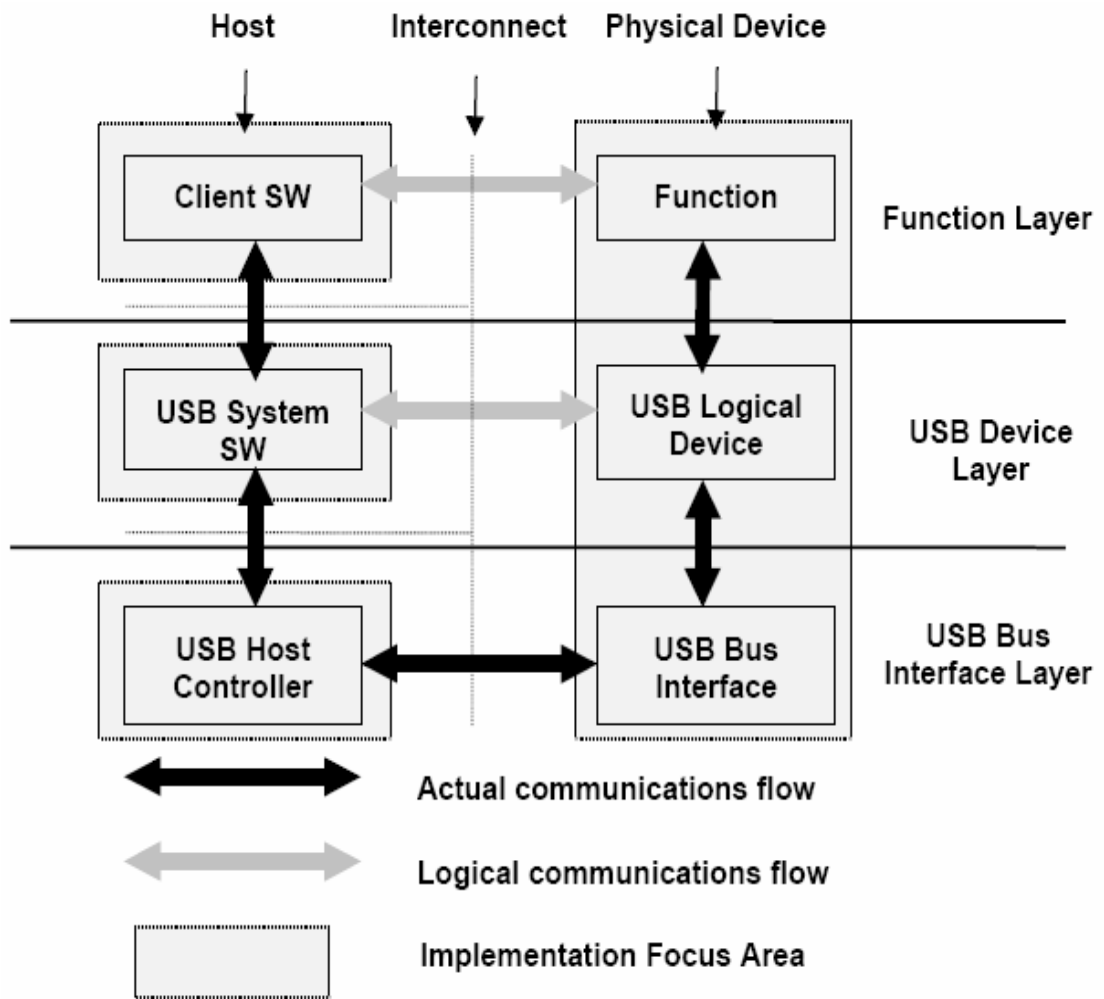


Figure 1-6 Communication flow in a USB system

## 1.5 USB Transfers and Packets

There are three stages in USB transfers. And the smallest unit is “Packet”; each packet contains information of transfer and data. The first byte of packet is always a Packet Identifier (PID), which defines the packet’s type. The packet identifier byte is formed with 4 bites and complement of these 4 bits. Table 1-4 shows some of packet identifiers (PID) and their types, and category. Others will be found in USB

document. Figure 1-7 will show the packet's element and the number below is the element's data (in bits).

Table 1-4 USB PID

<b>PID Value</b>	<b>Packet type</b>	<b>Packet category</b>
0101	SOF	token
1101	SETUP	token
1001	IN	token
0001	OUT	token
0011	DATA0	data
1011	DATA1	data
0010	ACK	handshake
1010	NAK	handshake
1110	STALL	handshake



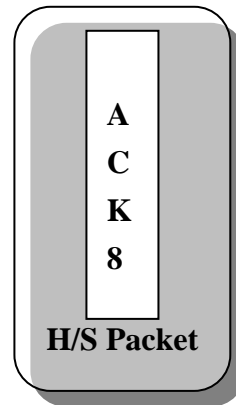
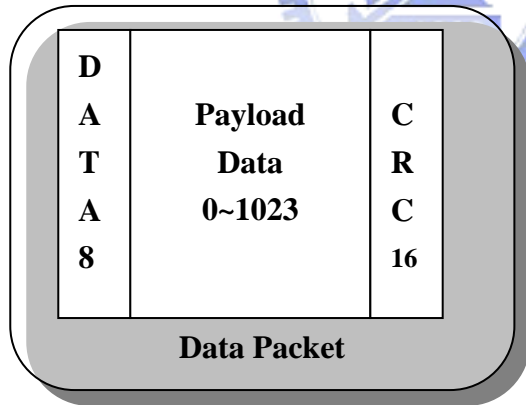
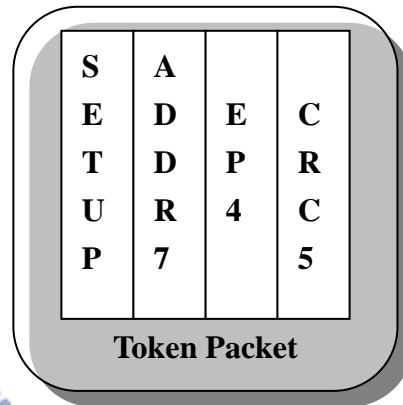
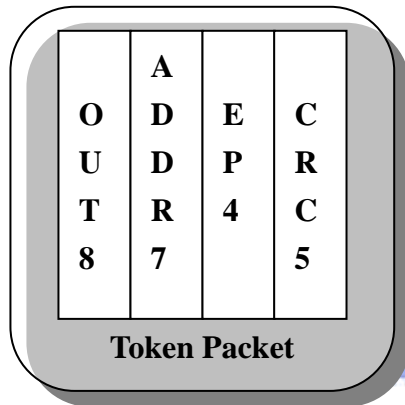
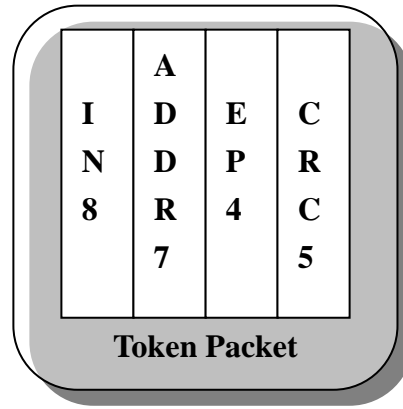
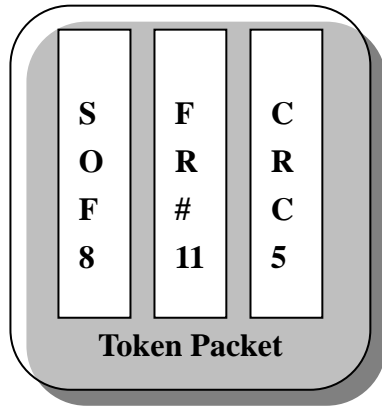


Figure 1-7 The packet's element

# Chapter 2

## USB Chip---EZ-USB

### 2.1 Introduction

The Cypress Semiconductor EZ-USB™ FX CY7C646xx is a compact integrated circuit that provides a highly integrated solution for a USB peripheral device. The key EZ-USB FX features are:

- The EZ-USB FX provides a “soft” (RAM-based) solution that allows unlimited configuration and upgrades.
- The EZ-USB FX delivers full USB throughput. Designs that use EZ-USB are not limited by the number of endpoints, buffer sizes, or transfer speeds.
- The EZ-USB FX does much of the USB housekeeping in the EZ-USB core, simplifying code and accelerating the USB learning curve. Figure 2-1 is the block diagram of EZ-USB FX.

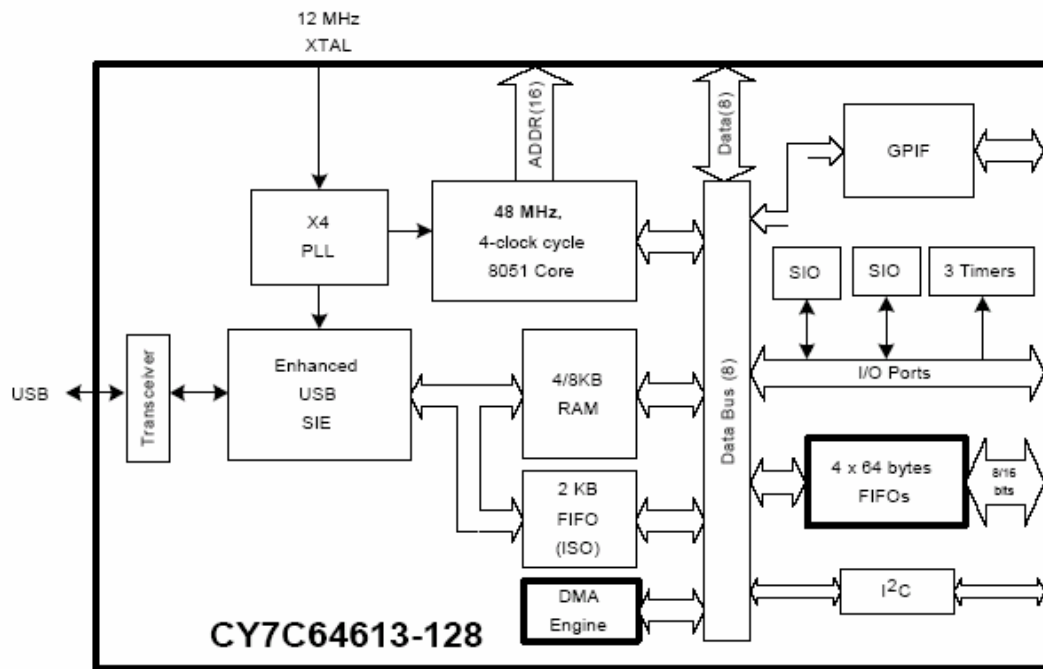


Figure 2-1 CY7C64613 (128 pin) simplified block diagram

Reference: Cypress Semiconductor

## 2.2 Features

- Single-chip integrated USB Transceiver, Serial Interface Engine (SIE), and Enhanced 8051 Microprocessor
- Soft: 8051 runs from internal RAM, which is: downloaded via USB, or loaded from EEPROM
- 14 Bulk/Interrupt endpoints, each with a maximum packet size of 64 bytes. 16 isochronous endpoints, with 2 KB of buffer space (1 KB, double buffered) which may be divided among the sixteen isochronous endpoints
- Integrated, industry standard 8051 with enhanced features: 4 clocks per cycle, 2 UARTS, 3 counter/timers and 256 bytes of register RAM.

- Integrated I2C™ controller
- Five 8-bit IO ports
- 48-MHz or 24-MHz 8051 operation selectable by EEPROM

configuration byte.

- Four integrated general purpose 8-bit FIFOs
- DMA Controller
- Moves data between slave FIFOs, memory, and ports
- Very fast transfers—one clock (20.8 ns) per byte for internal

transfers

- Can use external RAM as additional FIFO (addressed through A/D

buses)

- General Programmable Interface (GPIF)
- Allows direct connection to most parallel interfaces: 8- and 16-bit
- Programmable Waveform Descriptors and Configuration Registers

to define waveforms

- Supports multiple Ready (RDY) inputs and Control (CTL) outputs

Cypress' EZ-USB FX family is available in three packages: 52 PQFP, 80 PQFP, and 128 PQFP. The CY7C64613, a 128-pin version of the EZ-USB FX, has 40 IO pins, a 16-bit address bus and 8-bit data bus for external memory expansion. We will use this chip for our project. Figure

2-2 is the final system diagram of EZ-USB FX.

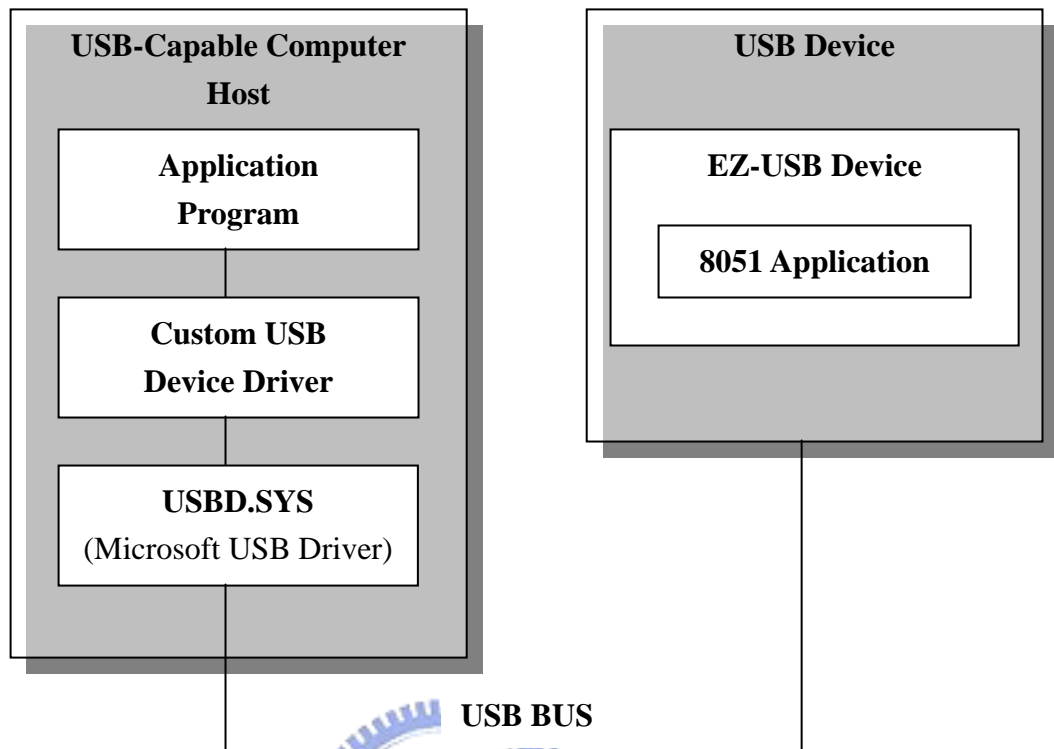


Figure 2-2 Final system diagram of EZ-USB FX

Reference: Cypress Semiconductor

We need to work on the parts of Windows application program and device firmware configuration and 8051 application program. Then using the drivers provided by Cypress and Microsoft Windows. Figure 2-3 is the flow chart of EZ-USB enumeration and re-enumeration.



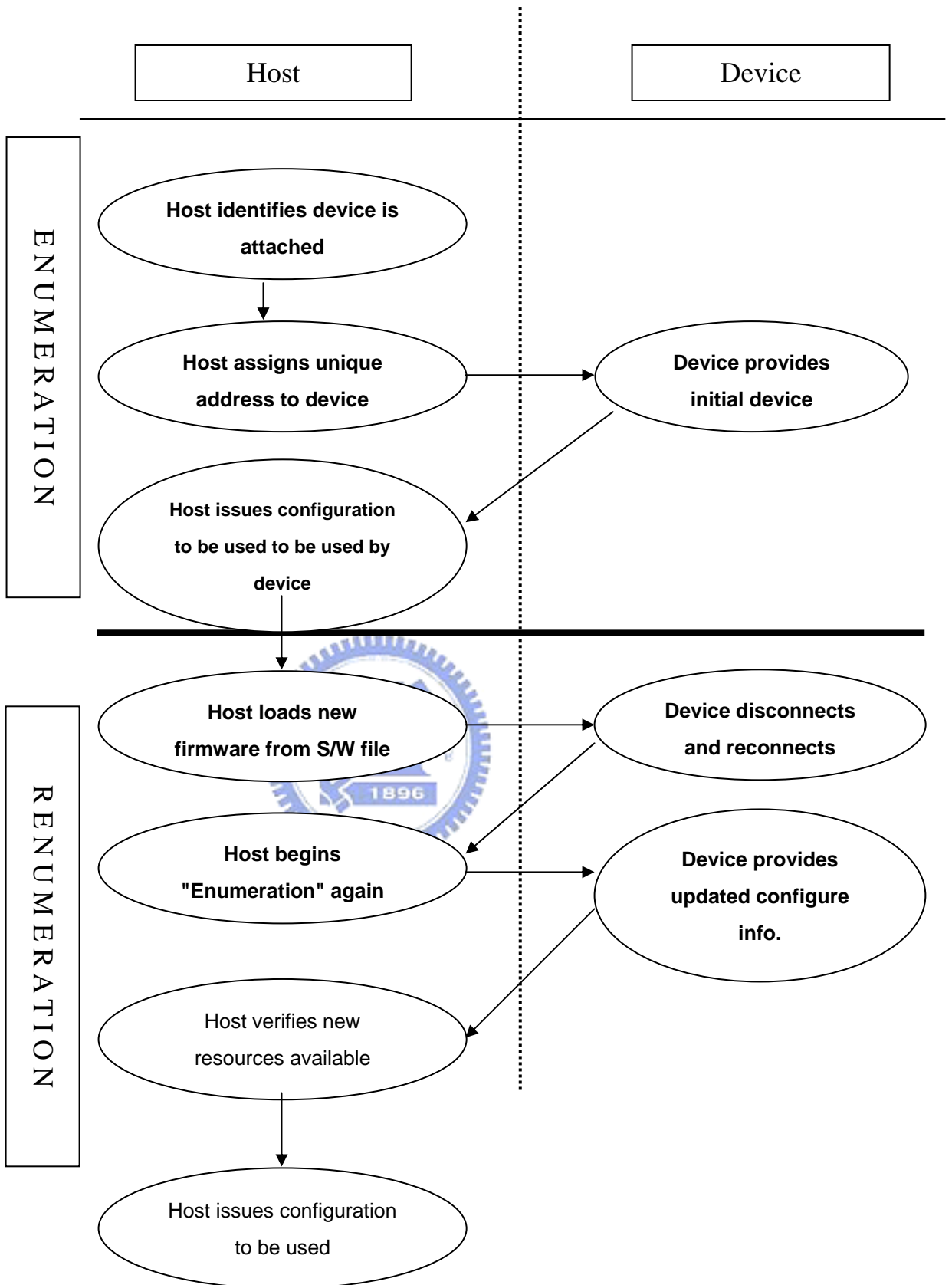


Figure 2-3 Enumeration & Renumeration

# Chapter 3

## Hardware Development

### 3.1 EZ-USB

The EZ-USB FX (CY7C64613) has 128 pins, including an enhanced 8051. Supply voltage is from +3.0V to +3.6V, with an oscillator 12MHz +/- 0.25%. And DC input voltage to any pin is from -0.5V to +5.8V.

Figure 3-1 is a block diagram of the EZ-USB's 8051-based CPU.

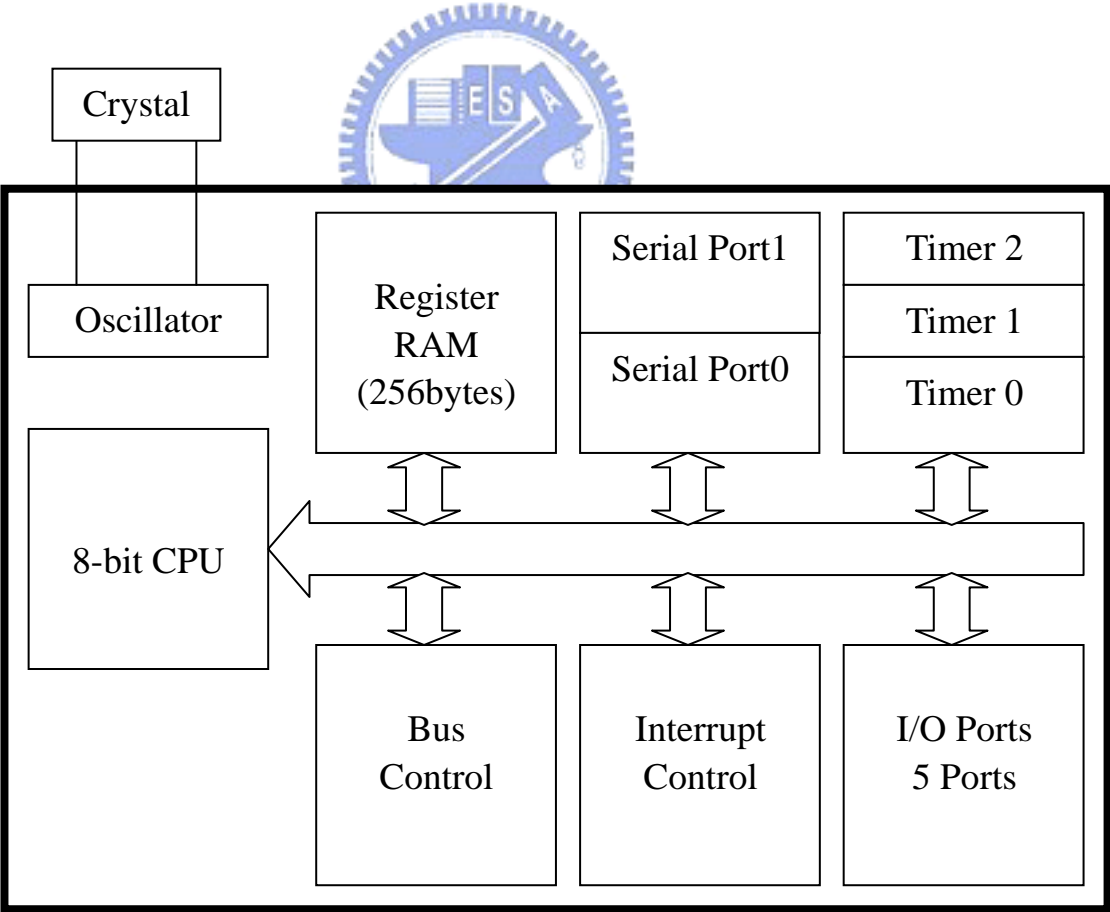


Figure 3-1 EZ-USB 8051-based CPU

There are some notices in the hardware fabrication. We should build smooth connection in USB signal lines: D+ and D-. The crystal connection is also important. Any rough welding will possibly result the Windows in being not able to recognize the device. Moreover, in order to avoid the noise transients and protect EZ-USB, we add a SN75240 between the USB port and the EZ-USB's D+ and D- pin.

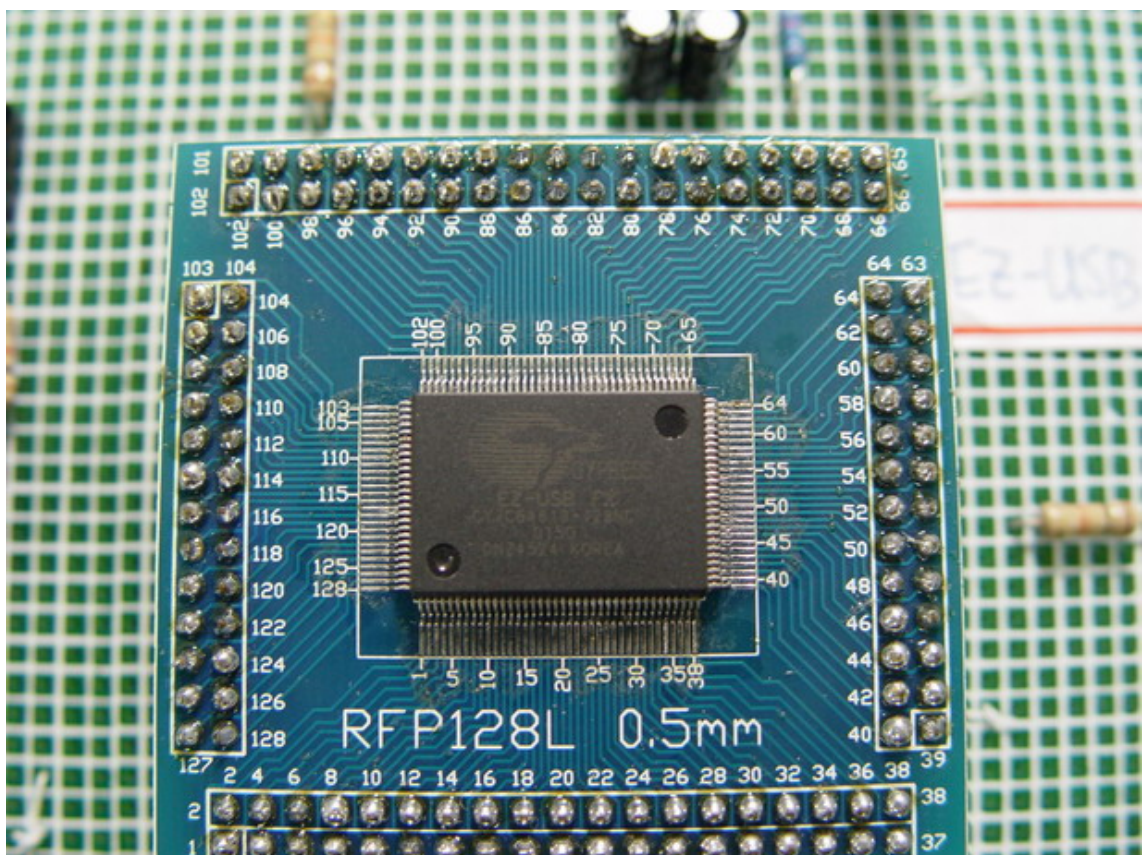


Figure 3-2 EZ-USB FX 128 Pin

## 3.2 AD Converter/ DA Converter

The AD converter, we use ADC0809, which is an 8-channel output, 8-bit resolution, and from 0V to 5V input range with single 5V power

supply. The conversion time of it is 100  $\mu$ s. Figure 3-3 displayed it's time diagram which shows the convert timing.

The DA converter, we use DAC0800, which is a monolithic 8-bit high-speed, current-output, digital-to-analog converter featuring typical settling times of 100ns. Supply voltage from  $\pm 4.5V$  to  $\pm 18V$ . Its output voltage is from -10V to +18V.

To process the analog-to-digital signal, we use EZ-USB's I/O port. Port A to receive the data comes from ADC0809, and Port B to control AD converter's "start" signal and to switch the channel of AD converter.

To process the digital-to-analog signal, we use Port D to send the signal to DA converter.




Figure 3-3 is the circuit of EZ-USB FX. As the schematic shows, there is a 12MHz oscillator connected on pin 19 and pin 20. EZ-USB uses I2C bus communicates with EEPROM. The "SCL" and "SDA" pins are used by I2C bus. Figure 3-4 is analog-to-digital signal input circuit. For AD converter, it has a 1.2MHz oscillator connected on pin 10. Pin 17 to Pin 21 are the signal lines connected to EZ-USB's Port A, and the channel selection pins are controlled by Port B. Figure 3-5 is digital-to-analog signal output circuit. We use an operation amplifier to output the DA signal.

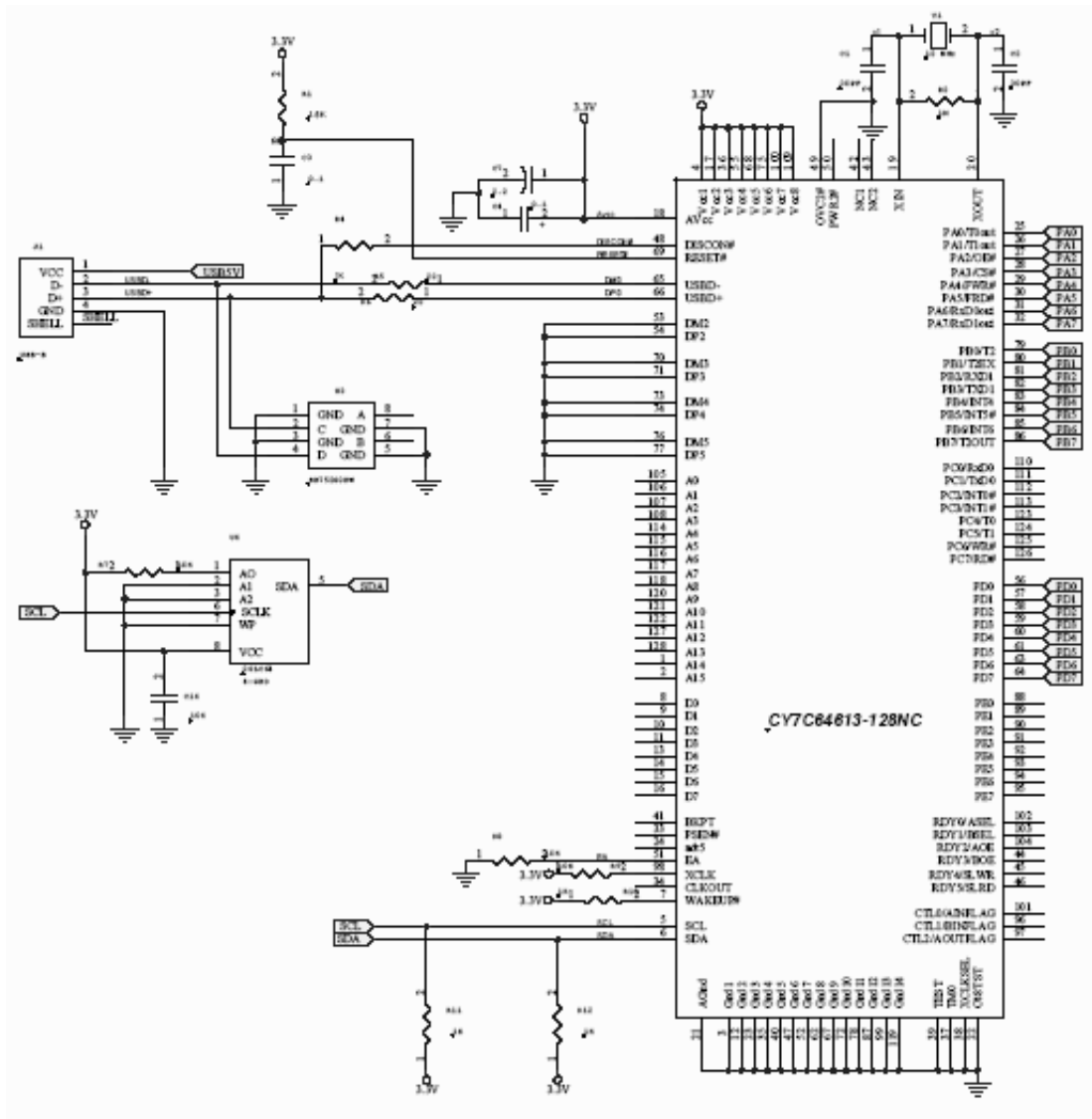


Figure 3-3 Main Schematic

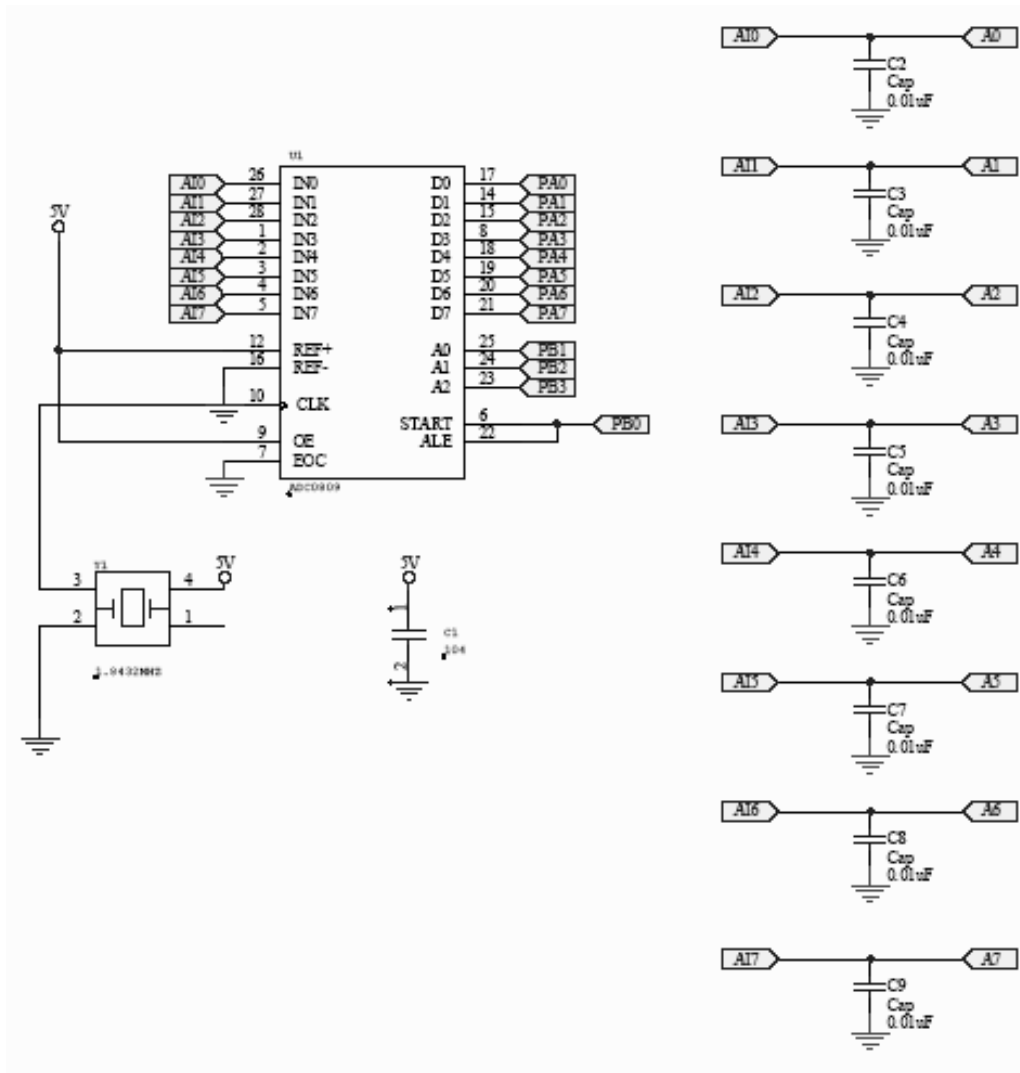


Figure 3-4 Analog-to-digital input Schematic

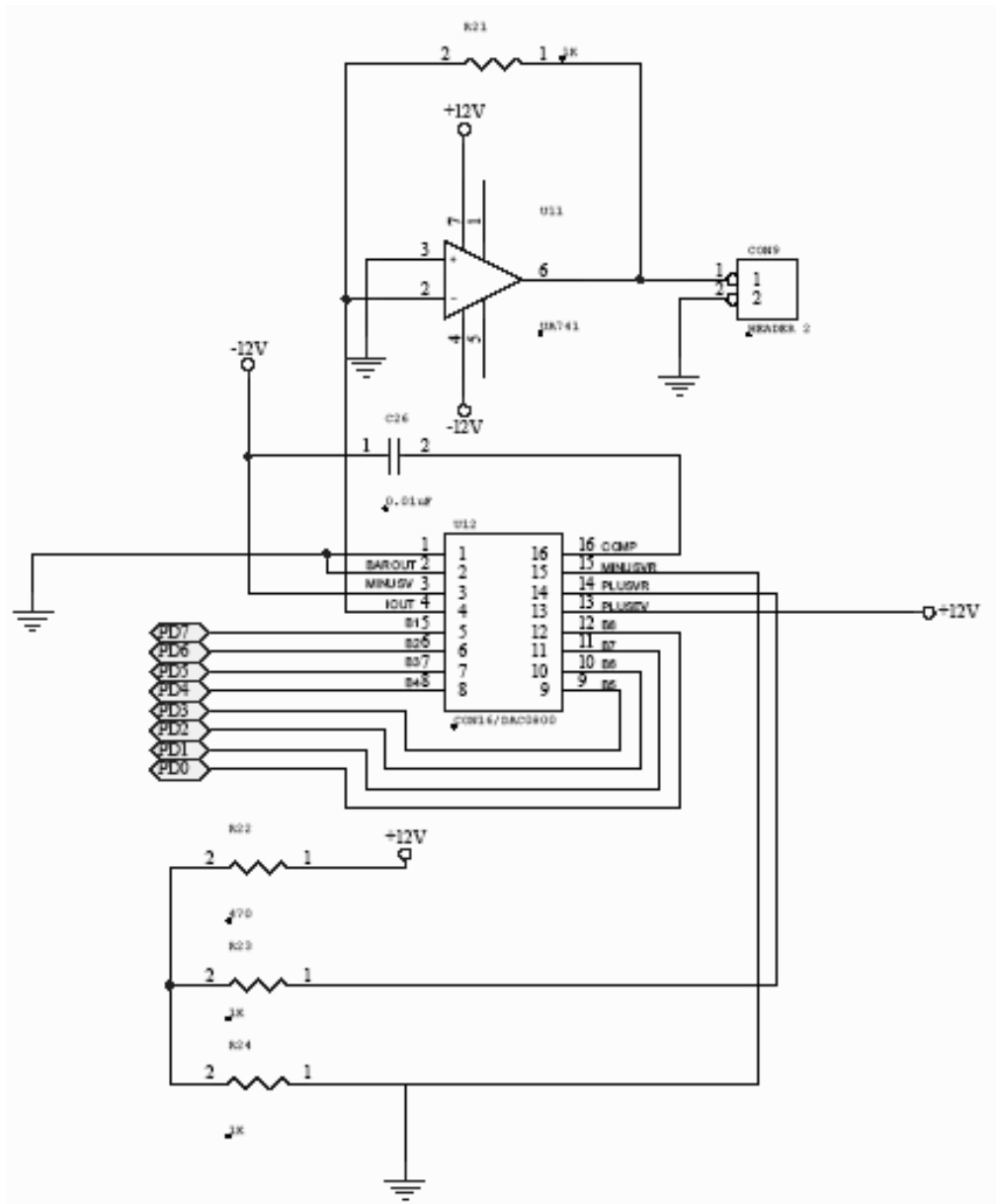


Figure 3-5 Digital-to-analog output  
Schematic



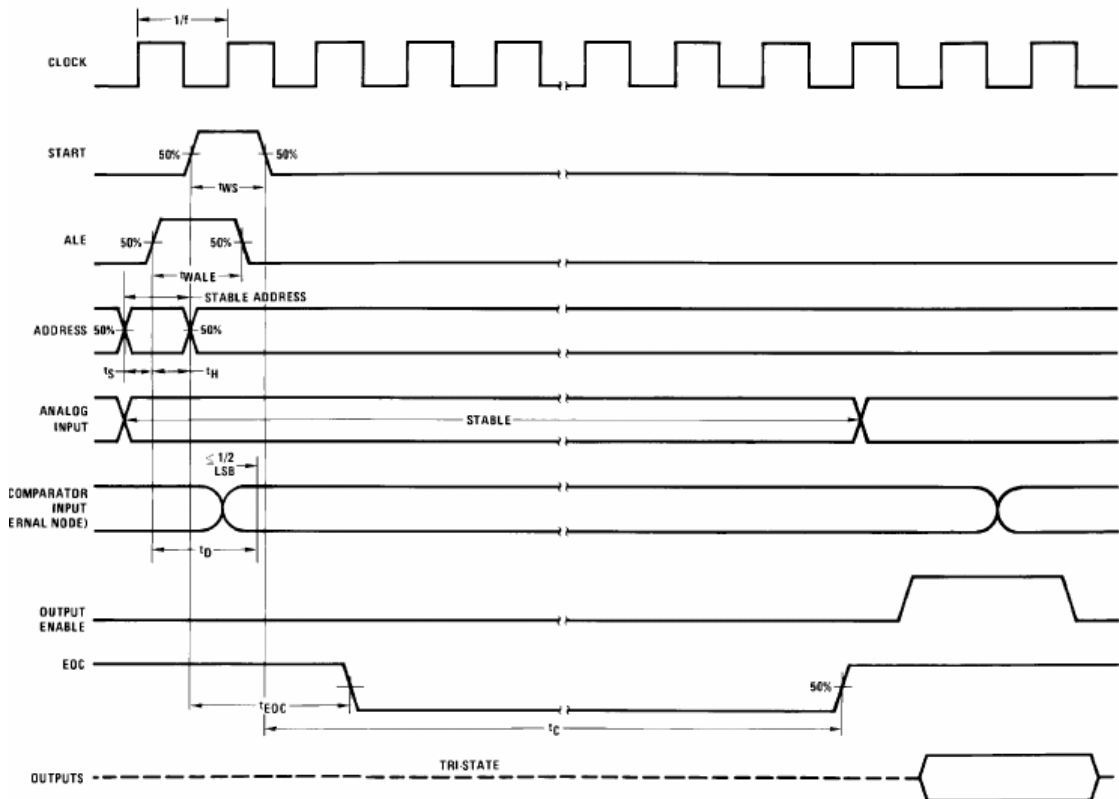


Figure 3-6 ADC0809 time diagram

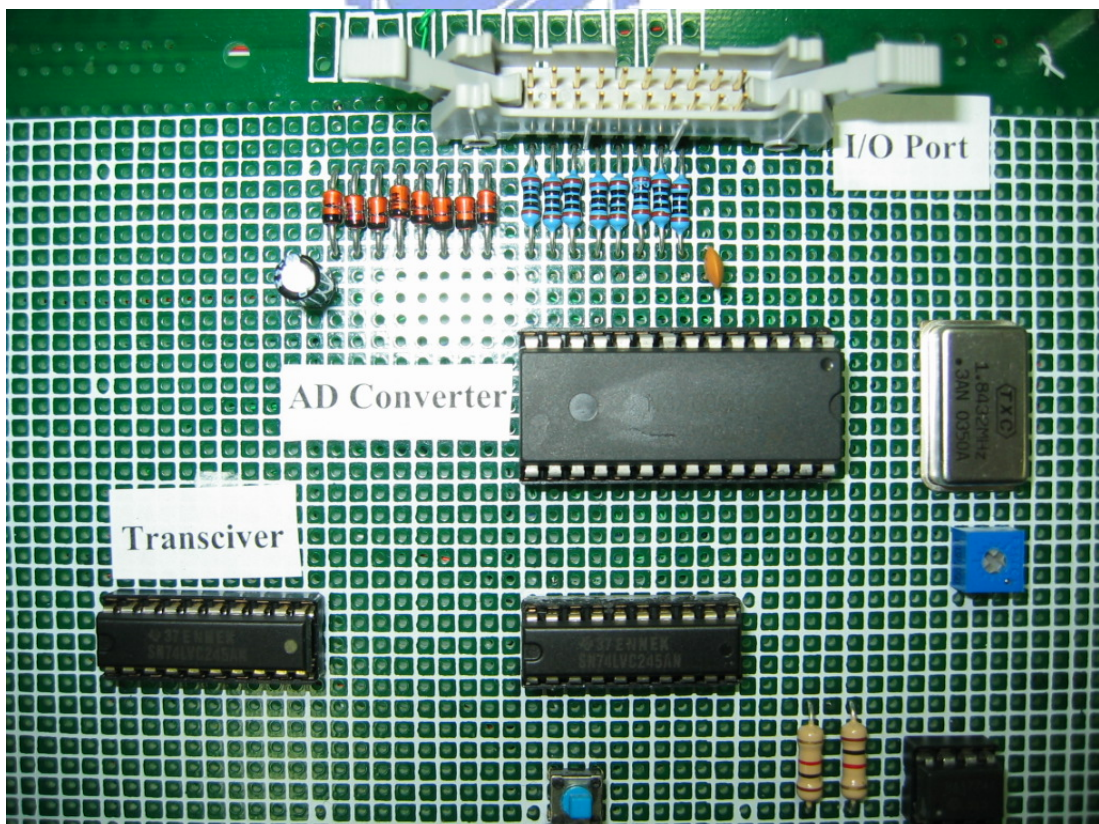


Figure 3-7 Photo of the AD converter



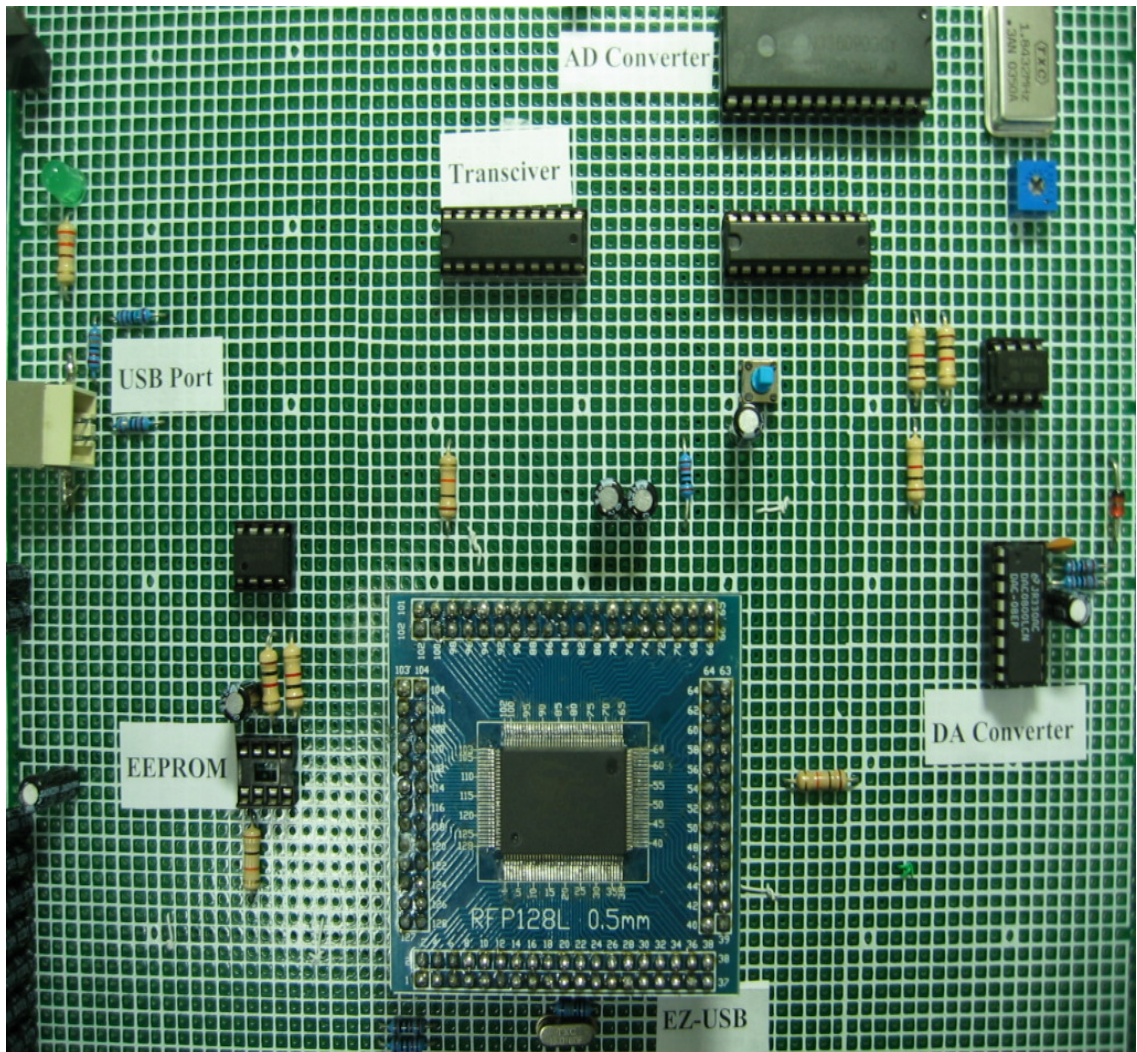


Figure 3-8 Prototype of the HID AD/DA Processing System

# Chapter 4

## Firmware Development

In this chapter, the firmware development of USB device will be briefly introduced.

### 4.1 Firmware configuration

EZ-USB firmware contains some files; some are provided by Cypress, other files we have to configure on our own. Table 4-1 shows the file that configures a firmware needs.

Table 4-1 Files that making a EZ-USB firmware needs

File	Description
FW.c	EZUSB Firmware code
Periph.c	User Function code
Dscr.a51	USB Descriptor table
EZUSB.lib	EZUSB Library( Provide by Cypress)
EZUSB.h	EZUSB header code( Provide by Cypress)
EZREGS.h	EZUSB register header code(Provide by Cypress)

EZ-USB firmware implements a simple co-operative tasking executive. Figure 4-2 is the flow chart of the firmware. Cypress has the example code for EZ-USB's basic firmware, so that we merely concentrate on the configuration that the input and output need, like the

periphery function program.

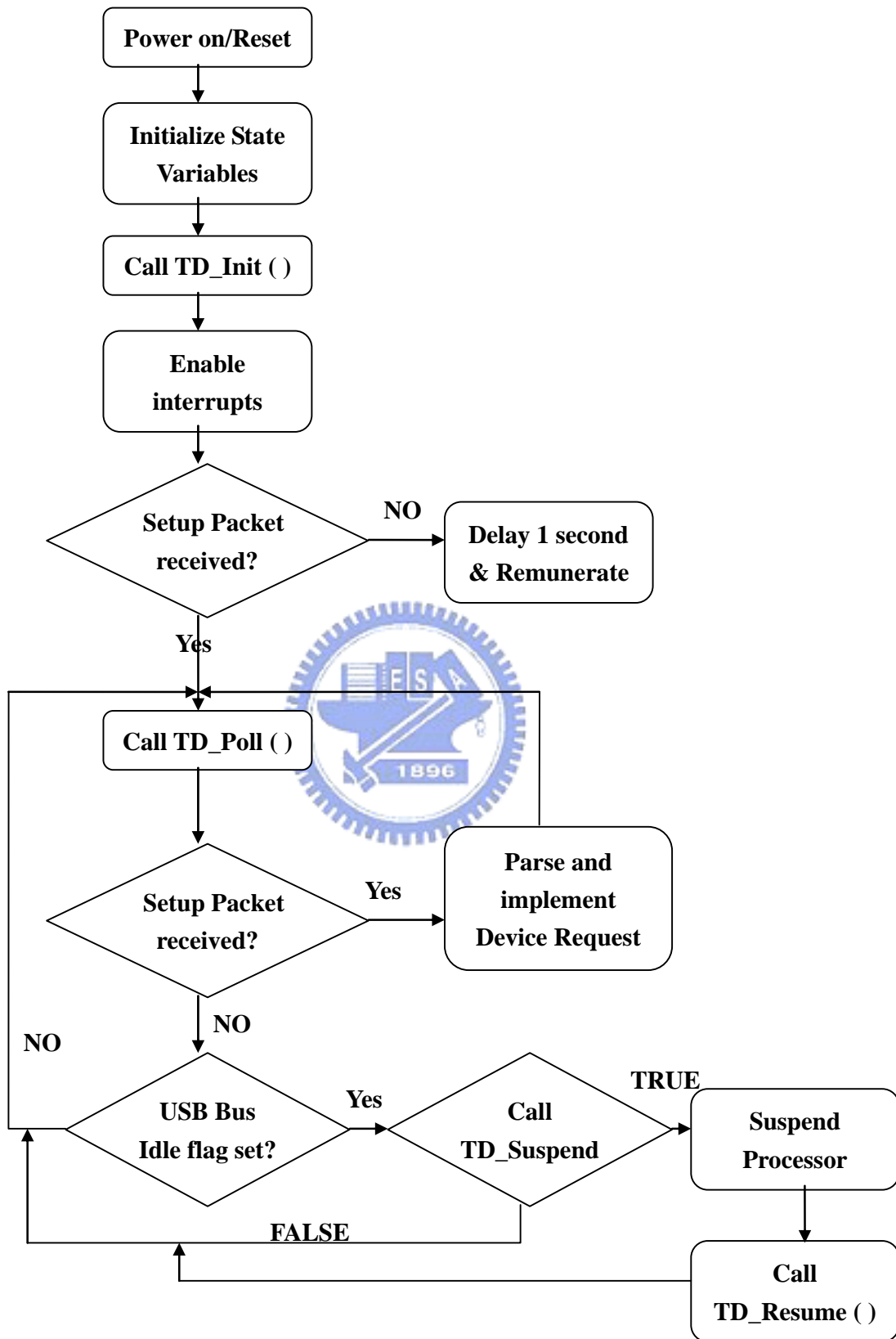


Figure 4-1 Flow chart of the EZ-USB driver function

We use an EEPROM to store our firmware. During the enumeration, EZ-USB will load the program form EEPROM by I<sup>2</sup>C bus. The detail of downloading firmware to EEPROM can be found in EZ-USB document.

In this project, we would like to carry out to an AD/DA processing, thus we will configure the EZ-USB chip which has 2 endpoints: 1 for IN and 1 for OUT. Because that the AD converter is an 8-bit resolution and 8 channels converter. Each endpoint is configured to have 64 bytes packet, and polling interval is 1ms. To implement these endpoints, we shall setup the related information in device descriptor, and the periphery function program.



#### **4.1.1 USB Descriptor**

All USB devices have their own descriptors. Descriptor data may include standard device descriptors, class descriptors, and user specific descriptors. There are 5 to 7 descriptor types in USB. The order of USB descriptors is listed in Figure 4-4 .And Figure 4-5 are descriptors constructions of USB. A HID device should have a HID descriptor and a Report descriptor. In the EZUSB firmware, the device descriptors are contented in DSCR.A51. Figure 4-6 listed the Device descriptor, Configuration descriptor, Interface descriptor in my USB device.

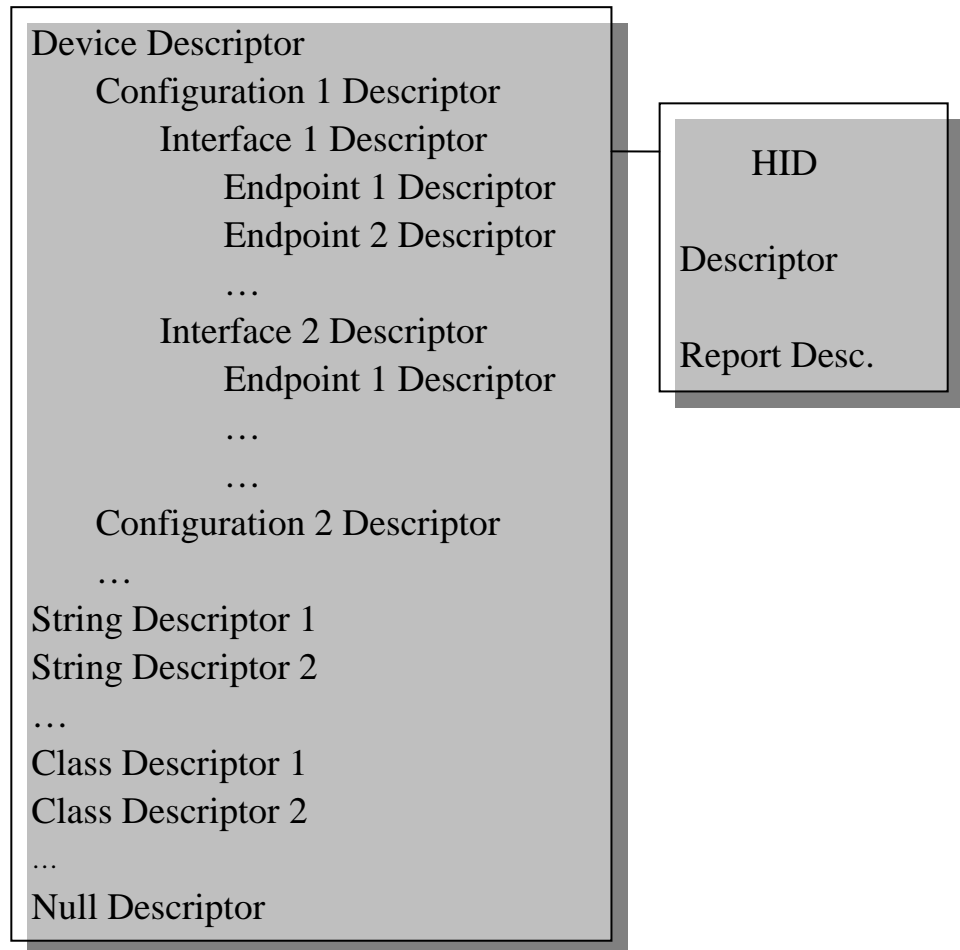


Figure 4-2 The order of USB descriptors

Device		Configure		Interface		Endpoint		HID descriptor	
Length =18	Type 1	Length =9	Type 2	Length =9	Type 4	Length = 7	Type = 5	Length = 9	Type = 21H
USB Ver.	Class	Total length.	Interfaces	This interface	Alternate	EP. Addr.	Attributes	County code	HID descriptor
SubClass	Protocol	ThisConfig	Config Name	Endpoint	Class	Max. Size	Max. Size	Report = 21H	Total Length
EP0 Size	VID	Attributes	Max power	Subclass	Protocol	Interface name	Max. Size	Report = 21H	Total Length
PID	Ver. Num.						Max. Size	Report = 21H	Total Length
	Manufacturer						Max. Size	Report = 21H	Total Length
	PrductName						Max. Size	Report = 21H	Total Length
	SerialNumbe						Max. Size	Report = 21H	Total Length
	Configuration						Max. Size	Report = 21H	Total Length

Figure 4-3 Descriptors information

(Device Descriptor)

db 18	:: Descriptor length
db DSCR_DEVICE	:: Descriptor type
db 10H,01H	:: Specification Version (BCD)
db 00H	:: Device class
db 00H	:: Device sub-class
db 00H	:: Device sub-sub-class
db 64	:: Maximum packet size
dw 3412H	:: Vendor ID ;*****
dw 7856H	:: Product ID ;*****
db 01h,00h	:: Product version ID
db 1	:: Manufacturer string index
db 2	:: Product string index
db 0	:: Serial number string index
db 1	:: Number of configurations

(Configure Descriptor)

db 9	:: Descriptor length
db DSCR_CONFIG	:: Descriptor type
db EPDscrEnd-ConfigDscr	:: Configuration + End Points length (LSB)
db 00	:: Configuration length (MSB)
db 1	:: Number of interfaces
db 1	:: Interface number
db 0	:: Configuration string
db 01100000b	:: Attributes (b7 - buspwr, b6 - selfpwr, b5 - rwu)
db 250	:: Power requirement (div 2 ma)

(Interface Descriptor)

db 9	:: Descriptor length
db DSCR_INTRFC	:: Descriptor type
db 0	:: Zero-based index of this interface
db 0	:: Alternate setting
db 2	:: Number of end points
db 03H	:: Interface class(HID:03H)
db 00H	:: Interface sub class
db 00H	:: Interface sub sub class
db 0	:: Interface descriptor string index

Figure 4-4 Descriptors in USB



## 4.1.2 Peripheral Circular Program

In EZUSB, there is an enhanced 8051 inside. The peripheral circular is used to setup the 8051 and initial the EZ-USB chip. Cypress Semiconductor had provided the driver and contained some functions that we can use. The flow chart of the EZ-USB driver function is in Figure 4-2.

The firmware was written, using the Keil C51 C compiler and tools. The compiler will create a HEX file that we can download to EZ-USB. EZ-USB has an enhanced 8051 in it, and 8051 is used to take care of input/ output control such as control AD converter and DA converter. The flowchart of the signal between the host and USB is in Figure 4-7. In the “Write” stage, PC sends an “OUT” request to USB at the first, after USB sends an “ACK” back to the PC, and then the PC will send the output data to USB. In the “Read” stage, PC sends an “IN” request to USB at the first, the USB will send the data in endpoint buffer back. The endpoint buffer contains the data that 8051 read from the AD converter.

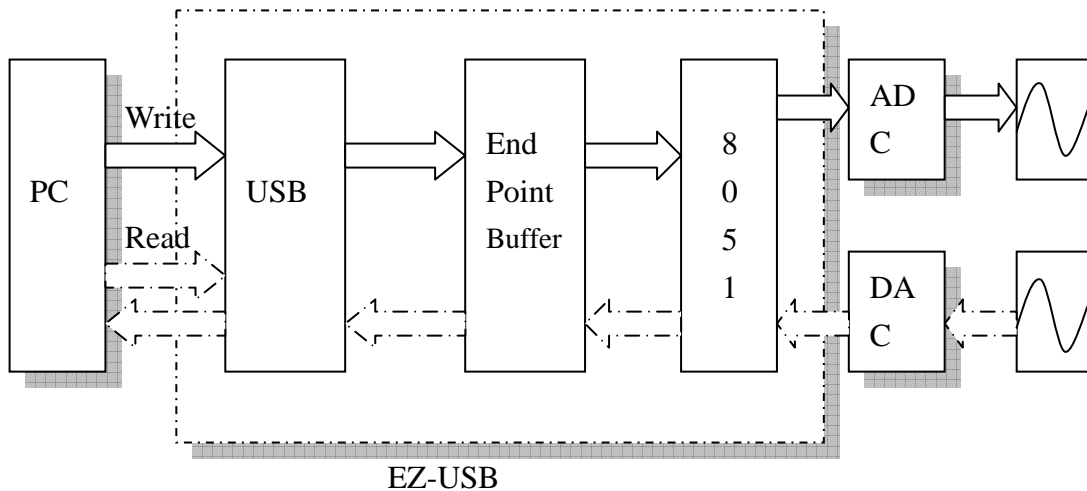


Figure 4-5 The flowchart of the signal between host and USB

## 4.2 HID class

HID (Human Interface Device) is the most useful class in USB application. There are many applications in HID: mice, keyboard, joy sticker are good examples of HID. A HID device does not require a human interface. After Windows 98 SE, Windows have supported HID. It means that user does not need to install driver for HID device. A HID-Class device supplies low amounts of data at infrequent times. The HID transfer speed is 800 Bytes/s at low-speed, 64KB/s at full-Speed, 24MB/s at high-speed. The latest version of HID is 1.1. EZ-USB FX (CY7C64613) is a full-speed USB chip. Therefore, I enumerate my device as a full-speed HID device.

The HID descriptor is a specific descriptor for HID Device, as Figure 3-4 shows. To configure a HID-class device, the class code in interface



descriptor must set to 3. The PC host will look for a HID Descriptor and a Report descriptor. This is an example code below, the total bytes is 9 bytes. The second parameter must be 21 for HID device, and the current version of HID is 1.1.

db 09h	; length
db 21h	; type: HID
db 10h,01h	; release: HID class rev 1.1
db 00h	; country code (none)
db 01h	; number of HID class descriptors
db 22h	; report descriptor type (HID)

Report descriptor is the unique descriptor in HID device. HID device communicates with PC host by sending report descriptors. Report descriptor contains input report and output report. While processing communication, HID sends data into the host according to the input report, and uses output report to process the data coming from host. To write the report descriptor, we can use Report Generator Tool, which is provided by USB-IF. Figure 4-8 is the report descriptor of HID device. The input report size is 8 bits, and report count is 63. The output report size is 8 bits, and report count is 8.

```

ReportDscr:
    db  06H, 0A0H, 0FFH  ;; Usage Page (vendor defined)
    db  09H, 01H        ;; Usage (Vendor defined)
    db  0A1H, 01H       ;; Collection (Application)
    db  09H, 02H        ;; Usage (vendor defined)
    db  0A1H, 00H       ;; Collection (Physical)
    db  06H, 0A1H, 0FFH ;; Usage Page (vendor defined)

;; The input report
    db  09H, 03H        ;; Usage (vendor defined)
    db  09H, 04H        ;; Usage (vendor defined)
    db  15H, 80H        ;; Logical minimum (80H = -128)
    db  25H, 7FH        ;; Logical maximum (7FH = 127)
    db  35H, 00H        ;; Physical minimum (0)
    db  45H, 0FFH      ;; Physical maximum (255)
    db  75H, 08H        ;; Report size (8 bits)
    db  95H, 3FH        ;; Report count (63 fields)
    db  81H, 02H        ;; Input (Data, Variable, Absolute)

;; The output report
    db  09H, 05H        ;; Usage (vendor defined)
    db  09H, 06H        ;; Usage (vendor defined)
    db  15H, 80H        ;; Logical minimum (80H = -128)
    db  25H, 7FH        ;; Logical maximum (7FH = 127)
    db  35H, 00H        ;; Physical minimum (0)
    db  45H, 0FFH      ;; Physical maximum (255)
    db  75H, 08H        ;; Report size (8 bits)
    db  95H, 08H        ;; Report count (8 fields)
    db  91H, 02H        ;; Output (Data, Variable, Absolute)

    db  0C0H            ;; End Collection (Physical)
    db  0C0H            ;; End Collection (Application)
ReportDscrEnd:

```

Figure 4-6 Report descriptor

# Chapter 5

## Host Application

There are three parts that we should put into consideration: device firmware, windows user application, and device driver. The flow chart is showed below as Figure 5-1.

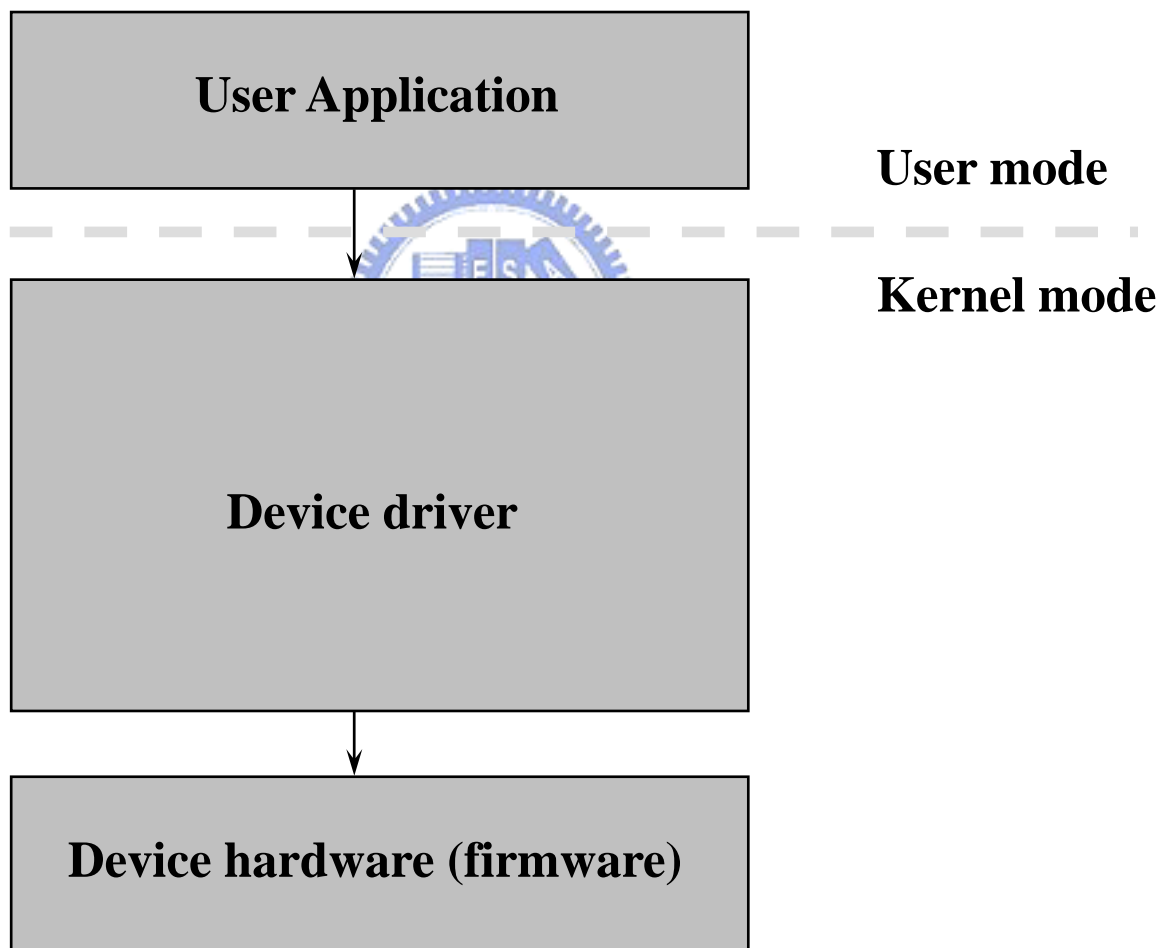


Figure 5-1 The flow chart of USB software development

All USB devices have a Vendor ID (VID) and a Product ID (PID) which are reported to Windows in the device descriptor. Windows uses the

VID and PID to detect the appropriate device driver. The INF file is what ties a VID/PID combination to a specific driver. Figure 4-3 shows the relation between PID/VID and USB device.

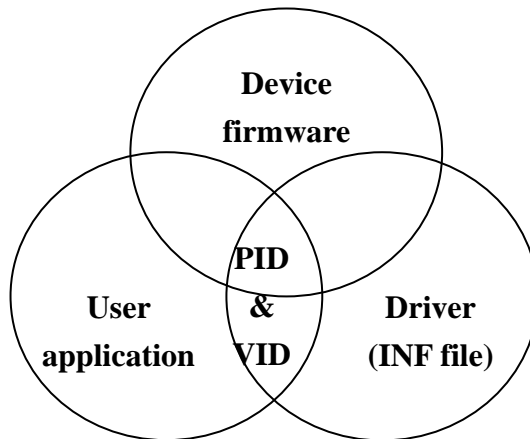


Figure 5-2 PID and VID

## 5.1 Windows Application

In USB device development, the host application is another important part. There are many programming languages that we can use to develop a windows application, here I choose Visual C++ as the development tool.

A user mode application firstly gets a handle to the device driver via a call to Win32 function "CreateFile ( )". Then the user mode application uses Win32 function "DeviceIoControl ( )" to submit an I/O control code and related input and output buffers to the driver through the handle returning by "CreateFile ( )". These two Win32 functions are provided by Visual C++. And other I/O Control Code (IOCTL) will refer to The EZ-USB General Purpose Driver.

There is another way to communicate to USB. If we configure our device as a HID (Human Interface Device), therefore we can use Windows API to establish communication with USB HID more easily. Table 5-1 lists some useful Windows API for HID device. In order to write a HID program, we should equip Windows DDK (Driver Development Kit). For the reason that there are many of the Windows system libraries and declaration files within Windows DDK.

The process of opening a device consists of several steps.

Step 1: First of all before windows application communicates to a HID device, it should obtain the Windows GUID (globally unique ID) of HID. GUID has a 128 bits length. Each object all has its own GUID; The GUID of HID class is contained in hidclass.h. We can use “HidD\_GetHidGuid ( )” to obtain the GUID of HID.

```
Example: // API function: HidD_GetHidGuid
         HidD_GetHidGuid(&HidGuid);
```

Step 2: After obtaining the GUID, and then we should get an array of structures that contain information about all attached HID devices. Here we use “SetupDiGetClassDevs ( )” to do this task. The function will return all attached HID devices.

```

Example: // API function: SetupDiGetClassDevs
hDevInfo=SetupDiGetClassDevs
    (&HidGuid,                //ClassGuid,
    NULL,                      //Enumerator,
    NULL,                      //hwndParent,
    DIGCF_DEVICEINTERFACE     //Flags
    );

```

Step 3: Now we use “SetupDiEnumDeviceInterfaces ( )” to get information about a device in the list that got from Step 2. We need to check each index of device information until find the one that matches the VID and PID that our device owns.

```

Example: // API function: SetupDiEnumDeviceInterfaces
Result=SetupDiEnumDeviceInterfaces
    (hDevInfo,                //DeviceInfoSet
    0,                        //DeviceInfoData
    &HidGuid,                 //InterfaceClassGuid
    MemberIndex,             // MemberIndex
    &devInfoData             // DeviceInterfaceData
    );

```

Step 4: When we already got the index of our device, using “SetupDiGetDeviceInterfaceDetail ( )” to return detailed data about the device indexed in the previous step. After this procedure, we can get this device path that we can use to open.

```

Example: // API function: SetupDiGetDeviceInterfaceDetail
Result = SetupDiGetDeviceInterfaceDetail
        (hDevInfo,          //DeviceInfoSet,
        &devInfoData,      //DeviceInterfaceData,
        NULL,              //DeviceInterfaceDetailData,
        0,                 // DeviceInterfaceDetailDataSize,
        &Length,           //RequiredSize,
        NULL               //DeviceInfoData
        );

```

Step 5: Now we can call “CreateFile ( )” to open the device using the path obtained in the previous step.

```

Example: // API function: CreateFile
DeviceHandle=CreateFile
        (detailData->DevicePath,
        GENERIC_READ|GENERIC_WRITE,
        FILE_SHARE_READ|FILE_SHARE_WRITE,
        (LPSECURITY_ATTRIBUTES)NULL,
        OPEN_EXISTING,
        0,
        NULL);

```

Step 6: We should compare the open device’s VID and PID to check the device is what we want to communicate. Here we can use “HidD\_GetAttributes( )” to obtain the attributes of device. If the VID and PID are incorrect, then we need to close the device handle and return to Step 3 to check the next device which the list indexes.

```

Example: // API function: HidD_GetAttributes
Result = HidD_GetAttributes
        (DeviceHandle,     // HidDeviceObject
        &Attributes        // Attributes
        );

```

Step 7: If we have got the device we want, we can use “HidP\_GetCaps( )” to obtain the device's capabilities. Such as Input report length, Output report length can be obtained from the function return. Figure 5-3 is the flow chart of the windows application checks the HID device that we desired in the initial process.

Once we step through all initial communication, we can start to write data to device and read data from device. I use “ReadFile( )” and “WriteFile( )” functions to communicate with my device.

```
Example: //      API function: ReadFile
Result = ReadFile
        (ReadHandle,
         &InputReport,
         Capabilities.InputReportByteLength,
         &NumberOfBytesRead,
         (LPOVERLAPPED) &HIDOverlapped
        );
```

```
Example: //      API function: WriteFile
Result = WriteFile
        (DeviceHandle,
         &OutputReport,
         Capabilities.OutputReportByteLength,
         &NumberOfBytesWritten,
         NULL
        );
```



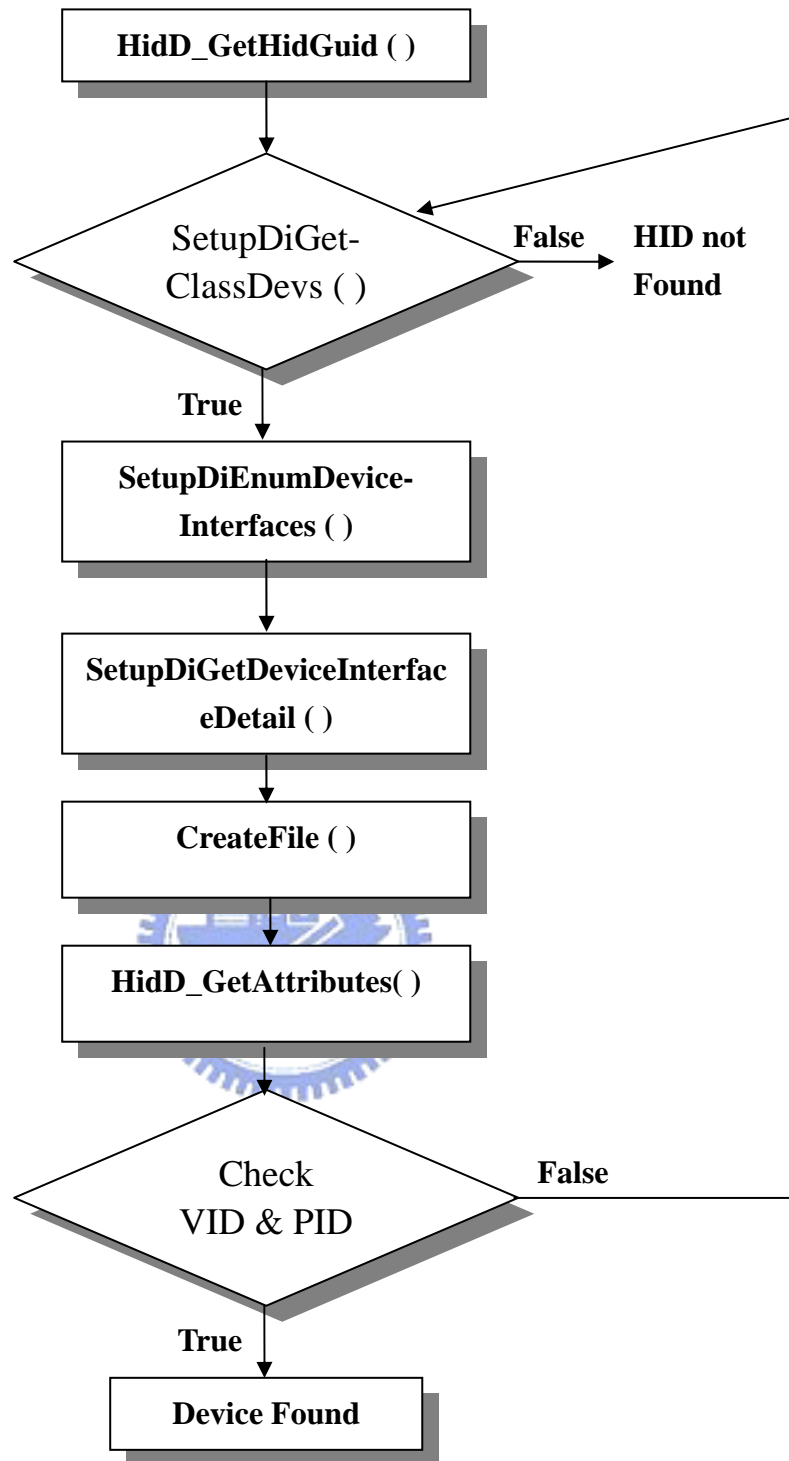


Figure 5-3 The Flow chart of HID device check

In my HID program, there are some features.

1. Detect specific HID device (HID AD/DA Processing System).
2. Show HID Features (input and output features)
3. Analog Output Control (form 0V~ 5V).

4. Analog Signal Receive

5. Graphical Display.

Figure 5-4 is the flow chart of my Windows program to communicate with HID device. The looping time is set to 10ms. There are 2 high-resolution timers in this program, “Timer1” and “Polling interval”. Polling interval is used in order to record the one procedure (Write function to Display function). Timer1 is to record the time that the program loops specific times spend. The program also generates a file which is used to record the transfer data. Due to the graphical display function, it would reduce the transfer performance so this function can be cancelled by user. When we test the performance, we turn off this function.

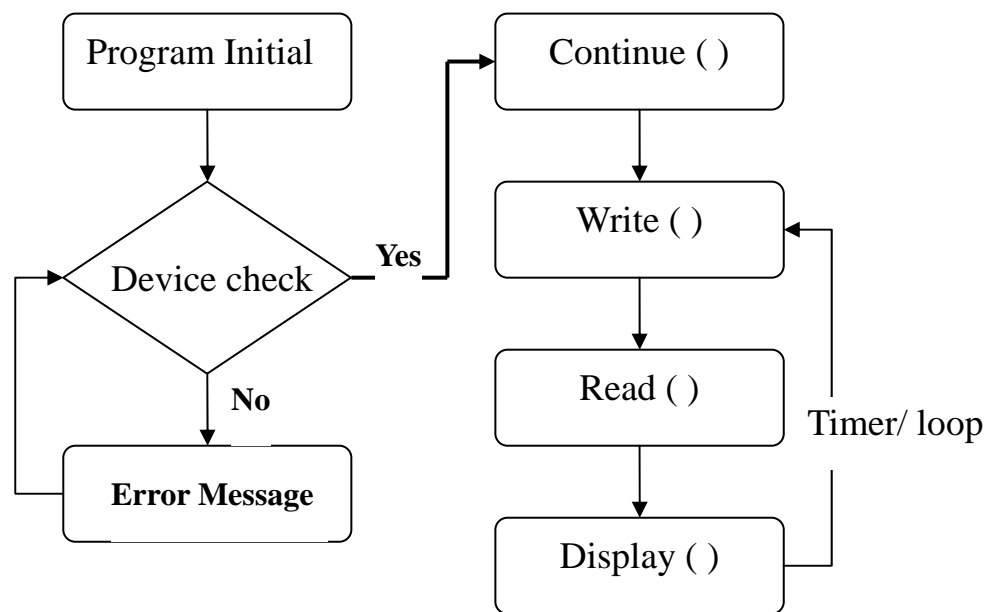


Figure 5-4 The flow chart of the program

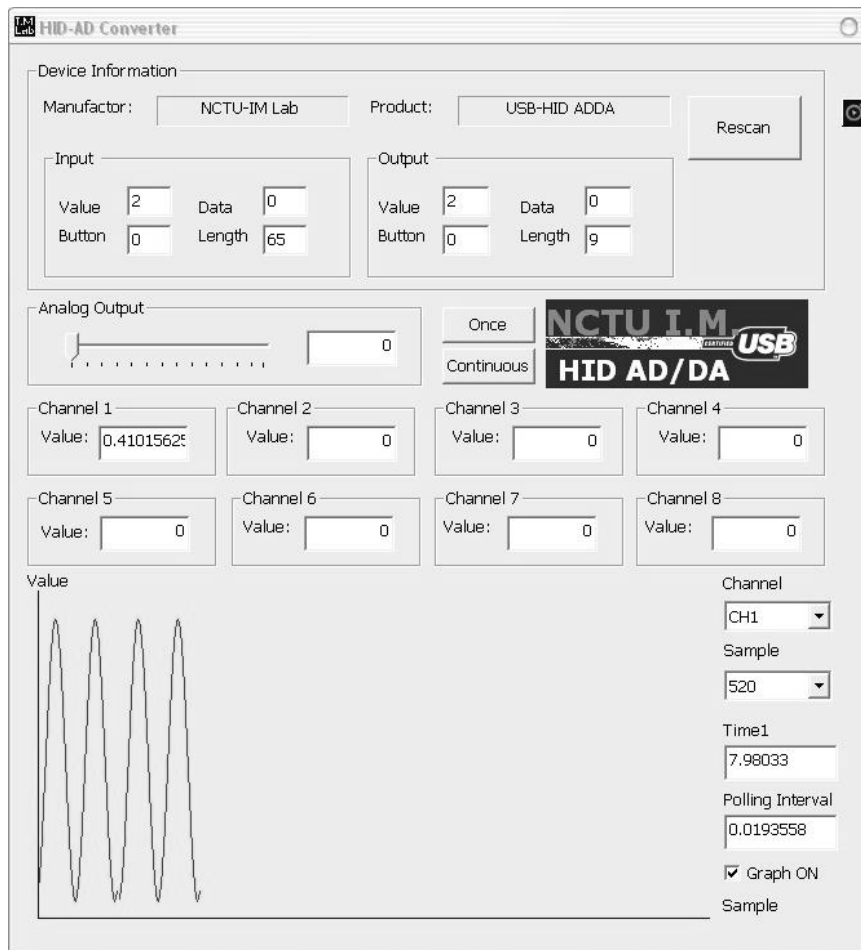


Figure 5-5 Snapshot of the program

Table 5-1 Useful Windows API for HID

API function	Use in HID communication
HidD_GetHidGuid	Returns the GUID associated with HIDs
SetupDiGetClassDevs	Returns an array of structures containing information about all installed HIDs
SetupDiEnumDevice-Interfaces	Returns a pointer to a structure that identifies an interface in the array returned by SetupDiGetClassDevs
SetupDiGetDevice-InterfaceDetail	Returns a device pathname for a specified device interface
CreateFile	Opens a handle to a HID using the pathname returned by SetupDiGetDevice-InterfaceDetail
HidD_GetAttributes	Returns the Vendor ID, Product ID, and Version for a specified HID
HidD_GetPreparsedData	Returns a handle to a buffer with information about a device's capabilities
HidD_GetCaps	Returns a structure describing a device's capabilities
HidD_GetValueCaps	Returns a structure describing the values in a device port
ReadFile	Reads in input report from a specified HID
WriteFile	Sends out output report to a specified HID

## 5.2 Hardware Driver

As shown in Figure 5-1, the device driver plays a role of connector between Windows and hardware. But USB driver doesn't talk directly to USB hardware; it talks to Microsoft USB driver called USBD.sys. In EZ-USB, Cypress Semiconductor had provided a driver called: ezusb.sys, so that we can use it directly.

As the previous section was described, we have configured the device

as a HID, so that we don't have to make our own INF file. Windows will recognize the device by using its own INF (in Windows 98, it will use "hiddev.inf". in Windows 2000, it will use "input.inf" for HID device.).

Figure 5-6 The HID device shows up in the device manager when we plug device into USB port.



Figure 5-6 HID device shows up on Device Manager.

# Chapter 6

## Conclusion

### 6.1 Transfer rate calculation

Now we will test our device and its performance. As we know, for the full-speed of HID, the maximum transfer speed is 64KB/s. We should consider the hardware constrains: the ADC's conversion time, and the Start pulse. For ADC0809, the conversion time is 100  $\mu$ s, and we set the Start pulse to about 0.6  $\mu$ s (the minimum time that EZ-USB's internal 8051 can reach). Because the EZ-USB's interrupt endpoint buffer is 64 bytes, so we set the 8051 to get 64 data from AD converter. We use double buffers to save the AD data. When one buffer is full then sent by USB core, 8051 can still save the data to another. The process will continuously repeat when another buffer is full. Each buffer is 64bytes. The USB core polling interval is set to 1 millisecond.

I set the program to run 100 loops and record the total spending time form "Timer1" in my program. The data what EZ-USB transferred will be output to file, "data". The graph display is done with the assist of EXCEL.

## 6.2 Experiment Result

Testing environment:

Operation System: Windows XP Professional;

PC: P4-1.3G, 512M DDR Ram;

USB Host: USB 2.0;

In report size: 64 bytes;

Out report size: 64 bytes;

Packet size: 64 bytes;

Polling interval: 1ms;

Testing results are illustrated as below figures. All figures are marked down 100 samples.

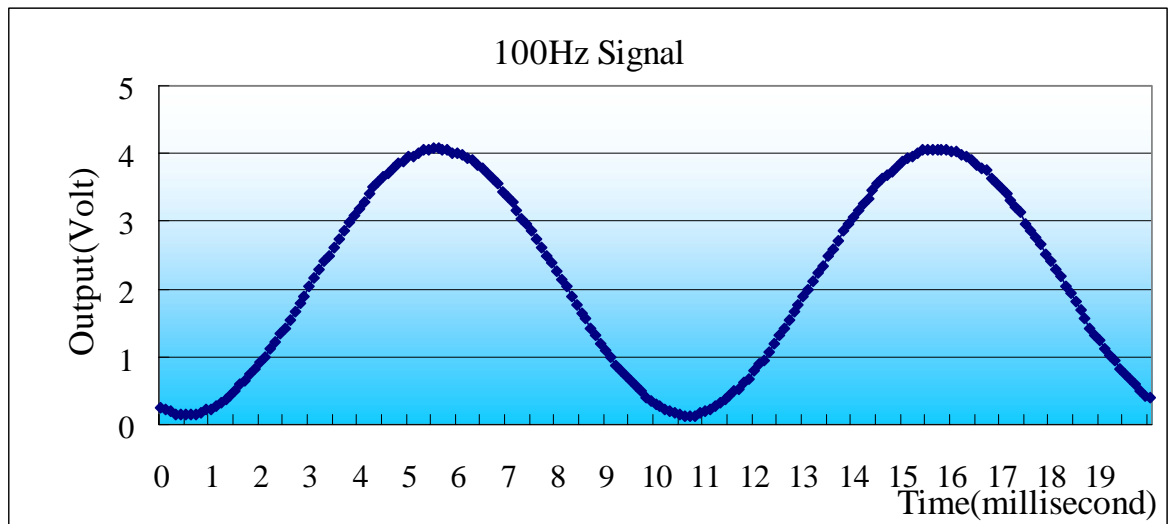
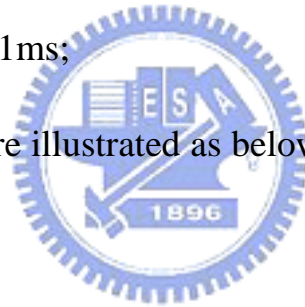


Figure 6-1 100Hz signal

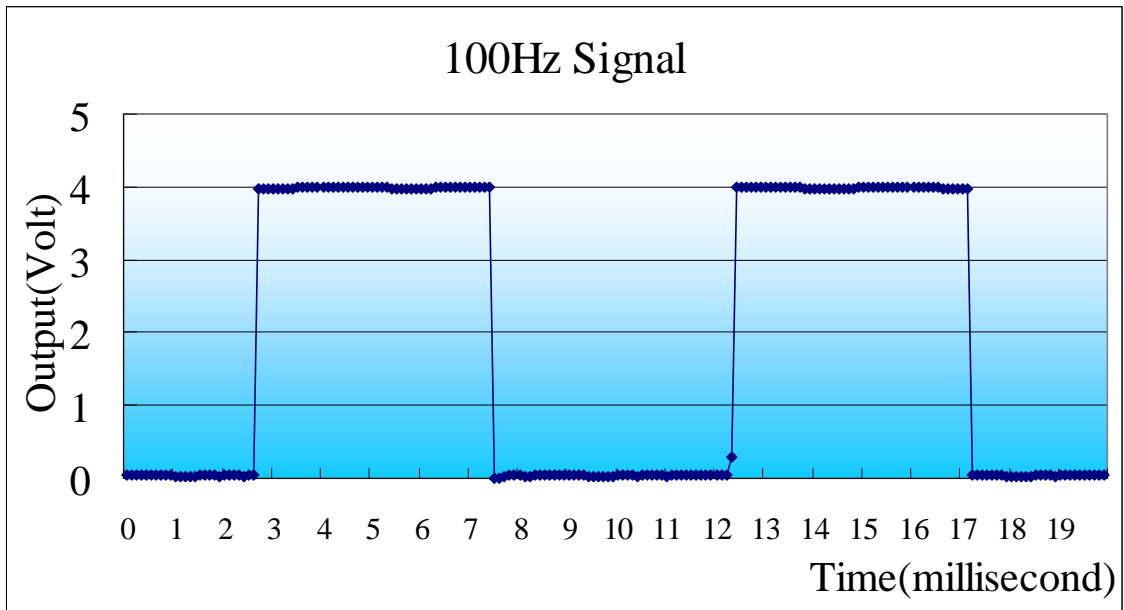


Figure 6-2 100Hz signal (2)

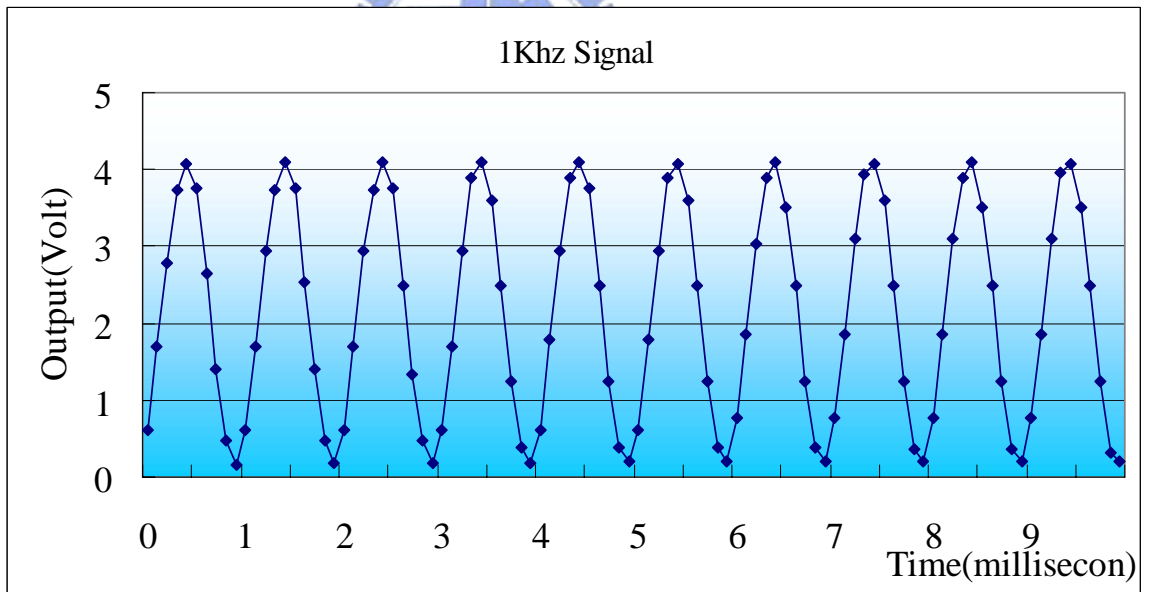


Figure 6-3 1 KHz signal



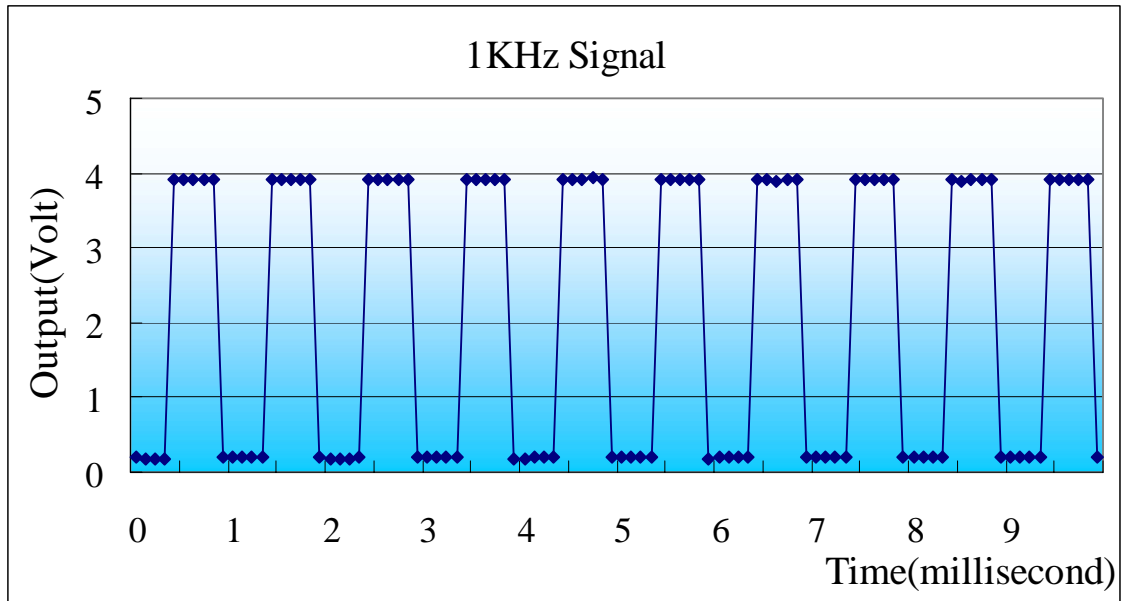


Figure 6-4 1 KHz signal (2)

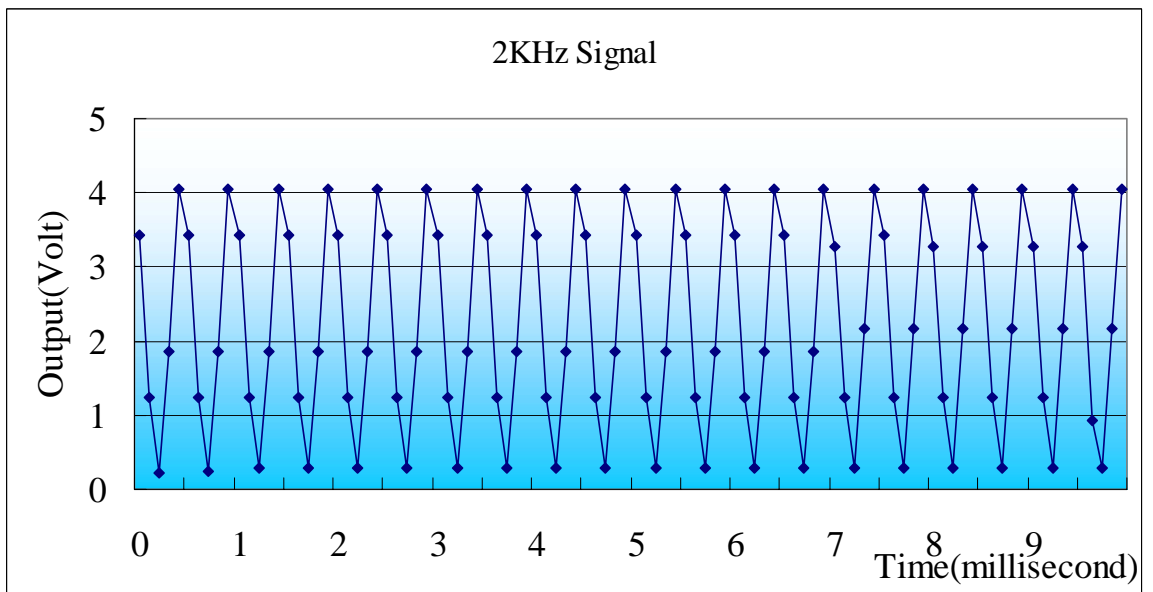


Figure 6-5 2 KHz signal

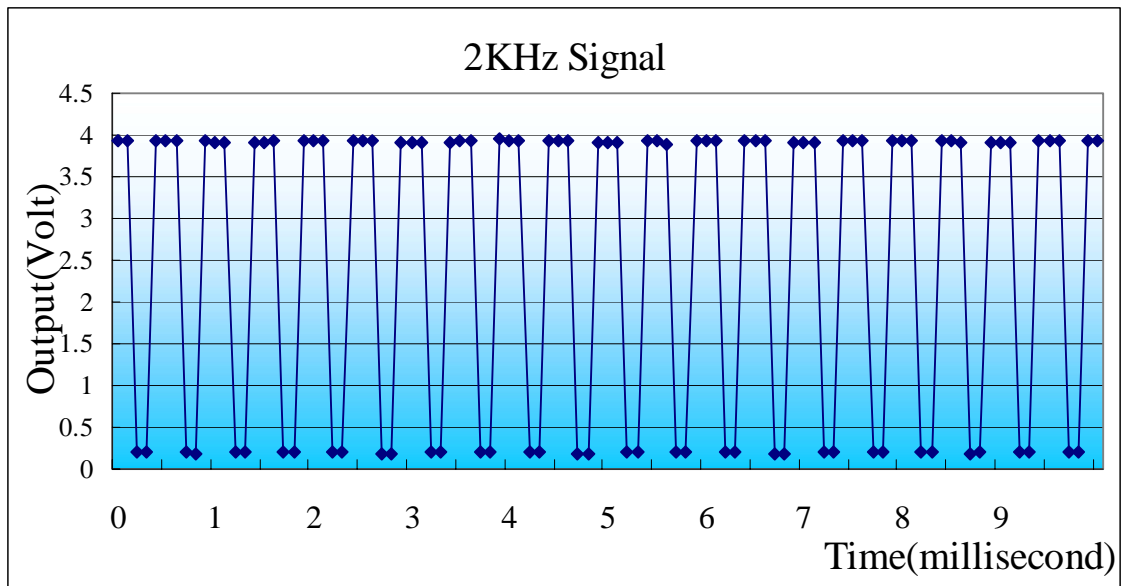


Figure 6-6 2 KHz signal (2)

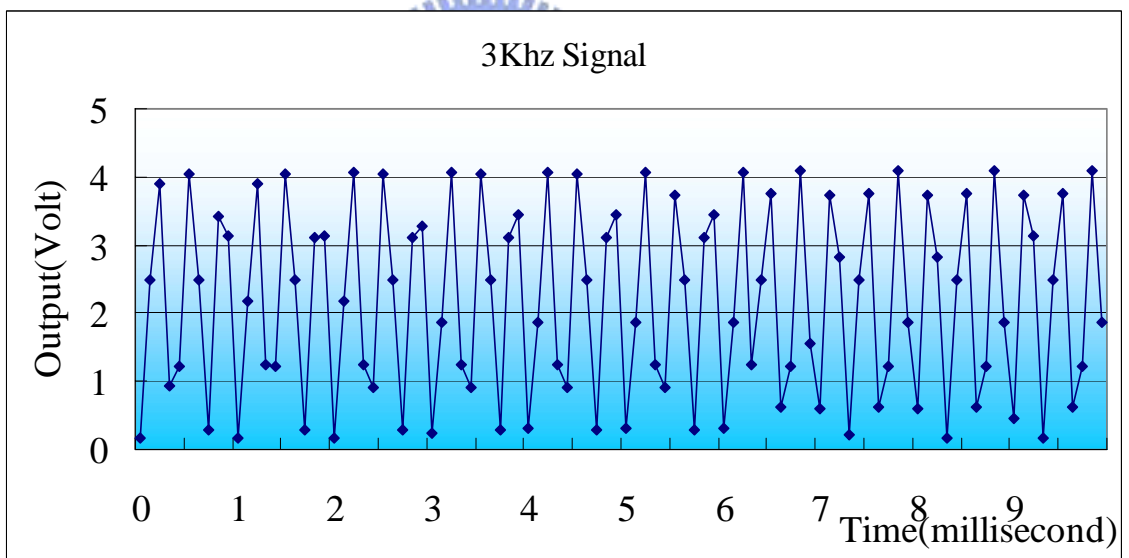


Figure 6-7 3 KHz signal

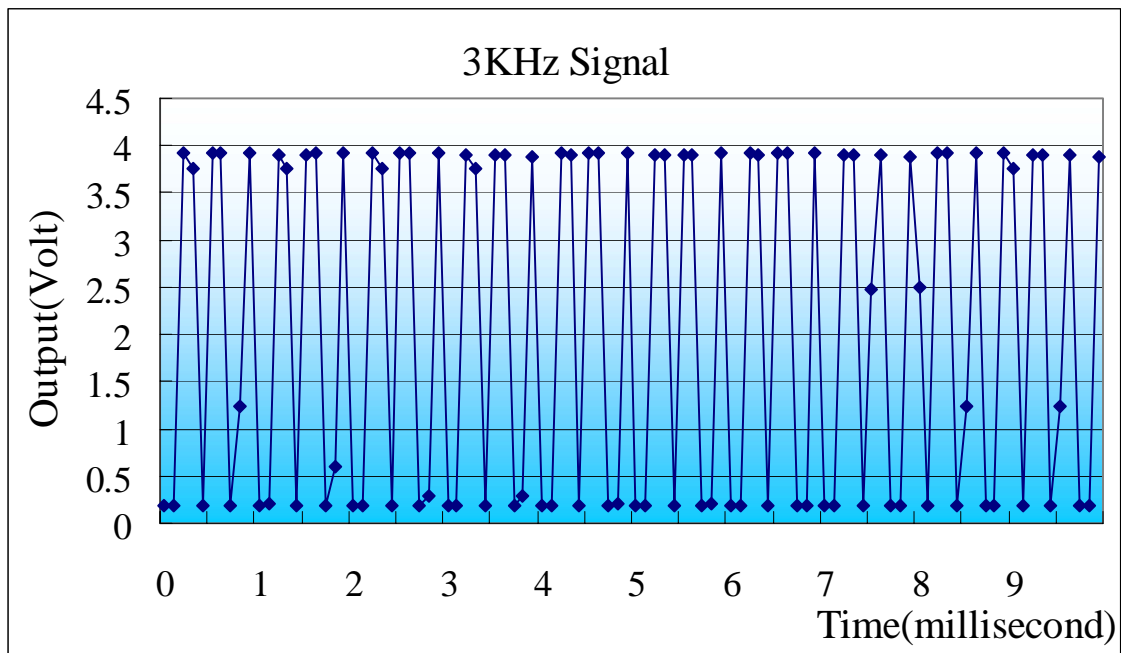


Figure 6-8 3 KHz signal (2)

From the results of my experiment, we can calculate the transfer speed. The transfer speed is approaching 6Kbytes/s. it means that the system can sample 6000 points within a second. Therefore, the input signal frequency must be under 3 KHz. As Figure 6-7 shows, when the input signal frequency is 3 KHz, the output has committed an error.

Furthermore, we found there is a non-continued point in every 64 points (a packet). That is due to the HID report whereas there ought to be a Report ID in each report. In the program setting, we can easy skip this byte, that is, we have skipped this byte while making the graphs.

Digital-to-Analog output result as below figure shows, I let my Windows program to generate a sine wave and capture the photography from oscilloscope.

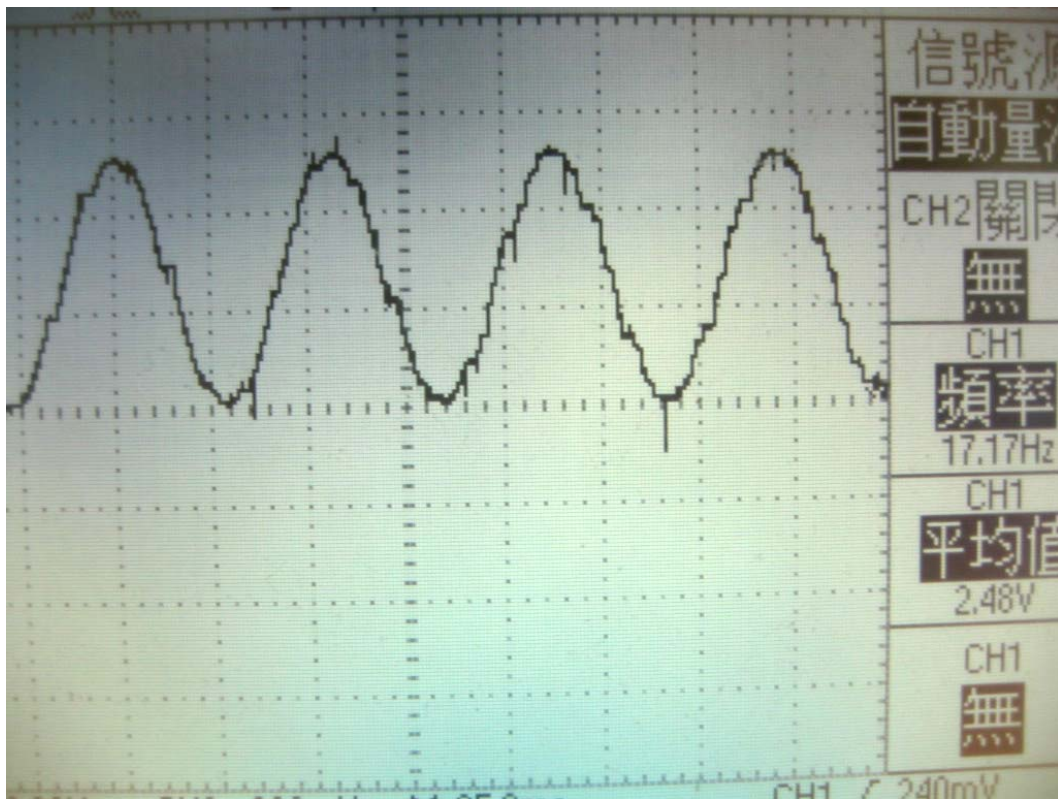


Figure 6-9 Digital-to-analog Output test (1)

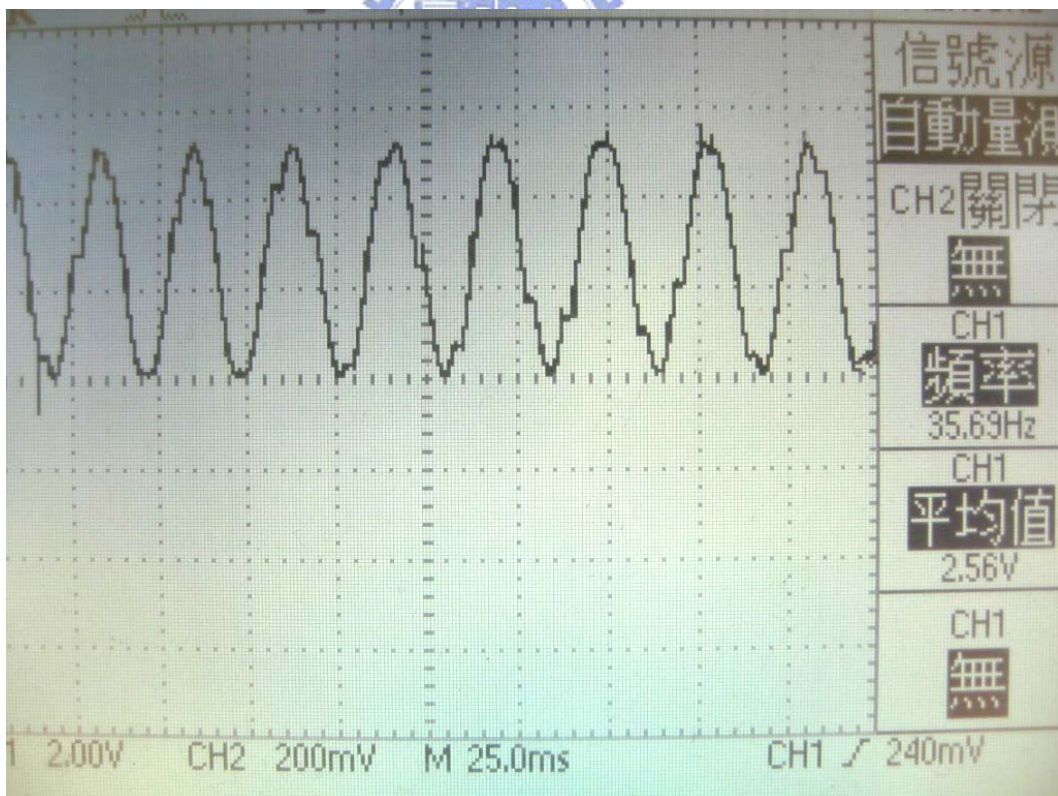


Figure 6-10 Digital-to-analog Output test (2)

## 6.3 Conclusion

In section 6.2, we can comprehend that the transfer speed is approaching 6 Kbytes/s, but the report ID byte will reduce the transfer rate of the useful data. In my experiment, there are 100 packets transferred. There will be 100 bytes for report ID, hence the useful data transferred will be 6400 bytes ( $6500 - 100 = 6400$ ). And the total data transferred into host is 6500 bytes/s.

Constrained by the Windows program and the AD converter's conversion time, the performance can't reach the utility to the best of HID ideal value, 64Kbytes/s. To improve the performance, there are some ways to do. In the software part, we can use multi-thread or DirectX to increase the software performance. In the Hardware part, we can change the AD converter for better performance.

Nevertheless, there is a main problem that the USB is a master-slave system. That is, the device cannot send the data unless the host makes request. In USB, HID class is usually used to implement on moderate device. In data acquisition, it provides an easy-used way, but not a fast sample rate. In the future, the sample rate could be enhanced when the system upgrades to USB 2.0.

## References

- [1] John Hyde, USB design by example: /a practical guide to building I/O devices, Wiley, 2001.
- [2] Don Anderson / Dave Dzatko, Universal Serial Bus System Architecture. Second Edition, Addison Wesley, 2001.
- [3] Jan Axelson, USB Complete: Everything You Need to Develop Custom USB Peripherals, Lakeview Research, 2001.
- [4] 許永和，微處理機程式設計(USB介面之完全解決方案)，長高出版社，2003.
- [5] 蕭世文，精通USB2.0硬體設計，文魁資訊股份有限公司，2002.
- [6] 劉志安，USB2.0程式設計，文魁資訊股份有限公司，2002.
- [7] 楊明豐，8051單晶片C語言設計實務---使用Keil C，2003.
- [8] EZ-USB General Purpose Driver Specification Document, Cypress Semiconductor, 1999.
- [9] EZ-USB Manual Technical Reference Vision 1.10, Cypress Semiconductor, 2002.
- [10] Jan Axelson, "HIDs Up", Embedded Systems Programming (ESP), October 2000.
- [11] USB HID specification, HID-usages tables, and HID descriptor tool,

[www.usb.org/developers/hidpage.html](http://www.usb.org/developers/hidpage.html).

[12][USB specification](http://www.usb.org/developers/docs.html), [www.usb.org/developers/docs.html](http://www.usb.org/developers/docs.html)

[13]Microsoft DDK, <http://msdn.microsoft.com>

