Digital Oscilloscope Adapter unit for a PC

Final Report

BY :

Dursun BARAN / Murat KEBELİ

Ahmet TUTUŞ /Ömer Faruk ÖZDEMİR

Submitted to :

EMO(İstanbul Branch)

Due to :

June 21th, 2007

Principal Investigator :

Prof. Avni MORGÜL

Co-Principal Investigators :

Prof. Ömer Cerid

Asc. Prof. Şenol Mutlu

A report submitted for EE491 (or EE492) senior design project class in partial fulfillment of the requirements for the degree of Bachelor of Science (Department of Electrical and Electronics Engineering) in Boğaziçi University

PC İÇİN SAYISAL OSİLOSKOP ADAPTÖRÜ

Murat KEBELİ

Ahmet TUTUŞ

Ömer Faruk ÖZDEMİR

Dursun BARAN

Prof. Avni MORGÜL

21 Haziran 2007

EMO İstanbul Şubesi

1) GİRİŞ

Projenin en temel amacı analog bir sinyali gerçek zamanlı olarak bilgisayar ortamında görebilmektir. Analog sinyal digital karşılığına çevrildikten sonra USB üzerinden bilgisayara aktarılır ve çizdirme işlemi yapılr.Çizdirme işlemi devamlı ve kullanıcının rahatlıkla takip edebileceği bir şekilde yapılır. Projenin diğer önemli bir amacı aktarılan bilgi paketlerinin bilgisayar ortamında kayıt edilebilmesi ve daha sonra tekrar bilgisayar üzerinde kullanılabilmesidir.

Tasarlanan ürünün hem taşınabilir olması hem de USB' den beslenebilir olması, ürünü önceki ölçüm aletlerinden farklılaştıran iki özelliğidir. Elektronik derslerinde kullandığımız ölçüm aletleri dışardan beslenmesi gereken ve taşınması oldukça güç olan tasarımlardır.

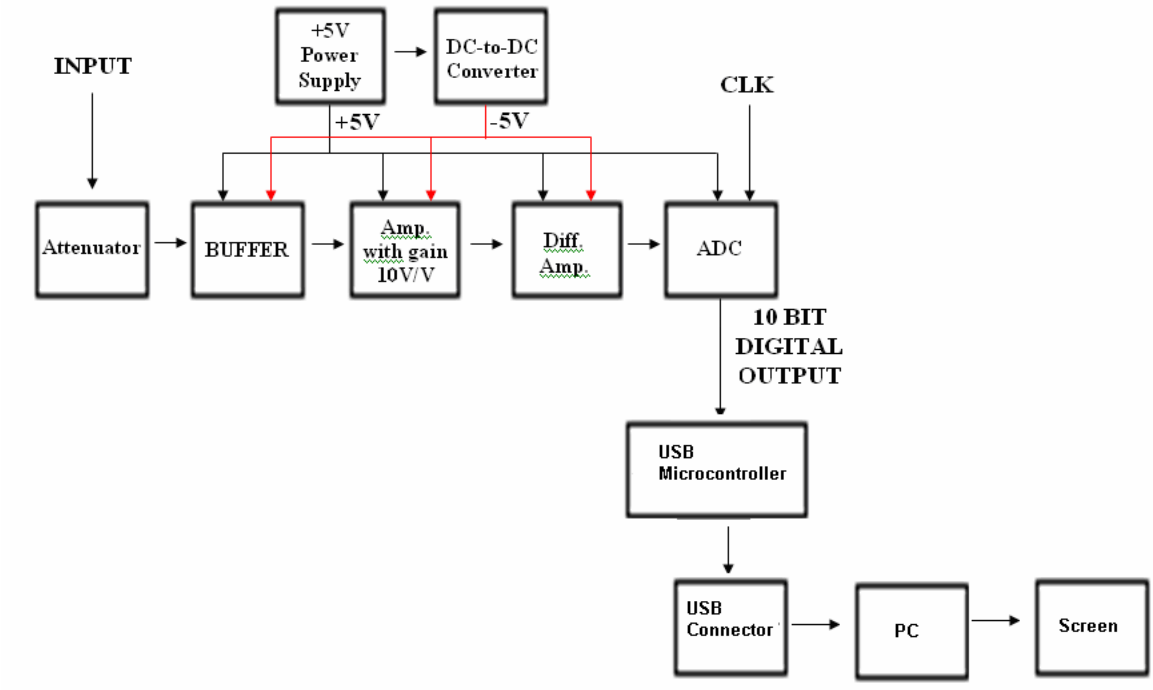
Proje kapsamında kullanıcının rahatlıkla öğrenebileceği ve birçok işlemi rahatlıkla yapabileceği bir kullanıcı arayüzü geliştirildi. Zaman(time) ve büyüklük(magnitude) eksen ölçeği değiştirme, ekrandaki sinyali dondurma ve tekrar çizdirmeye devam etme, aktarılan sinyali aynı zamanda kaydetme ve daha önce kaydedilen sinyali tekrar okuma gibi daha bir çok işlem arayüz sayesinde yapılabilmektedir.

Projenin gerçeklenmesinde ilk olarak, yoğun bir araştırma süreci yaşadık ve bu süreçten sonra blokları belirledik ve her bloğu ayrı ayrı çalıştırdık ve test ettik. Blokların gerçeklenmesi bittikten sonra, blokları birbirine bağladık ve ürünün son halini test ettik. Ürünün problemli taraflarını tespit ettik ve bunları gerekli diğişiklikleri yaparak düzelttik.

Proje gerçeklenmesi sürecinde, gömülü C(embedded C), Eagle and C# programları kullanılmıştır.

2) DONANIM

Giriş bölümde bahsedildiği üzere proje iki ayrı ana koldan oluşmaktadır. Birincisi olan donanım bu bölümde anlatılacaktır. Aşagıdaki figürde donanım blok diagramı verilmiştir.



Giriş sinyalinin büyüklüğünü ADC'ye uygun hale getirmek için sinyal ilk olarak zayıflatıcıya(attenuator) girer. Zayıflatıcı en basit anlamda, sinyalinin büyüklüğünü belli oranda küçültür. Zayıflatıcı, büyüklüğü fazla olan sinyaller için kullanılır.

İkinci basamak olarak, sinyal kazanç katına girer. Bu katın tasarım amacı, büyüklüğü oldukça az olan sinyalleri daha iyi örneklenmesini ve aktarılmasını sağlamaktır. Kazanç katından çıkan sinyal diferansiyel kazanç katına girer. Bu katın kazancı 1'dir ve gelen sinyale DC potansiyel ekler. Devrede kullanılan ADC pozitif kaynaktan beslenmekte ve negatif sinyalleri örneklemek için diffarensiyel katına ihtiyaç duyulmaktadır.

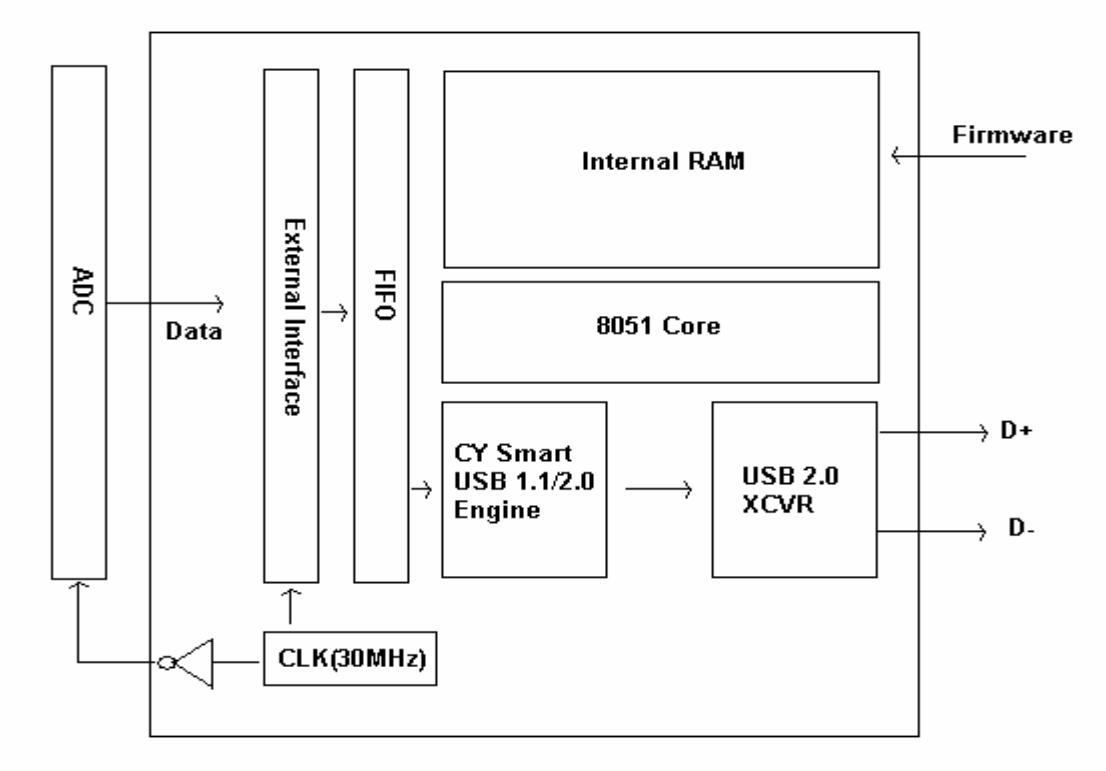
Diferansiyel katın çıktısı ADC'ye girer ve örnekleme işlemi başlar. Örnekleme işlemi clok ile yapılır ve clock dışardan verilir. Diferansiyel kazanç katının çıktısı ADC'nin giriş sinyal sahasına uygun olmalıdır. Bu şart, zayıflatıcının ölçeğini ayarlayarak sağlanabilir. ADC 10 bit sayısal çıktı verir.

ADC'nin çıktısı USB mikrodenetleyiciye girer. ADC'den gelen sayısal çıktılar mikrodenetleyicideki hafızaya(FIFO) yazılır ve burdan paketler halinde bilgisayara USB üzerinden aktarılır. ADC çıktısının hafızaya yazılma işlemi clok ile olur ve bu clock mikrodenetleyici tarafından üretilir. ADC'nin kullandığı clock mikrodenetleyicinin kullandığı clock'un ters çevrilmiş(inverted) halidir. USB mikrodenetleyicinin çıktısı(D+ ve D-) USB bağlayıcı ile bilgisayarın USB port'una bağlanır.

2) FİRMWARE

USB mikrodenetleyicinin hafızasına sayısal çıktıları yazma, yazılan bilgileri kaybetmeden bilgisayara aktarma, bilgisayar ile uygun bir iletişim yolu oluşturma gibi işleri yerine getirmek için USB mikrodenetleyici uygun bir kod ile programlanmalıdır. Bu görevleri yerine getirmek için firmware geliştirildi ve bu kod, ürünü kullanmadan önce USB mikrodenetleyiciye gömülmelidir. Gömülme işlemi

USB üzerinden, geliştirilmiş olan kullanıcı arayüzü ile çok kısa bir sürede yapılabilmektedir. Aşağıda USB kısmın blok digramı verilmiştir.



Geliştirilmiş olan kod, USB üzerinden USB mikrodenetleyicinin RAM'ine yazılır ve kod buradan koşmaya başlar. USB mikrodenetleyicinin hafızasına(FIFO) yazılmış olan sayısal bilgiler 'CY USB 1.1/2.0 Engine' ve 'USB 2.0 XCVR' ile uygun D+ ve D- sinyallerine çevrilir ve bu sinyaller ile bilgi aktarımı sağlanmış olur. Firmware, bu aktarılma işlemini kontrol eder ve gerekli olan yerlerde müdahale eder.

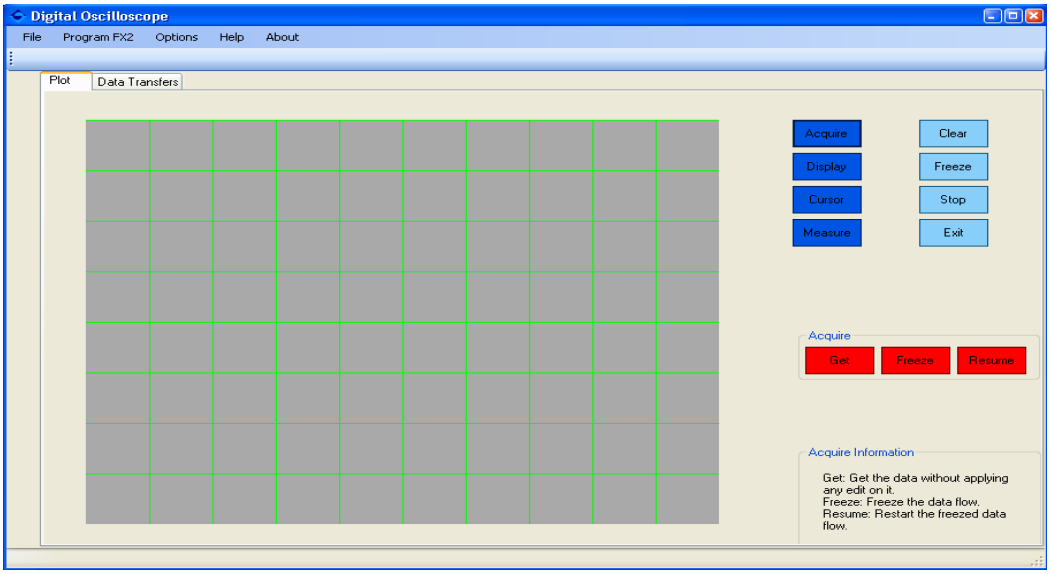
4) YAZILIM

Projenin diğer önemli kolu yazılımdır. Proje kapsamında, uygun bir yazılım geliştirilmiş ve bu yazılım sayesinde USB üzerinden gelen bilgiler okunarak uygun şekilde ekrana bastırılabilmektedir. Yazılım geliştirme ortamı olarak C# dili kullanılmıştır. Donanım ile haberleşmek için uygun bir sürücü dosyası gerekmektedir ve bu dosya USB mikrodenetleyici üreticisi(Cypress Semiconductor) tarafından sağlanmaktadır.

Geliştirilmiş olan kullanıcı arayüzü sayesinde, kullanıcı USB mikrodenetleyiciyi kolayca programlayabilmekte ve ürünü kullanıma hazır hale getirebilmektedir. USB mikrodenetleyiciye gömülecek kod .HEX uzantılı olmalıdır ve uygun .HEX file geliştirilmiş ve yazılım ile birlikte verilmektedir.

Kullanıcı arayüzü öğrenmesi ve kullanması kolay olacak şekilde tasarlanmış ve aynı zamanda fonksiyonel olmasına dikkat edilmiştir. Kullanıcı gerçek zamanlı olarak sinyali ekranda görebilmekte ve istediği zaman ekranı dondurabilmektedir. Dondurma işleminden sonra istediği zaman gerçek zamanlı aktarma işlemine devam edebilmektedir. Büyüklük ve time ekseni en uygun olacak şekilde kullanıcı tarafından ölçeklendirilebilmektedir.

Gerçek zamanlı aktarılan sinyal bilgisayara kaydedilebilmekte ve daha sonra bu sinyal tekrar arayüz sayesinde kullanılabilmektedir. Kayıt yapılacak yer kullanıcı tarafından belirlenir ve kullanıcı durdurana kadar kayıt işlemi devam eder. Kayıt edilen bilgi paketi sayısı kullanıcıya bildirilir. Kayıt edilmiş bir sinyal, kullanıcı arayüzünü kullanarak açılabilmekte ve gerçek zamanlı bir şekilde çizdirilmektedir. Gerçek zamanlı aktarma işleminde kullanılabilen bütün fonksiyonlar açma işleminde de kullanılabilmektedir.



5) SONUÇ

Projenin sağlayacağı en önemli faydalardan bir tanesi, ölçmek istenilen bir sinyal bilgisayarda görülebilecek ve gelen sinyal üzerinde istenilen değişiklik kolayca yapılabilecektir. Ölçülen sinyalin birçok özelliği otomatik olarak bilgisayar tarafından hesaplanabilecek ve bu sayede hesaplama hataları büyük ölçüde aşılabilecek.

Diğer bir önemli faydası, ölçülen sinyalin bilgisayar ortamında kaydedilebilmesidir. Bu sayede, kaydedilen bir sinyal daha sonra tekrar açılarak çizdirebilmektedir. Bilgisayar ortamında bilgi aktarımı oldukça kolay bir işlem olduğundan, kaydedilen bilgiler kolayca farklı ortamlara taşınabilmekte ve farklı yerlerde kullanılabilmektedir. Taşıma işlemi sırasında herhangi bir bilgi kaybı olmadığından çizdirme kalitesi düşmemektedir.

Ürünün kolayca taşınabilir ve USB üzerinden beslenebilir olması, ürünün kullanılması kolay kılan özelliklerdendir. Bu sayede, ölçüm işleri kolayca ve kısa sürede sonuçlandırılabilmektedir.

Ürünün son hali test edildi ve başarılı sonuçlar elde edildi. Sinyal üreticisi ile sinüs,üçgen ve kare dalga elde edildi ve bu sinyaller, kullanıcı arayüzü kullanılarak çizdirildi. Aynı sinyaller analog osiloskop ile de ölçüldü ve elde edilen çizimler karşılaştırldı. Çizimlerin yaklaşık aynı olduğu ama bilgisayardan elde edilen çizimler üzerinde küçük karıncalanmalar olduğu tespit edildi. Bu karıncalanmaların devreden kaynaklanan gürültülerden olduğu anlaşıldı.

Giriş sinyalinin frekansı ve büyüklüğü değiştirilerek, bilgisayar üzerindeki değişimler gözlendi. Ürünün tepki süresi, kabul edilebilir sınırlar içerisinde olduğu belirlendi. Uygulanan sinyallere ofset verilerek, bilgisayar üzerindeki değişim gözlendi ve doğru sonuçlara ulaşıldı.

ACKNOWLEDGEMENTS

This special project which is given by the department of Electrical and Electronics Engineering aims to increase the experience of the students. First, it is a necessity to state the desire and diligence of each group member to the project. Each group member spends a large amount of efforts and time on this project and therefore, we thank all group members for their continuous contribution to the project. All barriers that we have encountered overcame by means of the involvement of the group members and it is clear to comprehend what each member of the group worked for the sake of the success of the project.

First, we appreciate the endless contribution of the principal investigator Prof. Avni Morgül to the project. He was always available to ask questions about the project and he has an incredible amount of the experience about the schematic drawings and PCB preparation process. Our project necessitates a complex layout and we could not overcome the problems about the PCB without the support of Prof. Avni Morgül.

We thank to the co-principal investigator Prof. Ömer Cerid for his supports about the code writing and circuit schematic drawings. He is an expert about the microcontroller programming and the circuit construction and it is unbelievable chance for us to become a member of this project. He has provided insights about the solution of the problems to us, and his suggestions always responded our questions. We also appreciate the help of Mr. Cerid in choosing the components of our hardware.

We think that it is important to point that the component support of some IC producers are pleased us. Maxim-IC, Cypress, Analog Devices and Texas-Instruments are our IC suppliers and none of them want any payment for their support.

Finally, we want to thank to Prof. Şenol Mutlu for his help about the documentation and report format. He always supported us about which way we should follow to overcome the barriers. He has always found enough time and effort for our questions.

ABSTRACT

This project aims to solve the problem of the reading and the storing of an analog signal. As we know from our electronic courses, the reading an analog signal is a quite important problem. In order to solve the problem of reading and the storing of the analog signal, we try to design an adapter unit that is capable of the transferring of analog data to computer in digital form via USB interface. In addition to data transfer, our design includes adjustments of the received data.

The approach that we have followed is first search, then implement and finally test the device. Therefore, we conducted a big amount of research about the project and then, we tried to construct the blocks of the implementation and finally we constructed the blocks. After this step, we tried to connect each blocks in order to complete the untested implementation. Eventually, we test the device to see faulty sides of the implementation and makes changes if necessary.

In the implementation of the blocks, we have made use of some the programs namely embedded-C(C18), layout program(Eagle) and object oriented program (C#). The outputs of each block were measured by an analog oscilloscope and a debugger.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

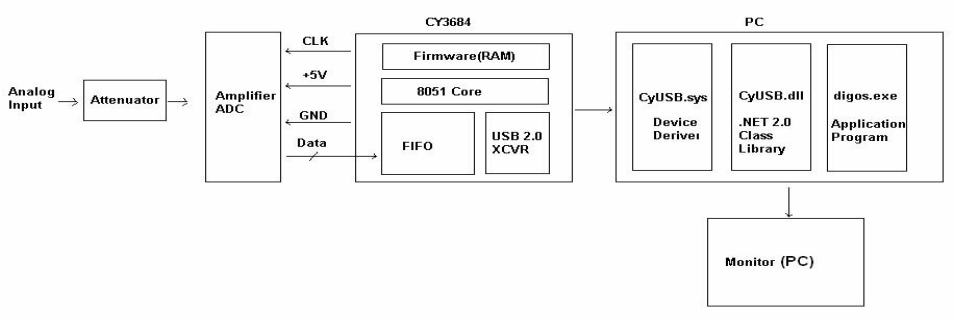
Appendices

INTRODUCTION

This project aims to solve the problem of the reading and the storing of an analog signal. As we know from our electronic courses, the reading an analog signal is a quite important problem. In order to solve the problem of reading and the storing of the analog signal, we try to design an adapter unit that is capable of the transferring of analog data to computer in digital form via USB interface. In addition to data transfer, our design includes adjustments of the received data.

The approach that we have followed is first search, then implement and finally test the device. In last semester, we conducted a big amount of research about the project and then, we tried to construct the blocks of the implementation and finally we constructed the blocks. After this step, we tried to connect each blocks in order to complete the untested implementation. After the implementation of all blocks, we made all tests of our circuit last semester and now, we are aware of the impossibility of implementing all of the circuit without any mistake or error. Therefore, this semester we changed our problem solving approach. We began to implement independently each block of the circuit and test each block independent of other blocks. This method provides us with finding the source of the problem easily if it occurs, and changing it without affecting other parts of the circuit. Eventually, we test the each block of our circuit to see faulty sides of the implementation and makes changes if necessary.

In the implementation of the blocks, we have made use of some the programs namely C# (object oriented programming language) and layout program (Eagle). Overall Flow Chart of the signals and system is below:



CHAPTER 1: HARDWARE IMPLEMENTATION

1.1. OVERALL WIEV OF HARDWARE

In the first semester, the circuit we designed didn't work although we made lots of changes on it. Therefore, we thought we should change it before it is too late. The problems we have investigated in that circuit were the amplifier and ADC. The amplifier was giving different results when we fed it from a different source (We mean with a different source, in fact, inputs having different resistances). This is actually a result of lack of a buffer. We, then, thought that we should use a buffer in order input not to change for different sources. The problem with ADC was more severe but, we could never make it work. After a detailed analysis of ADC's on the internet, we found that most of ADCs need amplifiers in order to work properly. With amplifiers, we add common mode voltages to ADCs because again most of ADCs do not convert negative voltages. Their ranges are generally defined positive, e.g., 0-2V. So, because we want to convert analog voltages in the range between -1 and +1 V, we should add at least +1V (It depends on other things also.) common mode voltage to incoming voltage. With these two big problems, it seemed to us, we should create a new PCB by taking into consideration these problems and solutions we found for those problems.

We designed a new circuit with 5 main blocks: An attenuator, a buffer, an amplifier for gain, a DC-to-DC converter, a differential amplifier in order to add a common mode voltage to ADC, an ADC.

The block diagram of new circuit is shown below.

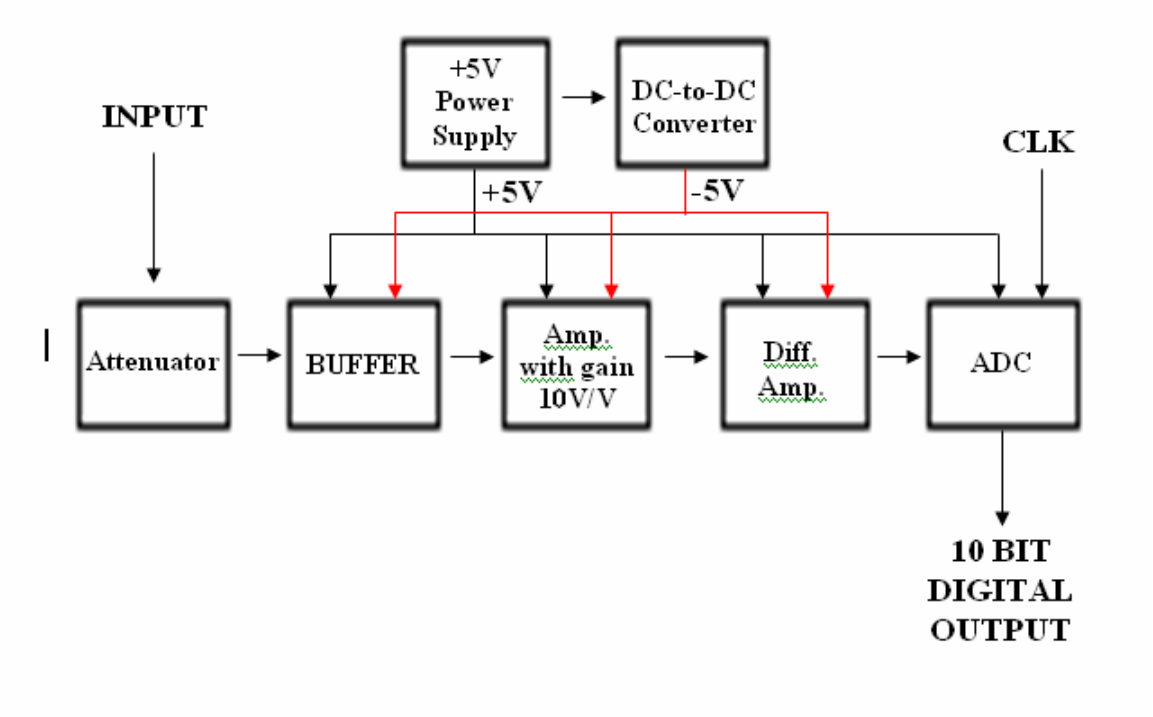


Figure 1.1.1: Overall Block Diagram of hardware implementation.

Up to now, we mentioned about why we needed to design a new circuit for our project and how we implemented this circuit. Now we are going to analyze the main blocks of new circuit in detail.

1.2. DC-TO-DC CONVERTER

In real life, we don't have negative power supplies. Therefore, we have to convert positive power supply to negative, because most of our components work with dual supplies. For this purposes, we used MAX660 converter of Maxim Integrated Products which is used to convert +5V to -5V.

Below are the characteristics of this IC.

- 0.65V Typical Loss at 100mA Load
- Low 120µA Operating Current
- 6.5Ω Typical Output Impedance
- Guaranteed R_{OUT} < 15 Ω for C1 = C2 = 10µF
- Inverts or Doubles Input Supply Voltage
- Selectable Oscillator Frequency: 10kHz/80kHz
- 88% Typical Conversion Efficiency at 100mA (IL to GND)

As is seen from its characteristics it is a very suitable IC for our purposes. Below are its pin diagram and configuration for +5V to -5V conversion.

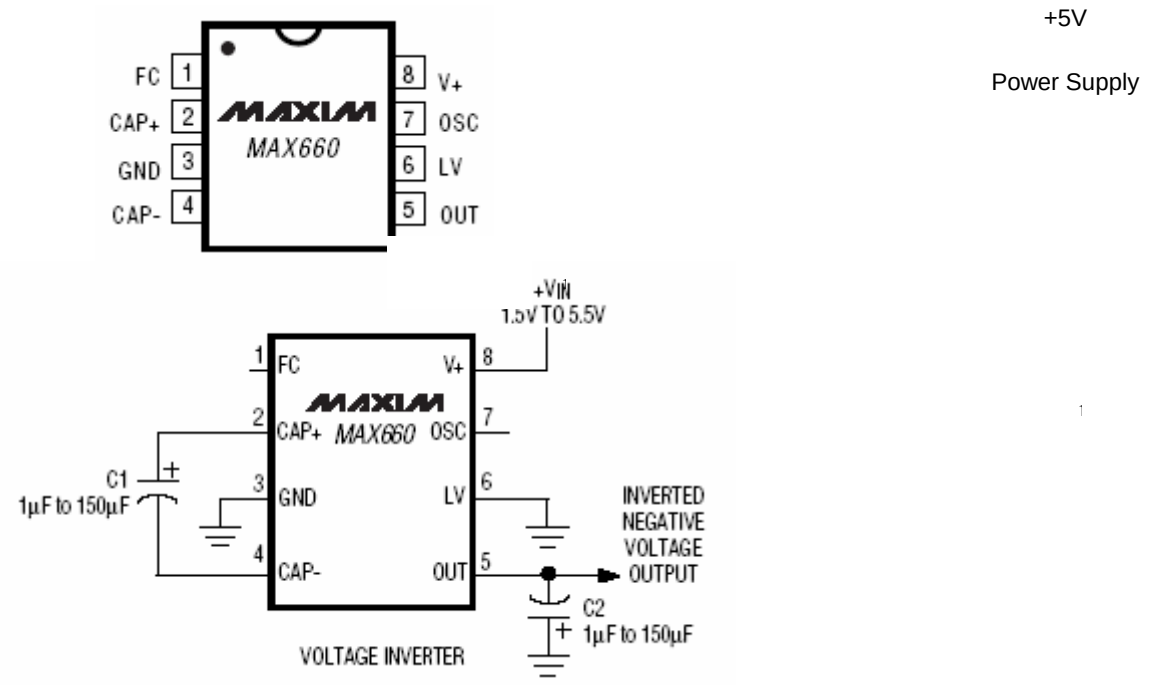


Figure 1.2.1: Pin Diagram of MAX660 and its configuration for +5V to -5V conversion.

Although we used it to convert +5V to -5V, MAX660 voltage inverter can convert a +1.5V to +5.5V input to a corresponding -1.5V to -5.5V output. With a typical operating current of only 120µA provides ideal performance for board level voltage conversion applications. MAX660 can also double the output voltage of an input power supply.

A frequency control (FC) pin which we didn't use selects either 10kHz or 80kHz (40kHz min) operation to optimize capacitor size and quiescent current. The oscillator frequency can also be adjusted with an external capacitor or driven with an external clock.

As we stated in its characteristics, depending on the load, the error of the output voltage of MAX660 changes. Therefore, we thought it is better to measure what it gives actually. Below is the result.

$V_{out} = 4.80V$

1.3. BUFFER

As we discussed before, a buffer to the input of the circuit is necessary because of loading effect. We used an OPAMP in voltage follower mode. The Op-Amp we used is OPA656 IC of Texas Instruments.

Below are the characteristics of this IC.

- 500MHz Unity-Gain Bandwidth
- Low Input Bias Current: 2pA
- Low Offset and Drift: ±0.25mV, ±2μV/ºC
- Low Distortion: 74dB SFDR at 5MHz
- High Output Current: 70mA
- Low Input Voltage Noise: 7nV/√Hz

It is a very suitable Op-Amp for our circuit. (High bandwidth, low input bias current, offset voltage and distortion, etc.) Below is its pin diagram and configuration for a unity gain buffer.

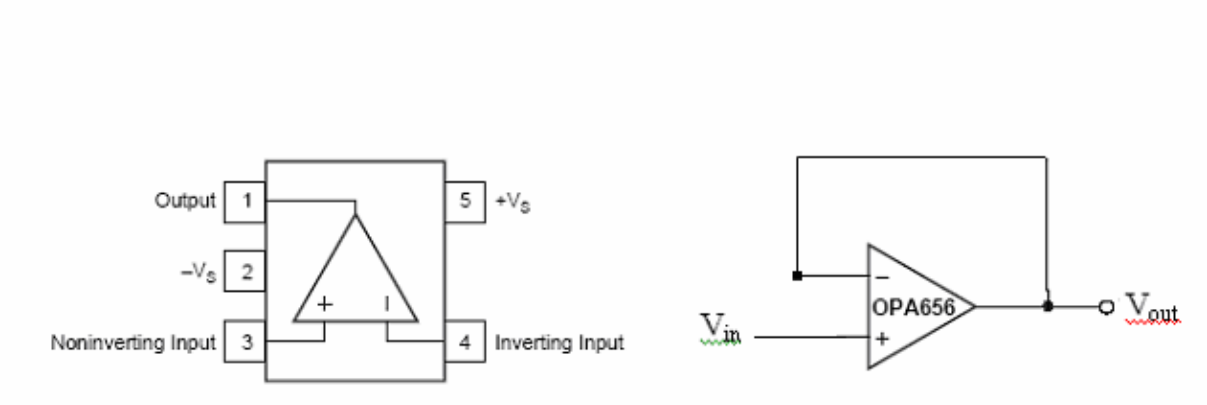


Figure 1.3.1: Pin Diagram of OPA656 and Its Configuration for a Unity-Gain Buffer.

The OPA656 combines a very wideband, unity-gain stable, voltage-feedback Op-Amp with a FET-input stage to offer an ultra high dynamic-range amplifier. Extremely low DC errors give good precision in applications. The high unity-gain stable bandwidth and JFET input allows exceptional performance in high-speed. The high input impedance and low bias current provided by the FET input is supported by the ultra-low 7nV/√Hz input voltage noise to achieve a very low integrated noise in wideband applications. Broad transimpedance bandwidths are achievable given the OPA656's high 230MHz gain bandwidth product.

1.4. AMPLIFIER WITH A GAIN

This part is needed because of 2 reasons. First reason is to amplify small signals. In order to display small signals in the μV range more accurately we have to amplify them. Other reason is that we need a LP filter to prevent aliasing in the circuit. As we will see in ADC part, ADC can work up to 30Msps. Therefore, signals with frequency of greater than 15MHz will cause aliasing according to well-known sampling theorem. Consequently, in this amplifier, we implemented both a gain of 10V/V and a LP filter of 20MHz cut-off frequency. The amplifier we designed is a multiple feedback LP filter which is a commonly used amplifier.

To implement this amplifier, we used OPA657 Op-Amp of Texas Instruments. It is almost the same as OPA656 which we used to implement a buffer.

Below are the characteristics of this IC.

- High Gain Bandwidth Product: 1.6GHz
- High Bandwidth: 275MHz (G = +10V/V)
- Low Input Offset Voltage: ±0.25mV
- Low Input Bias Current: 2pA
- Low Input Voltage Noise: 4.8nV/√Hz
- High Output Current: 70mA
- Fast Overdrive Recovery

It is a very suitable Op-Amp for our circuit. (High bandwidth, low input bias current, offset voltage and distortion, etc.) Below is its pin diagram and configuration for a LP filter of gain 10V/V.

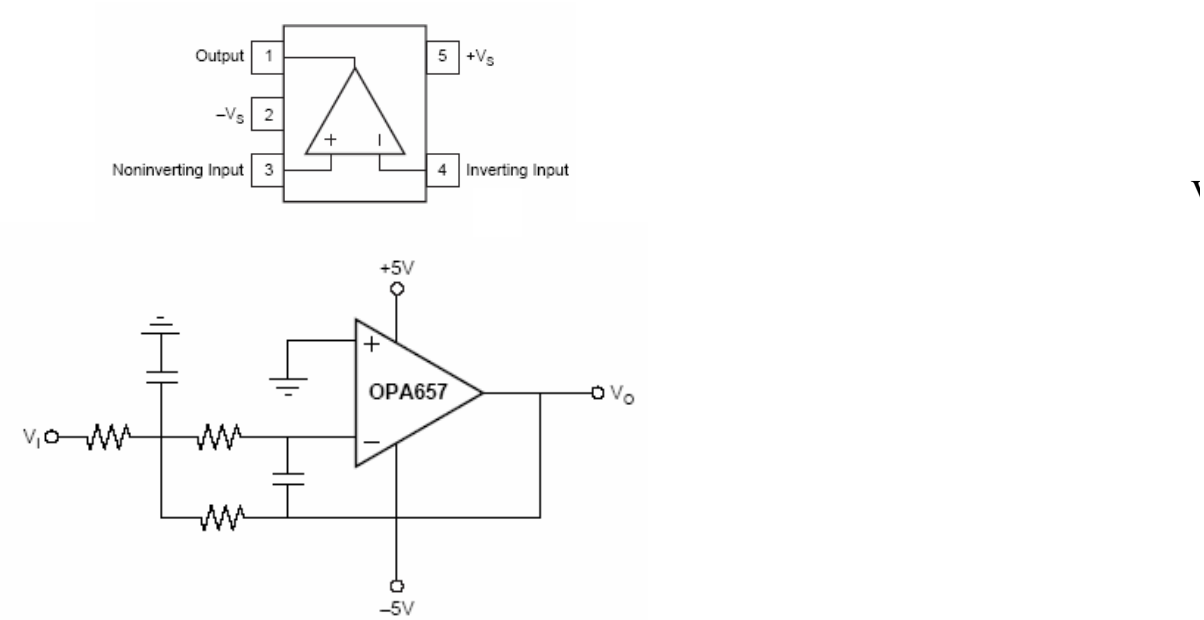


Figure 1.4.1: Pin Diagram of OPA657 and Its Configuration for a LP Filter of Gain 10V/V.

The OPA657 combines a high gain bandwidth, low distortion, voltage-feedback op amp with a low voltage noise JFET-input stage to offer a very high dynamic range amplifier

for high precision applications. Very low level signals can be significantly amplified in a single OPA657 gain stage with exceptional bandwidth and accuracy. Having a high 1.6GHz gain bandwidth product will give > 10MHz signal bandwidths up to gains of 160V/V (44dB). The very low input bias current and capacitance will support this performance even for relatively high source impedances. The JFET input contributes virtually no current noise while for broadband applications; a low voltage noise is also required. The low 4.8nV/ Hz input voltage noise will provide exceptional input sensitivity for higher bandwidth applications.

Having these very special characteristics, we configured this Op-Amp as a LP filter of gain 10V/V. Below are equations for this configuration and component values.

H_{0LP}= - R_3/R_1,

$f_0= 1/2\pi(R_2R_3C_1C_2)^{0.5}$,

$Q= (C_1C_2)^{0.5}/ [(R_2R_3/R_1^2)^{0.5} + (R_3/R_2)^{0.5} + (R_2/R_3)^{0.5}]$,

C_1= 10pF,

C_2= 10pF,

R_1= 100Ω,

R_2= 1kΩ,

R_3= 1kΩ,

According to these component values, gain and f_0 of filter is found below.

H_{0LP}= - 1k/100=-10V/V,

$f_0= 1/2\pi(1kx1kx10px10p)^{0.5}$= 15.92MHz,

1.5. DIFFERENTIAL AMPLIFIER

We mentioned before why we needed a differential amplifier. We used THS4141 differential amplifier of Texas Instruments. Luckily, there is an application of this IC as an ADC driver in its datasheet.

Below are the characteristics of this IC.

- High Performance
  - 160 MHz –3 dB Bandwidth (VCC = ±15 V)
  - 450 V/μs Slew Rate
  - -79 dB, Third Harmonic Distortion at 1 MHz
  - 6.5 nV/√Hz Input-Referred Noise
- Differential Input/Differential Output
  - Balanced Outputs Reject Common-Mode Noise
  - Reduced Second Harmonic Distortion Due to Differential Output
- Wide Power Supply Range
  - VCC = 5 V Single Supply to ±15 V Dual Supply
- ICC(SD) = 880 μA in Shutdown Mode

The THS4141 is one in a family of fully differential input/differential output devices fabricated using complementary bipolar process. The THS4141 is made of a true fully-differential signal path from input to output. This design leads to an excellent common-mode noise rejection and improved total harmonic distortion which an essential part of our project.

Below is its pin diagram and configuration to drive an ADC.

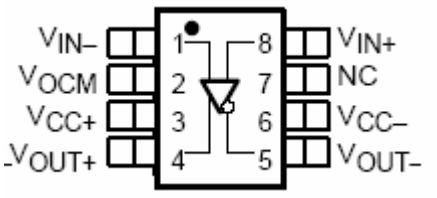


Figure 1.5.1: THS4141 Pin Diagram.

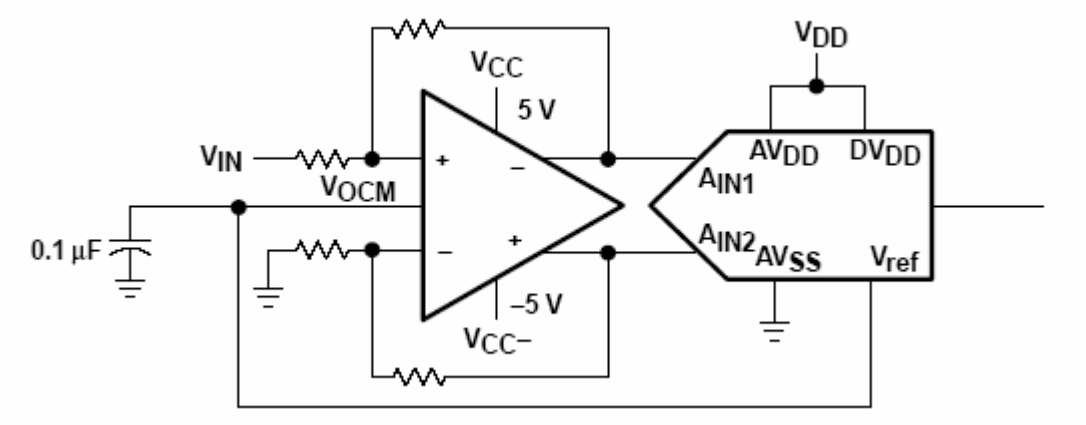


Figure 1.5.2: THS4141 Configuration To Drive an ADC.

The THS4141 can also work with a single power supply, but circuit diagram for that configuration is more complex. Therefore we used it with dual power supplies.

If nothing is connected to its V_{OCM} pin, it is automatically set to arithmetic mean of power supplies.

0.1µF capacitor is a bypass capacitor for stability. In addition, if we choose 4 resistances equal, the gain of this amplifier will be 1V/V. Therefore, we used 1kΩ resistances for those 4 resistances.

1.6. ANALOG TO DIGITAL CONVERTER

After all steps up to here, the signal is now ready to be converted. We used THS1030 ADC of Texas Instruments.

Below are the characteristics of this IC.

- 10-Bit Resolution, 30 MSPS Analog-to-Digital Converter
- Configurable Input: Single-Ended or Differential
- Differential Nonlinearity: ±0.3 LSB
- Signal-to-Noise: 57 dB
- Spurious Free Dynamic Range: 60 dB
- Adjustable Internal Voltage Reference
- Out-of-Range Indicator
- Power-Down Mode

The THS1030 is a CMOS, low-power, 10-bit, 30 MSPS ADC that can operate with a supply range from 3 V to 5.5 V. The THS1030 has been designed to give circuit developers flexibility. The analog input to the THS1030 can be either single-ended or differential. The THS1030 provides a wide selection of voltage references to match the user's design requirements. For more design flexibility, the internal reference can be bypassed to use an external reference to suit the dc accuracy and temperature drift requirements of the application. The out-of-range output is used to monitor any out-of-range condition in THS1030's input range. The THS1030 is characterized for operation from 0°C to 70°C.

Below is its pin diagram.

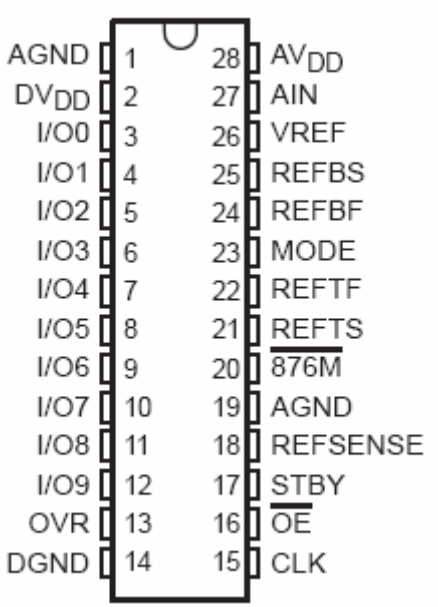


Figure 1.6.1: Pin Diagram of THS1030 ADC.

TERMINAL NAME	NO.	I/O	DESCRIPTION
AGND	1, 19	I	Analog ground
AIN	27	I	Analog input
AV_{DD}	28	I	Analog supply
CLK	15	I	Clock input
DGND	14	I	Digital ground
DV_{DD}	2	I	Digital driver supply
I/O0	3	O	Digital I/O bit 0 (LSB)
I/O1	4		Digital I/O bit 1
I/O2	5		Digital I/O bit 2
I/O3	6		Digital I/O bit 3
I/O4	7		Digital I/O bit 4
I/O5	8		Digital I/O bit 5
I/O6	9		Digital I/O bit 6
I/O7	10		Digital I/O bit 7
I/O8	11		Digital I/O bit 8
I/O9	12		Digital I/O bit 9 (MSB)
MODE	23	I	Mode input
$\overline{OE}$	16	I	High to 3-state the data bus, low to enable the data bus
OVR	13	O	Out-of-range indicator
REFBS	25	I	Reference bottom sense
REFBF	24	I	Reference bottom decoupling
REFSENSE	18	I	Reference sense
REFTF	22	I	Reference top decoupling
REFTS	21	I	Reference top sense
STBY	17	I	High = power-down mode, low = normal operation mode
VREF	26	I/O	Internal and external reference
$\overline{876M}$	20	I	High = THS1030 mode, low = TLC876 mode (see section 4 for TLC876 mode)

Figure 1.6.2: Pin Descriptions of THS1030

1.7. PRINCIPLES OF OPERATION

The analog input A_{IN} is sampled in the sample and hold unit which is shown in the figure below, the output of which feeds the ADC core, where the process of analog to digital conversion is performed against ADC reference voltages, REFTF and REFBF.

Connecting the MODE pin to one of three voltages, AGND, AVDD or AVDD/2 sets up operating configurations. The three settings open or close internal switches to select one of the three basic methods of ADC reference generation. Depending on the user's choice of operating configuration, the ADC reference voltages may come from the internal reference buffer or may be fed from completely external sources. Where the reference buffer is employed, the user can choose to drive it from the onboard reference generator (ORG), or may use an external voltage source. A specific configuration is selected by connections to the REFSENSE, VREF, REFTS and REFBS, and REFTF and REFBF pins, along with any external voltage sources selected by the user.

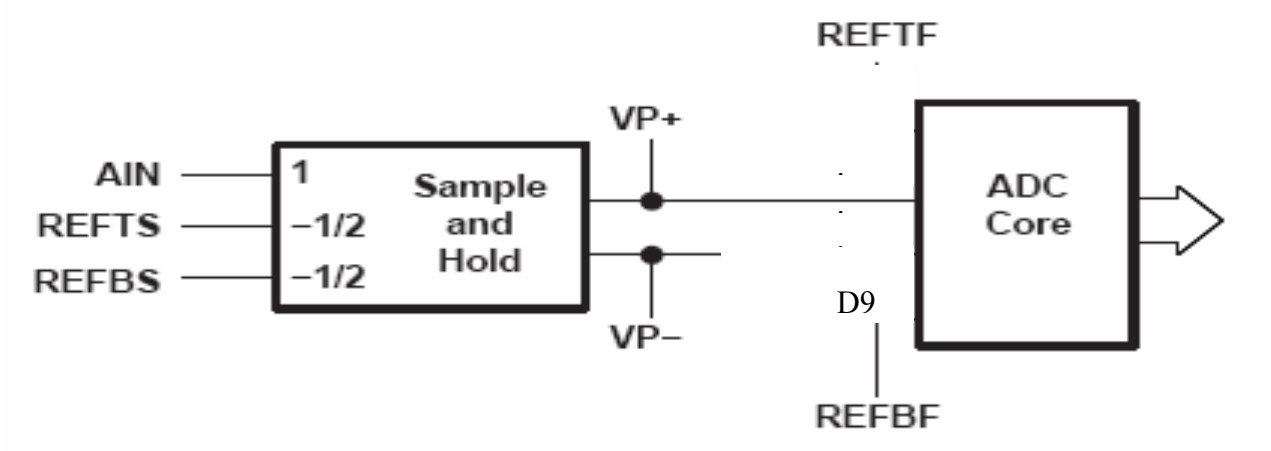


Figure 1.7.1: Analog Input Signal Flow.

The ADC core drives out through output buffers to the data pins D0 (LSB) to D9 (MSB). The output buffers can be disabled by the OE pin. A single, sample-rate clock (30 MHz maximum) is required at pin CLK. The analog input signal is sampled on the rising edge of CLK, and corresponding data is output after following third rising edge which is shown below.

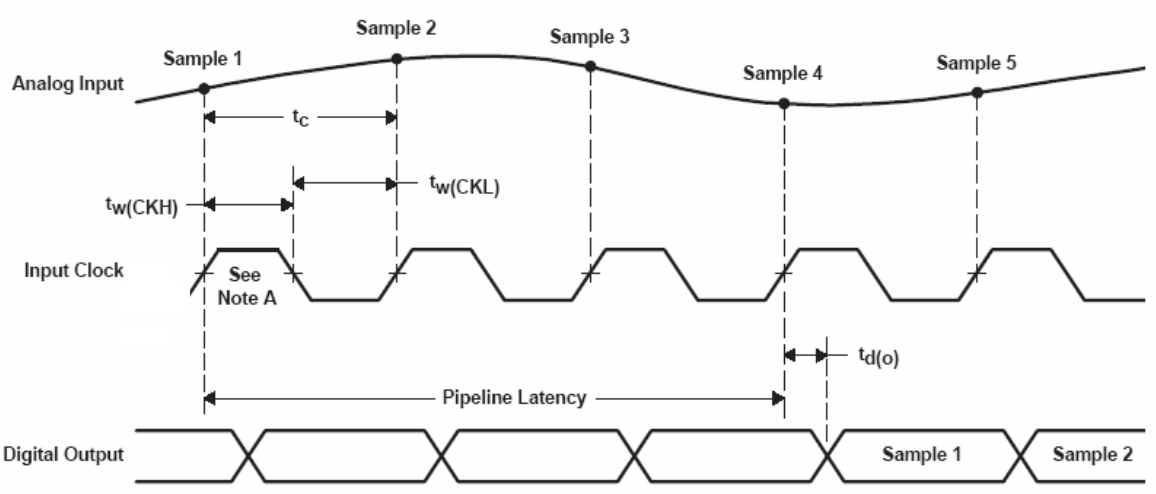


Figure 1.7.2: Operation of THS1030

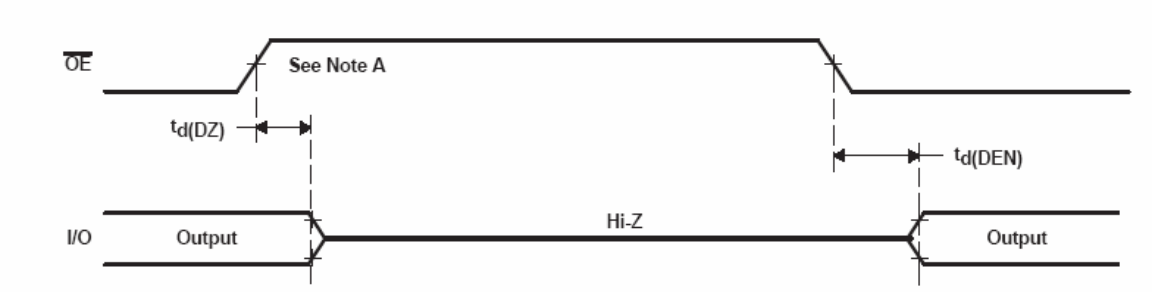


Figure 1.7.3: Output Enable Timing Diagram

The STBY pin controls the THS1030 power down. The user-chosen operating configuration and reference voltages determine what input signal voltage range the THS1030 can handle.

The analog input signal AIN is applied to the AIN pin, either dc-coupled or ac-coupled. The differential sample and hold processes AIN with respect to the voltages applied to the REFTS and REFBS pins, to give a differential output VP+ − VP− = VP given by:

$$VP = A_{IN} - VM$$

$$VM = \frac{(REFTS + REFBS)}{2}$$

For single-ended input signals, VM is a constant voltage; usually the AIN mid-scale input voltage. However if MODE = AVDD/2 then REFTS and REFBS can be connected together to operate with AIN as a complementary pair of differential inputs. In all operating configurations, VP is digitized against ADC reference voltages REFTF and REFBF, full-scale values of VP being given by:

$$VPFS + = \frac{+ (REFTF - REFBF)}{2}$$

$$VPFS - = \frac{- (REFTF - REFBF)}{2}$$

VP voltages outside the range VPFS− to VPFS+ lie outside the conversion range of the ADC. Attempts to convert out-of-range inputs are signaled to the application by driving the OVR output pin high. VP voltages less than VPFS− give ADC output code 0. VP voltages greater than VPFS+ give output code 1023.

Combining the above equations, the analog full scale input voltages at AIN pin which give VPFS+ and VPFS− at the sample and hold output are:

$$A_{IN} = FS+ = VM + \frac{(REFTF - REFBF)}{2}$$

and

$$A_{IN} = FS- = VM - \frac{(REFTF - REFBF)}{2}$$

The analog input span (voltage range) that lies within the ADC conversion range is:

$$\text{Input span} = [(FS+) - (FS-)] = (REFTF - REFBF)$$

Therefore REFTF and REFBF voltage difference sets the device input range.

The THS1030 has three primary modes of ADC reference generation, selected by the voltage level applied to the MODE pin. Connecting the MODE pin to AGND gives full external reference mode. In this mode, the user supplies the ADC reference voltages directly to pins REFTF and REFBF. This mode is used where there is need for minimum power drain or where there are very tight tolerances on the ADC reference voltages. Only single-ended input is possible in this mode. Connecting the MODE pin to AVDD/2 gives differential mode. In this mode, the ADC reference voltages REFTF and REFBF are generated by the internal reference buffer from the voltage applied to the VREF pin. This mode is suitable for handling differentially presented inputs, which are applied to the AIN and REFTS/REFBS pins. Connecting the MODE pin to AVDD gives top/bottom mode. In this mode, the ADC reference voltages REFTF and REFBF are generated by the internal reference buffer from the voltages applied to the REFTS and REFBS pins. Only single-ended input is possible in

top/bottom mode. When MODE is connected to AGND, the internal reference buffer is powered down, its inputs and outputs disconnected, and REFTS and REFBS internally connected to REFTF and REFBF respectively. These nodes are connected by the user to external sources to provide the ADC reference voltages.

REFERENCE MODE	MODE	REFSENSE	VREF VOLTAGE	REFTS, REFBS	ANALOG INPUT
External	AGND	AV_{DD}	Disabled	Reference buffer powered down, reference voltage provided directly by REFT and REFB	Single-ended
Internal	$AV_{DD}/2$	VREF	1 V	Externally connect REFTS to REFBS. This pair then forms AIN– to the ADC.	Differential
		AGND	2 V		
		External divider	1 + Ra/Rb		
External (through internal reference buffer)	AV_{DD}	AV_{DD}	Disabled	REFTS = V_{FS+} REFBS = V_{FS-}	Single-ended (top-bottom mode)
Output of VREF can be externally tied to REFTS or REFBS to provide one of the reference voltages		VREF	1 V		
		AGND	2 V		
		External divider	1 + Ra/Rb		

Figure 1.7.4: The Summary of Modes of Operation of THS1030.

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
Differential input voltage (REFTF – REFBF) (REFSENSE = VREF)			0.9	1	1.1	V
Differential input voltage (REFTF – REFBF) (REFSENSE = AGND)			1.9	2	2.1	V
Input common mode voltage (REFTF + REFBF)/2		AV_{DD} = 3 V	1.3	1.5	1.7	V
		AV_{DD} = 5 V	2	2.5	3	
REFTF voltage (MODE = AV_{DD})	VREF = 1 V	AV_{DD} = 3 V		2		V
		AV_{DD} = 5 V		3		
	VREF = 2 V	AV_{DD} = 3 V		2.5		V
		AV_{DD} = 5 V		3.5		
REFBF voltage (MODE = AV_{DD})	VREF = 1 V	AV_{DD} = 3 V		1		V
		AV_{DD} = 5 V		2		
	VREF = 2 V	AV_{DD} = 3 V		0.5		V
		AV_{DD} = 5 V		1.5		
Input resistance between REFTF and REFBF				600		Ω
Power up time for valid ADC conversions (t_{PUconv})		See Note 1		1.2		µs

Figure 1.7.5: REFTF and REFBF Reference Voltages of THS1030.

We used THS1030 in differential mode, therefore we should examine this mode in detail.

When MODE = AVDD/2, the internal reference buffer is enabled, its outputs internally switched to REFTF and REFBF and inputs internally switched to VREF and AGND. The REFTF and REFBF voltages are centered on AVDD/2 by the internal reference

buffer and the voltage difference between REFTF and REFBF equals the voltage at VREF. The internal REFTS to REFBS and REFTF to REFBF switches are open in this mode, allowing REFTS and REFBS to form the AIN− to the sample and hold. Depending on the connection of the REFSENSE pin, the voltage on VREF may be externally driven, or set to an internally generated voltage of 1V, 2V, or an intermediate voltage. Below is the configuration for 2V reference voltage which is used in our circuit.

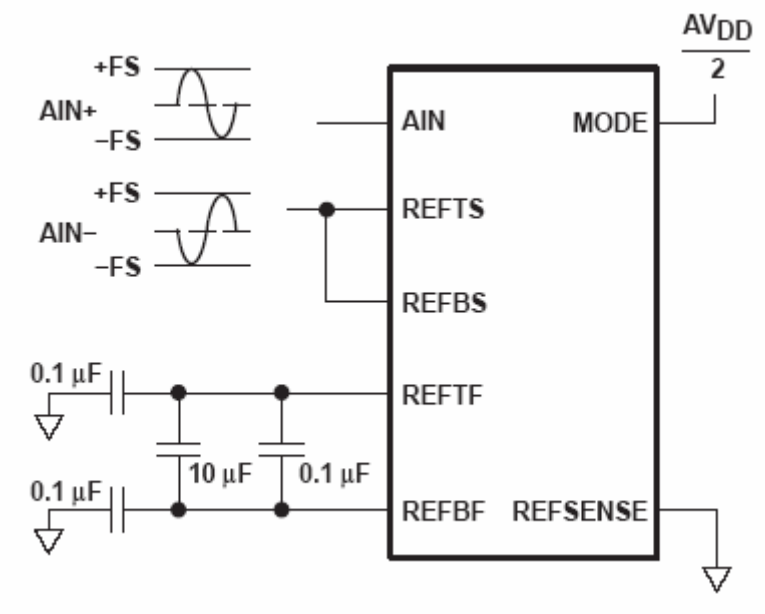


Figure 1.7.6: Differential Mode 2V Reference Voltage Configuration of THS1030.

Now, we are ready to specify operating values of THS 1030 based on figures and formulas we gave up to now.

From figure 13, we see that (REFTF-REFBF) = 2V in differential mode and when REFSENSE = GND. Therefore, VPFS = ±(REFTF-REFBF)/2=±1V.

In summary,

MODE= AVDD/2,

REFSENSE= 0V,

VREF=2V,

(REFTF-REFBF)=2V,

VPFS=±1V,

As operating frequency increases, some of the output bits of ADC changes randomly. Therefore, we have to take into account this for a precise measurement. Below is the graph taken from datasheet of THS1030 which shows the relationship between frequency and effective number of bits.

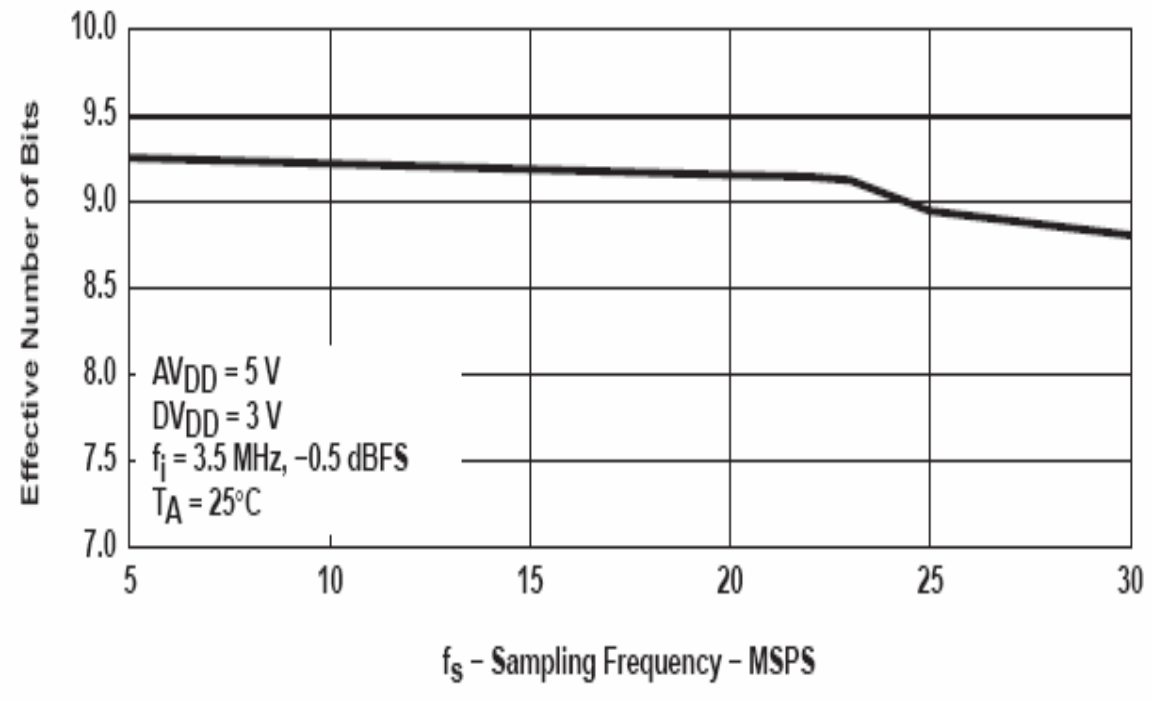


Figure 1.7.7: Effective number of bits of Sampling Frequency Graph.

As is seen from, for full scale speed of conversion of ADC, effective number of bits is around 8.75.

To show whether incoming input signal is in the range or not, we connected a LED to OVR pin with a 1kΩ resistance. If applied signal is not in the range, OVR pin will be high and LED will glow. This will be a warning to the user, implying to switch the attenuator to a higher or a lower range.

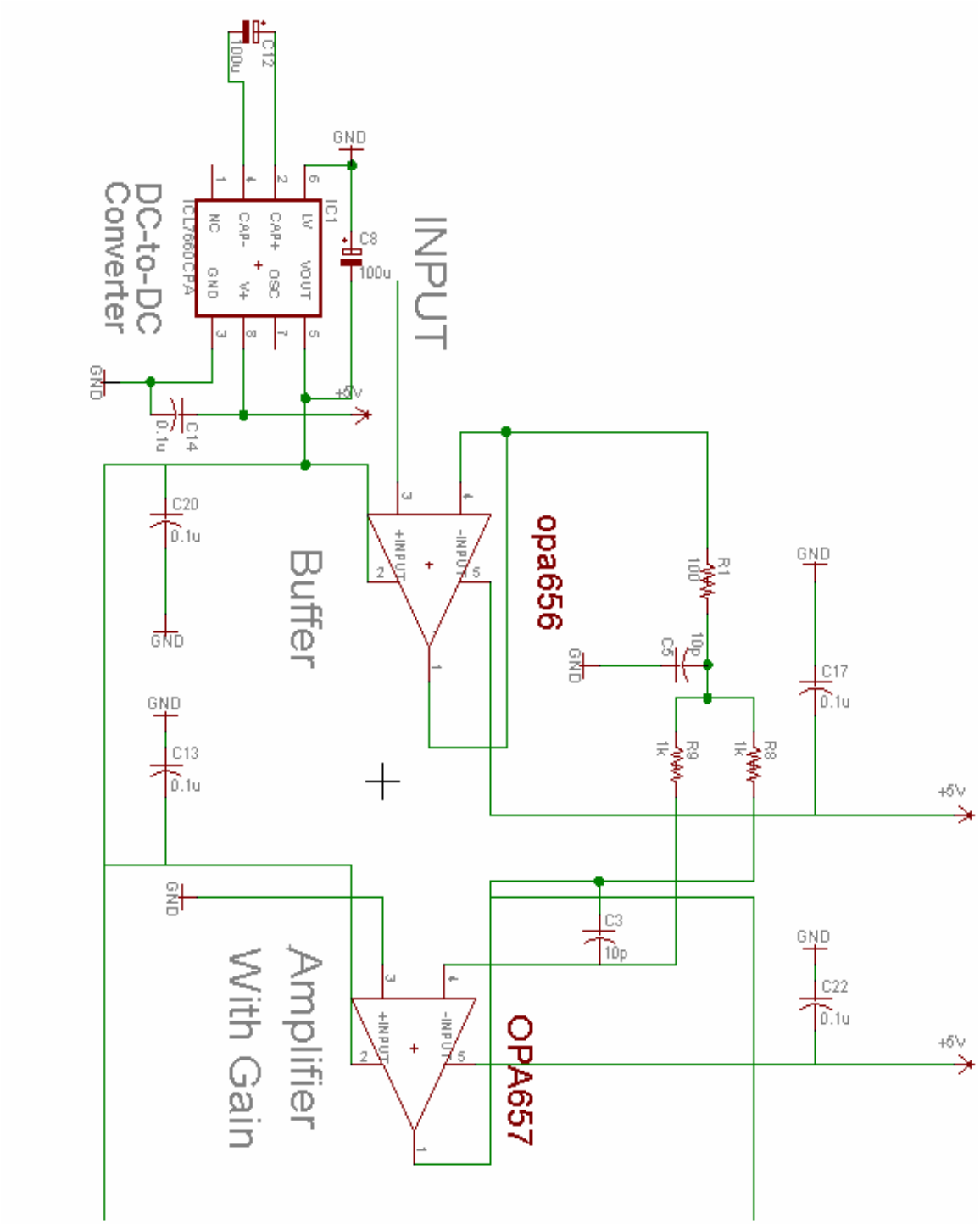
Final Circuit Diagram

We examined the main blocks of the circuit in detail, therefore we are now ready to connect these parts. In fact, once we know the parts separately, it is very straight forward to connect them.

We added all positive and negative power supply pins of ICs' 0.1μF capacitor in order to smooth DC voltage applied. In addition, as is recommended in datasheets, we connected a common 6.8μF capacitor to power supplies. These capacitors are for smoothing purposes.

In order to make MODE = $V_{DD}/2$, we used two 1kΩ resistances as a voltage divider one of them is connected to V_{DD} and the other is connected to GND.

In the successive two pages, the final circuit diagram drawn in Eagle program is represented:


DC-to-DC
Converter
TCL7660UCPA
IC1
C12
100u
C8
100u
C14
0.1u
C20
0.1u
C13
0.1u
INPUT
Buffer
opa656
+INPUT
-INPUT
R1
100
C5
10p
R9
1k
R8
1k
C17
0.1u
C3
10p
C22
0.1u
Amplifier
With Gain
OPA657
GND
+5v

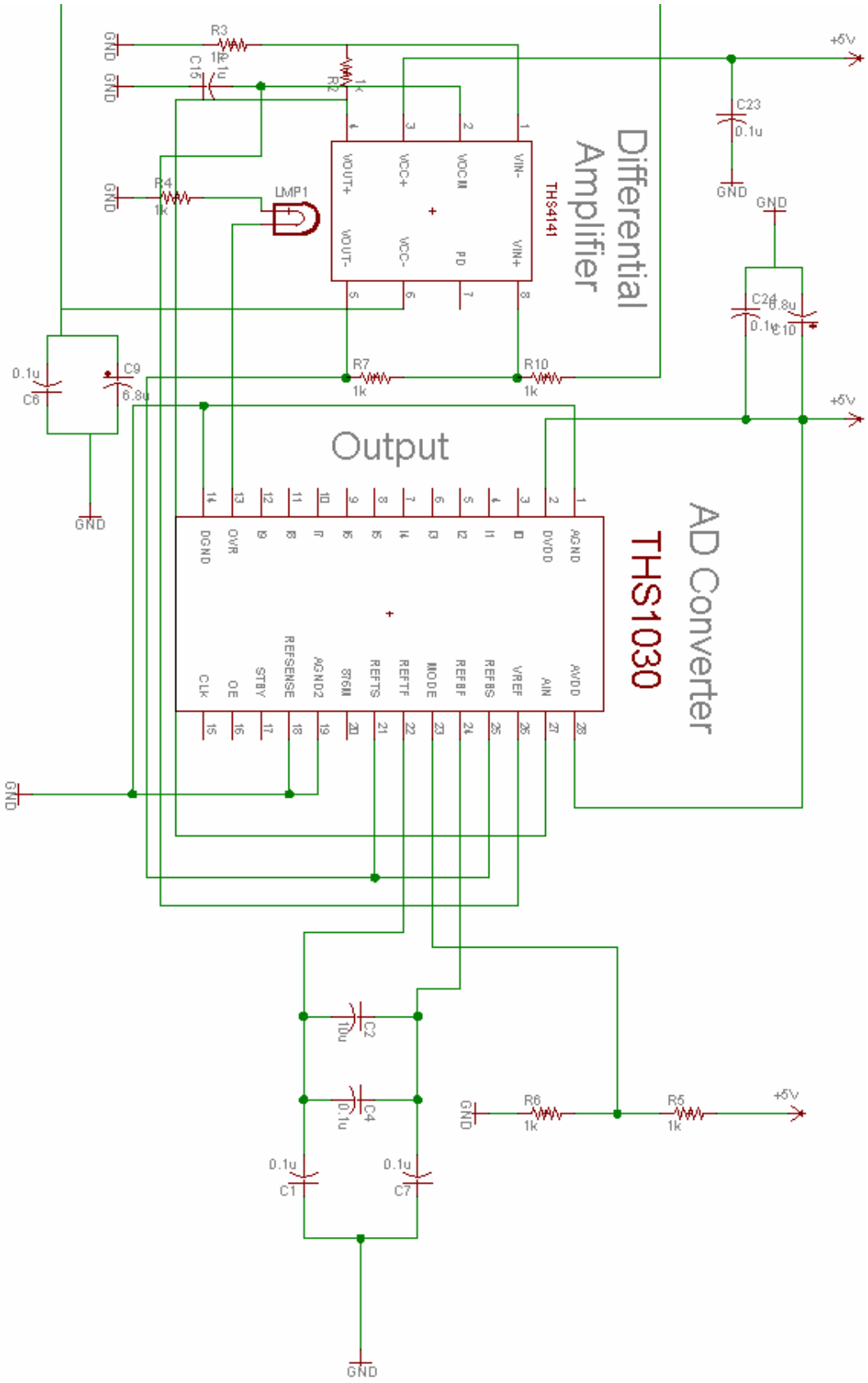


Figure 1.7.8: Circuit Diagram.

1.8. PCB LAYOUT

After we drew the circuit diagram, now we should prepare the PCB layout of the circuit. We designed PCB layout of it in the Eagle layout program which is a very commonly used layout program. Therefore, a little summary about this program will be useful.

For PCB layout section we have done lots of works. First of all, we learned a new program called “Eagle 4.16”. For learning purposes, we have read its tutorial. After a little bit practice, it was a very easy and useful program to use. It has a very large library which includes some of ICs we used such as MAX660. But to create a footprint is not a difficult task also. With Eagle, new footprints can be created very easily. First of all, you open a new library page and then enter the name for new component. Then, you draw pin diagram of the IC. Afterwards, you draw the PCB layout of the IC. Finally, pins of these two are connected. Eagle also has an auto router feature, it tries to connect all of the circuit itself. With proper positioning of components in the circuit it is able to connect as 90 or 95 % of the circuit. If we again think about our very complicated circuit, it is a useful feature for us. Moreover, Eagle program converts PCB layout into a general format called “Gerber Format” which is used in most of board printing companies.

After a little bit about Eagle, we will continue with PCB considerations which are taken into account when we are designing PCB layout of the circuit. Some of them are basic principles of designing layout and some of them are written in the datasheets of the components.

- DC supply and ground wires in the PCB should be as large as possible. This is a must for a good grounding and supplying of the circuit.

- The empty areas in the PCB should be filled with ground areas for a good grounding of the circuit.

- 45 degree-wires should be preferred where possible in the layout.

- The overall PCB drawing should be neat and clear.

- In the areas of the amplifier inputs and output, the ground plane can be removed to minimize the stray capacitance.

- Use a 6.8-μF tantalum capacitor in parallel with a 0.1-μF ceramic capacitor on each supply terminal. In addition, the 0.1μF capacitor should be placed as close as possible to the supply terminal.

- The circuit layout should be made as compact as possible, thereby minimizing the length of all trace runs. Particular attention should be paid to the inverting input of the amplifier. Its length should be kept as short as possible.

- Using surface-mount passive components is recommended for high frequency amplifier circuits for several reasons. First, because of the extremely low lead inductance of surface-mount components, the problem with stray series inductance is greatly reduced. Second, the small size of surface-mount components naturally leads to a more compact layout thereby minimizing both stray inductance and capacitance. If leaded components are used, it is recommended that the lead lengths be kept as short as possible.

- For ADC, voltages on AIN, REFTF and REFBF and REFTS and REFBS must all be inside the supply rails.

- Minimize the distance (< 0.25") from the power-supply pins to high-frequency 0.1uF decoupling capacitors.

- Never use wirewound type resistors in a high frequency application. Use smd or metal film resistors. Since the output pin and inverting input pin are the most sensitive to parasitic capacitance, always position the feedback and series output resistor, if any, as close as possible to the output pin.

According to these considerations, we draw our PCB layout of the circuit as is shown

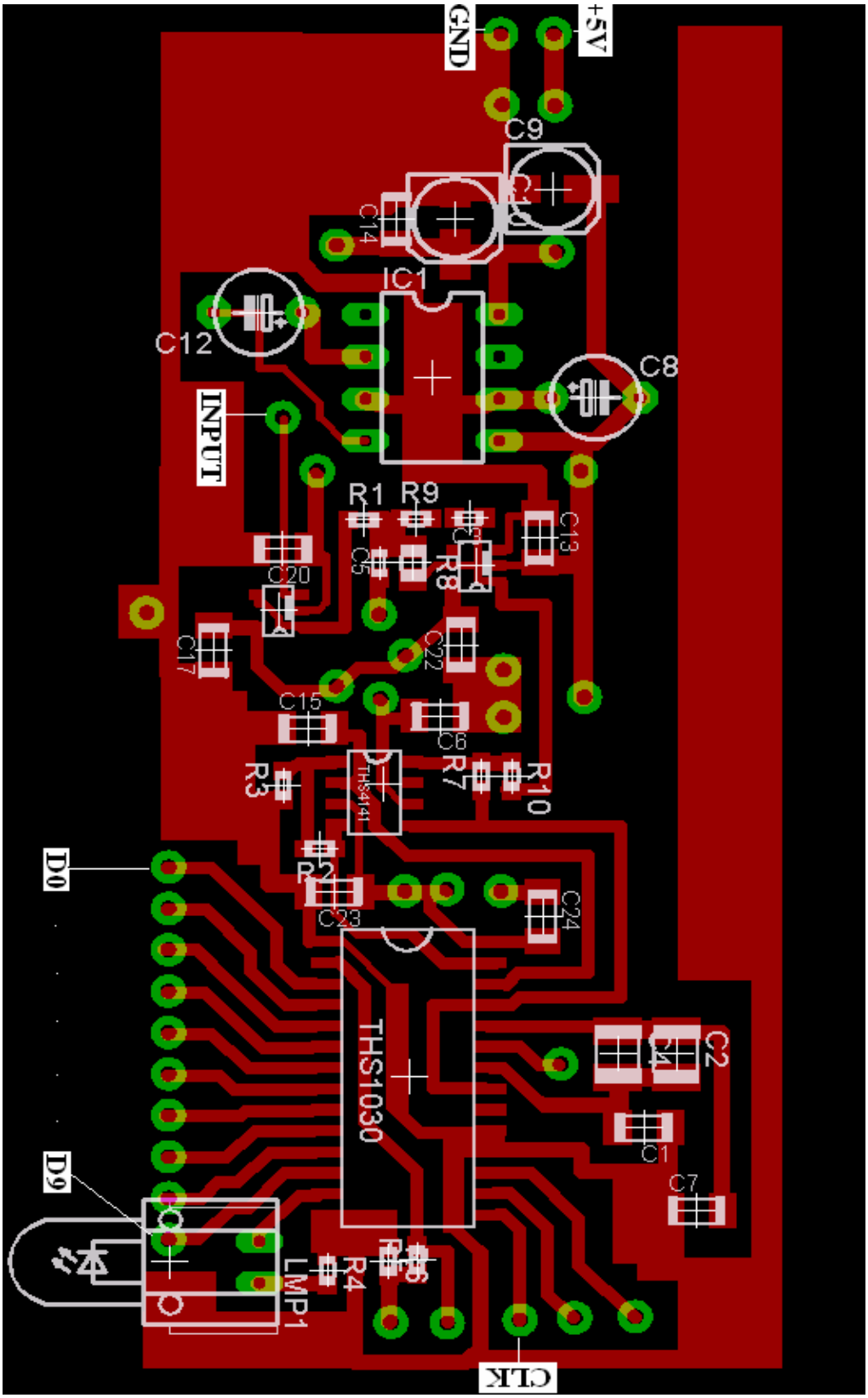


Figure 1.8.1: PCB Layout of Circuit.

1.9. TEST RESULTS

Before we start, we should emphasize that the hardware part is working quite well except OVR pin of ADC. Although we thought very much about it, we couldn't find why it is not working. It does not become high although input we applied is out of range. But, remaining part is functioning quite well.

First of all, we tested -3dB frequency of the circuit and obtained following graph.

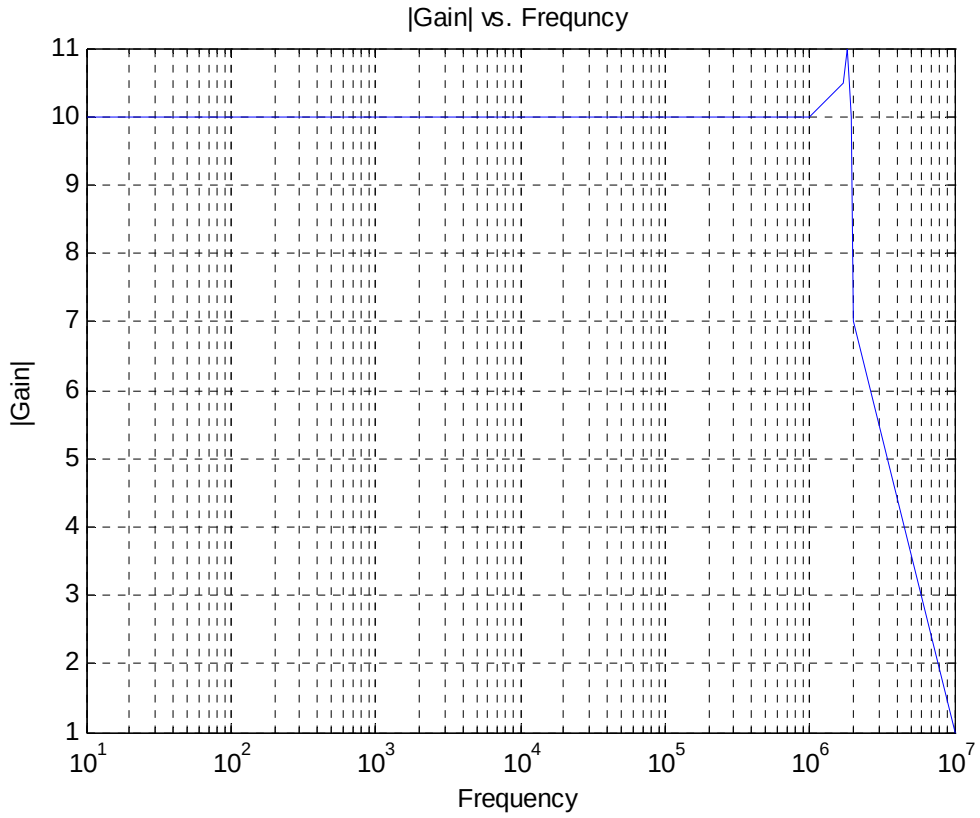


Figure 1.9.1: -3dB Frequency Graph.

Now, we should give some information about digital outputs of ADC. For signals above 1V, it gives all bits as 0. For signals less than -1V, it gives all bits 1. For values between these voltages 4 LSB bits changes continuously, and remaining bits change according to large variations in the input signal.

CHAPTER 2: USB FIRMWARE

As stated in the project objectives, the data comes from the output of the ADC should be trasferred to PC over USB. As being easily guessed, the amount of the data processed is huge and it is a clear and important point to consider that whether USB can support this high speed transfer or not. In order to answer this critical question, the specification of the USB will be mentioned first and then making an easy calculation to verify the answer.

2.1. GENERAL USB SPECIFICATIONS

Universal Serial Bus (USB) is a serial bus standard to interface devices. The USB standard uses the NRZI system to encode data. USB signals are transmitted on a twisted pair of data cables, labelled D+ and D−. These collectively use half-duplex differential signaling to combat the effects of electromagnetic noise on longer lines. D+ and D− usually operate together; they are not separate simplex connections. Transmitted signal levels are 0.0–0.3 volts for low and 2.8–3.6 volts for high **[1].**

The USB specification provides a 5 V (volts) supply on a single wire from which connected USB devices may draw power. The specification provides for no more than 5.25 V and no less than 4.35 V between the +ve and -ve bus power lines. Initially, a device is only allowed to draw 100 mA. It may request more current from the upstream device in units of 100 mA up to a maximum of 500 mA**[1].** The power is supplied to circuit should be +5V and the current consumption is not more than 100 mA and so, the circuit can be fed from the USB and without any battery.

The knowledge above is not complex and in the design, it has not be necessary to think about these electrical specifications because of the fact that a proper USB chip is used and it can do all the things needed for a successful transfer. It means that all decoding and voltage level adjustments are made in a single chip and in the project CY7C68013A is used. This is a USB 2.0 compliant, high speed USB peripheral and discussed next.

2.2 USB 2.0 VERSUS USB 1.1:

A core team from Compaq, Hewlett Packard, Intel, Lucent, Microsoft, NEC and Philips is leading the development of the USB Specification, version 2.0, that will increase data throughput by a factor of 40. This backwards-compatible extension of the USB 1.1 specification uses the same cables, connectors and software interfaces so the user will see no change in the usage model. They will, however, benefit from an additional range of higher performance peripherals, such as video-conferencing cameras, next-generation scanners and printers, and fast storage devices, with the same ease-of-use features as today's USB peripherals **[3].**

The main difference between USB 2.0 and USB 1.1 is the data transfer rates.

2.3 DATA BANDWIDTH

To return to main question after some brief information about USB, the supported data rates by USB should be dealt. In theory, it is clear and the followings are the available data bandwidth of USB:

- A **Low Speed** rate of up to 1.5 Mbit/s (187.5 kB/s) that is mostly used for Human Interface Devices (HID) such as keyboards, mice, and joysticks**[1].**
- A **Full Speed(USB 1.1)** rate of up to 12 Mbit/s (1.5 MB/s). Full Speed was the fastest rate before the USB 2.0 specification and many devices fall back to Full Speed. Full Speed devices divide the USB bandwidth between them in a first-come first-served basis and it is not uncommon to run out of bandwidth with several isochronous devices. All USB Hubs support Full Speed **[1].**

- A **Hi-Speed(USB 2.0)** rate of up to 480 Mbit/s (60 MB/s) **[1].**

As seen above, USB 2.0 can support up to 480Mbits/s and this is a huge amount of data. This is available in theory and there are lots of limiting factors that decrease the data throughput rate of a USB system.

The maximum rate currently (2006) attained with real devices is about half of the full theoretical (60 MB/s) data throughput rate. Most hi-speed USB devices typically operate at much slower speeds, often about 3 MB/s overall, sometimes up to 10-20 MB/s **[1].**

The necessary data bandwidth for the design can be calculated as follows:

BW=(ADC conversion frequency)*(ADC resolution) = [30*10^6)]*10=300Mbits/s

The clock of used ADC is 30MHz and the resolution is 10 bits. The data throughput rate is sufficient for our design but, in practice, this is not attainable as stated above. At the same time, the EZ-USB chip can support only 8bits or 16 bits external interface and that is to say if you make use of 10 bits of ADC, you have use 16bits interface and every 6bits for each 16 bits are discarded and this also increases the needed speed without any gain. The least two significant bit of ADC can not give sufficiently accurate results. Therefore, we determined to ignore these LSB bits and only make use of MSB 8 bits and the external interface is set to 8 bits width. After making the changes, the necessary bandwidth is the following:

BW=(ADC conversion frequency)*(ADC resolution) = [30*10^6)]*8=240Mbits/s.

Again, this value is high. However, we can not decrease the necessary bandwidth anymore and therefore, by implementing a good firmware and an application program, we try to increase the speed of the data transfer to the needed level. Actually, the speed depends on the configuration of the interface and endpoints of EZ-USB chip that will be discussed next. Eventually, we can say that High Speed USB 2.0 has to be used in the project and the final device should be used in a computer has USB High Speed Host Controller.

2.4 ENUMERATION

Enumeration is made when a USB device is connected to a USB port. First, it is detected and then the enumeration begins. In order to establish a connection path between the device and the host, the host sends a sequence of request to device to learb about the device.

The followings are the typical sequence of the enumeration**[4]:**

- The user attaches a device to a USB port. Or the system powers up with a device already attached. The port may be on the root hub at the host or a hub that connects downstream from the host. The hub provides power to the port, and the device is in the **Powered state**
- The hub detects the device
- The host learns of the new device. Each hub uses its interrupt endpoint to report events at the hub. On learning of an event, the host sends the hub a Get_Port_Status request to find out more.
- The hub detects whether a device is low or full speed. The hub detects the speed of a device by determining which line has the higher voltage when idle
- The hub resets the device. When a host learns of a new device, the host controller sends the hub a Set_Port_Feature request that asks the hub to reset the port.
- The host learns if a full-speed device supports high speed.
- The hub establishes a signal path between the device and the bus. The host verifies that the device has exited the reset state by sending a Get_Port_Status request. A bit in the returned data indicates whether the device is still in the reset state. If necessary, the host repeats the request until the device has exited the reset state.
- The host sends a Get_Descriptor request to learn the maximum packet size of the default pipe.
- The host assigns an address. The host controller assigns a unique address to the device by sending a Set_Address request.
- The host learns about the device's abilities. The host sends a Get_Descriptor request to the new address to read the device descriptor. This time the host retrieves the entire descriptor. The descriptor is a data structure containing the maximum packet size for Endpoint 0, the number of configurations the device supports, and other basic information about the device.
- The host assigns and loads a device driver (except for composite devices). After learning about a device from its descriptors, the host looks for the best match in a device driver to manage communications with the device. In selecting a driver, Windows tries to match the information in the PC's INF files with the Vendor ID, Product ID, and (optional) release number retrieved from the device. If there is no match, Windows looks for a match with any class, subclass, and protocol values retrieved from the device. After the operating system assigns and loads the driver, the driver may request the device to resend descriptors or send other class-specific descriptors.

- The host's device driver selects a configuration. After learning about a device from the descriptors, the device driver requests a configuration by sending a Set_Configuration request with the desired configuration number.

The above sequence is the same for the most of the USB systems. For our project, a similar enumeration steps should be held for a healthy connection. The most of the enumeration steps are fulfilled by the EZ-USB chip and as USB device developers, we do not have to deal with the enumeration implementation. The EZ-USB chip have a default enumeration descriptors and the users can load different ones.

Driver searching and the driver attachment are difficult and neceassary steps to establish a logical path between the device and the host. Therefore, we should talk about the device drivers.

Devices that attach to the bus can be full-custom devices requiring a full-custom device driver to be used, or may belong to a device class. These classes define an expected behavior in terms of device and interface descriptors so that the same device driver may be used for any device that claims to be a member of a certain class. An operating system is supposed to implement all device classes so as to provide generic drivers for any USB device. Device classes are decided upon by the Device Working Group of the USB Implementers Forum **[1].**

Some of the device classes are the following **[1]:**

- **0x03:HID(Human Interface Device):** Keyboards and mice..
- **0x08:Mass Storage Device:** flah drivers, memory cards…
- **0x0E:Wireless controllers:** Bluethoot dongles

As stated above, in order to use a device over USB, a proper driver should be attached to the device and the driver file is the first point of the tranfer. The speed of the transfer also depends on the content of the driver file. That is to say, to have a high speed transfer rate the device driver should support this requirement. The writing of a driver file is a quite complex job and it is not necessary for our design. Fortunately, with the USB chip, there is also a suitable driver file that is written for high speed transfer applications. The name of the file is

CyUSB.sys and it is working well. Near the driver file, a managed Microsof .NET 2.0 Class Library(CyUSB.dll) is also available, and this class is used to communicate the device with an application program that written in the .NET.

2.5 USB DESCRIPTORS

At the end of the enumeration, the host tries to learn about the device by using USB descriptors. These explanatory tables provide the needed knowledge to the host to communicate the device appropriately. USB descriptors are standard to the most of the systems and the common ones are the following:

- **Device Descriptors:** USB devices can only have one device descriptor. The device descriptor includes information such as what USB revision the device complies to, the Product and Vendor IDs used to load the appropriate drivers and the number of possible configurations the device can have **[5].**

- **Configuration Descriptors:** The configuration descriptor specifies values such as the amount of power this particular configuration uses, if the device is self or bus powered and the number of interfaces it has. When a device is enumerated, the host reads the device descriptors and can make a decision of which configuration to enable. It can only enable one configuration at a time **[5].**

- **Interface Descriptors:** The interface descriptor could be seen as a header or grouping of the endpoints into a functional group performing a single feature of the device **[5].**

- **Endpoint Descriptors:** Each endpoint descriptor is used to specify the type of transfer, direction, polling interval and maximum packet size for each endpoint. Endpoint zero, the default control endpoint is always assumed to be a control endpoint and as such never has a descriptor **[5].**

- **String Descriptors:** String descriptors provide human readable information and are optional. If they are not used, any string index fields of descriptors must be set to zero indicating there is no string descriptor available **[5].**

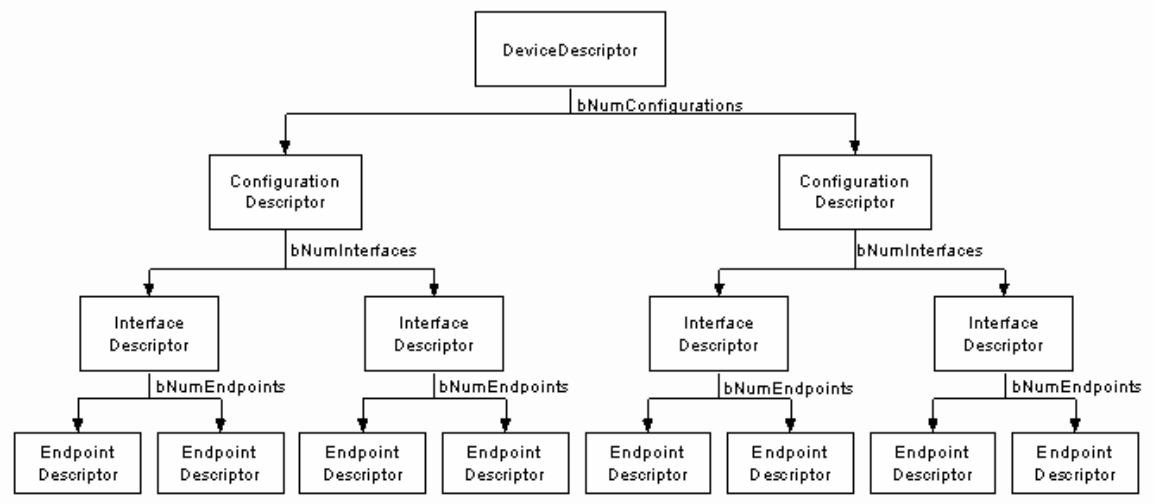


Figure 2.5.1: USB Descriptor Tree

The architecture of a generic USB device is multi-layered. A device consists of one or more configurations, each of which describes a possible setting the device can be programmed into. Such settings can include the power characteristics of the configuration (for example, the maximum power consumed by the configuration and whether it is self-powered or not) and whether the configuration supports remote wake-up **[2].**

Each configuration contains one or more interfaces that are accessible after the configuration is set. An interface provides the definitions of the functions available within the device and may even contain alternate settings within a single interface. For example, an interface for an audio device may have different settings you can select for different bandwidths **[2].**

Each layer of a USB device provides information about its attributes and resource requirements in its descriptor, a data structure accessible through device interface functions. By examining the descriptors at each layer, you can determine exactly which endpoint you need to communicate successfully with a particular device**[2].**

2.6. ENDPOINTS

As stated above, the data transfer process can be configurated with the descriptors. However, in the device, there are a fixed number of communication channels called endpoints that can be used in different configurations. The data flow is made on the endpoints that are logical connections between the computer and the device. The endpoints can be thought as follows:

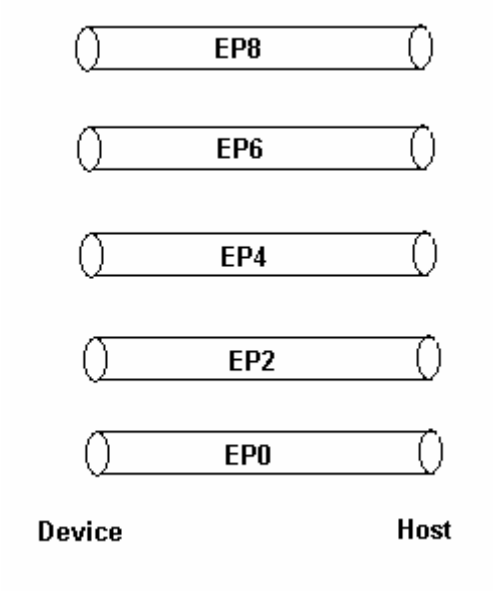


Figure 2.6.1: Logical Endpoint Connection

Each endpoint can be activated or deactivated by setting the endpoint configuration register in the EZ-USB chip. In default, all the endpoints are enabled and so, in order to deactivate some of them, a proper firmware is needed.

An endpoint can have different characteristics such as transfer type, maximum packet size and the transfer direction that can be set in the endpoint descriptor table. The same endpoints can be used in different interfaces. The settings of an endpoint is quite important

because of the fact that the data flow is made over these logical channels and the data transfer speed mainly depends on them.

An endpoint have two types of transfer direction namely IN and OUT with respect to the host. If an endpoint is programmed as IN, the data comes from USB chip to the host system(more clearly computer). By the way, the problem is that where the data comes from. Each endpoint has a FIFO or other form of memory to store the incoming data from the external interface if it is a IN endpoint. Similarly, if the endpoint is programmed as OUT, the data comes from the host systems is stored to this memory. In the project, we mainly make use of IN endpoint type because, the data comes from the external supply to tranfer to the computer and OUT endpoint is not necessary for the project. OUT endpoints are used when the host sends data to a external hardware. The endpoints, except control endpoints, are unidirectional and that is to say, an endpoint can be configurated as either IN or OUT, but not both.

A special kind of endpoint is the control endpoint and this one is used to transfer the control signals between the host and the device. This endpoint is always used and two types of transfer direction are available for this one. Interrupts, setup packets and similar things are transferred over this endpoint. The memory size of the control endpoint is less than the other ones. The enumeration makes use of the control endpoint.

The another important property of an endpoint is the its transfer type. As easily guessed, there are different kinds of data transfers and the endpoints should have different data transfer capabilities. There are four different transfer types namely Control, Interrupt, Isochronous and Bulk.

- Control: It is intended to support configuration, command, and status communication between the host software and the device. Control transfers support error detection and retry**[2].**

  Control transfers are typically used for command and status operations. They are essential to set up a USB device with all enumeration functions being performed

using control transfers. They are typically bursty, random packets which are initiated by the host and use best effort delivery **[6].**

- **Interrupt:** It is used to support small, limited-latency transfers to or from a device such as coordinates from a pointing device or status changes from a modem. Interrupt transfers support error detection and retry**[2].**

  Interrupt transfers are typically non-periodic, small device "initiated" communication requiring bounded latency. An Interrupt request is queued by the device until the host polls the USB device asking for data **[6].**

- **Isochronous:** It is used for periodic, continuous communication between the host and the device, usually involving time-relevant information such as audio or video data streams. Isochronous transfers do not support error detection or retry**[2].**

- **Bulk:** It is intended for non-periodic, large-packet communication with relaxed timing constraints such as between the host software and a printer or scanner. Bulk transfers support error detection and retry**[2].**

  Bulk transfers are only supported by full and high speed devices. For full speed endpoints, the maximum bulk packet size is either 8, 16, 32 or 64 bytes long. For high speed endpoints, the maximum packet size can be up to 512 bytes long **[6].**

The project requires continuous data flow with the time constraint. In the circuit, we don't make use of any memory device and therefore, the incoming data should be transferred wihout any time delay. The incoming data is huge and some small amount of data loss is not too much important. By making use of these knowledge, we can conclude that the endpoint should be programmed as isochronous IN endpoint.

It is necessary to understand the isochronous data transfers for the sake of success of the project.

Isochronous Transfers provide **[6]**

- Guaranteed access to USB bandwidth.
- Bounded latency.
- Stream Pipe – Unidirectional.
- Error detection via CRC, but no retry or guarantee of delivery.
- Full & high speed modes only.
- No data toggling.

The maximum size data payload is specified in the endpoint descriptor of an Isochronous Endpoint. As the maximum data payload size is going to effect the bandwidth requirements of the bus, it is wise to specify a conservative payload size. If you are using a large payload, it may also be to your advantage to specify a series of alternative interfaces with varying isochronous payload sizes **[6].**

In the project, the development of the firmware for the endpoint is quite critical and important mission. It is obvious that if the endpoints are not configurated appropriately the data transfer speed is not sufficient.

CHAPTER 3: EZ-USB DEVELOPMENT KIT CY3684

As expressed before, in the project, a high speed USB peripheral namely CY7C68013A is planned to use.However, the advanced packaging of the chip makes difficult to place the chip on a simple PCB. At the same time, the necessary firmware and the driver files necessiates an advanced knowledge about the USB and the driver, a simplification at the start is needed to use such a chip. The solution is a EZ-USB based development kit.

CY3684 is a EZ-USB advanced development kit. It includes a Cypress USB generic driver file, EZ-USB firmware library and firmware frameworks, Cypress USB class library(CyApi), Cypress USB console, Cypress GPIF(General Programmable Interface) designer, Cypress Firmware Download driver sample and Keil 8051 Development Tools that is necessary to develop the firmware.

The development kit makes simple for us to develop a firmware and application program on the host side. A working firmware and application program is developed and a working circuit diagram of the EZ-USB chip is available and it means that we can bypass the development kit from the project easily.

EZ-USB Development kit requires the following system properties:

- Microsoft Windows XP, Window 2000 or Windows ME
- 64MBytes RAM
- 55Mbytes Disk space
- 300MHz or higher processor
- Super VGA display
- USB host controller(High speed)

On the development board, an EZ-USB FX2LP chip, a re-programmable GAL, 128kilobytes RAM, EEPROMs used for EZ-USB initialization are available. In the project, we mainly work with EZ-USB FX2LP chip namely CY7C68013A.

The external connection to the development kit can be made via the connectors. The development kit has 7 such connector and these are connected to the proper pins of the chips and the connection scheme can seen from the datasheet of the development kit. These connectors can have diverse functions depend on the firmware loaded to the internal RAM.

By using the application program provided with the development kit, the connected devices, their descriptors and the other information about the connections and devices can be obtained. Sample firmwares can be downloaded to the internal RAM of chip, and the external data can be transferred to the host over bulk or isochronous endpoints. This is quite instructive for us and we can simplify the firrmware development process considerably.

The used USB chip in the project is CY7C68013A is available on the development board. Via the development board, we can work with the EZ-USB peripheral easily.

3.1. CY7C68013A:

The data transfer is fulfilled by this chip. This is a USB2.0-USB-IF high speed certificated microcontroller. As expressed above, the project necessiates a huge amount of data transfer and we have to use a high speed USB. Inside the chip an USB2.0 transceiver, smart SIE and enhanced 8051 microprocessor are available.

This chip supports both of full speed and high speed data transfers. The used USB signaling scheme depends on the USB descriptors.

8051 microprocessor should be programmed before the usage and the code can run from the internal RAM or external memory device. The code can be downloaded to the internal RAM via USB or EEPROM. That is to say, after the power down, the code in the internal RAM of chip is deleted and for every power up the internal RAM should be loaded again. If EEPROM is used, the internal RAM automatically programmed after the power up. Otherwise, the program code in HEX should be transferred to the internal RAM via USB.

In the chip, there are four programmable bulk/interrupt/isochronous endpoints and the buffering options for them are double, triple and quad. These settings are determined by the firmware downloaded to the internal RAM via USB in our project. The endpoints can be configurated as IN or OUT, but in the project, we only make use of IN endpoint.

The chip can support 8 or 16 bit external data interface and one of them should used in the data transfer. Again, this is determined by the firmware. The chip has a GPIF(General Programmable Interface) that is used to connect directly to the most parallel interface. The power supply is 3.3V,but the input magnitude can be 5V. The chip can be powered from the USB port by using a voltage regulator. 8051 cpu operation can be 48MHz, 24MHz or 12MHz and again this set by the firmware.

Four integrated FIFOs are available and these are written by using internal or external clock. In the project, these memories are filled by the output of ADC and then transferred to the computer. When the FIFO buffer for an endpoint is filled fully, a flag interrupt is created and the flag pin of the chip changes its status appropriately. This flag interrupt can be used by 8051 microprocessor to handle some functions. By making use of these interrupts, some predetermined data can be added to FIFO or for a while FIFO is disabled and similar things can be implemented in the firmware.

During the power-up sequence, the chip checks an EEPROM connection to load the descriptors. If it can not found the connection, it enumerates with the default internal

descriptors with VID=0x04B4, PID=0x8613 and DID=0xAnnn(nnn depends on the chip revision).

3.2. EZ-USB FIRMWARE

In order to use the EZ-USB peripheral, it should be programmed with a firmware downloaded to the internal RAM of the chip. The tasks of the firmware consist of the initialization of the necessary registers and the control of the data transfer.

The firmware download is the first step that should be done and in the project the firmware is downloaded via the USB. So, the application program should also able to program the chip and the application program is discussed next.

After explaination of the USB specifications and the used USB chip, it is time to talk about the developed firmware. First of all, it is quite instructive to look at the blocks of the USB system as follows:

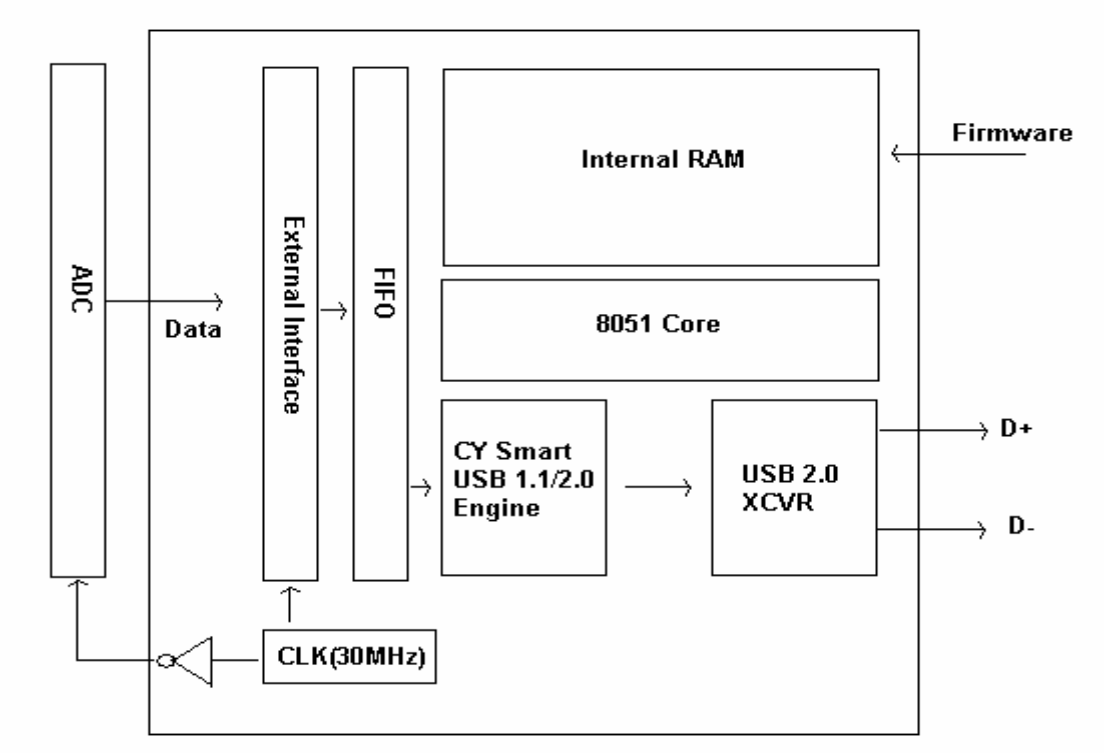


Figure 3.2.1: USB System

The firmware runs from the internal RAM and the internal RAM is programmed via USB in the project. The code in the internal RAM is used by the 8051 core to function. 8051 controls the interrupts, I/O ports, the external interface and so on. The decoding and the encoding of the USB signals are made by the smarts USB engine and USB XCVR. The microprocessor is not necessary to form D+ and D- signals. External interface can be programmed in 8-bits or 16-bits width and the incoming data is written to FIFO with respect to some external or internal sources(maybe CLK as shown in the above figure). The data from the FIFO is converted to corresponding USB signals and then sent to the host and this process is bit by bit .

In development of the firmware, the first step is the initialization of the registers. This is done by TD_Init() function in the CYStream.c file.(The source code of the firmware at the **appendix-1**). In this function, the endpoints, interface and the cpu clock is configurated. The most important register to set is the interface configuration register because of the fact that it controls the external data flow scheme to FIFOs. The bit by bit definition of the register is the following:

IFCONFIG			Interface Configuration(Ports, GPIF, slave FIFOs)				E601
b7	b6	b5	b4	b3	b2	b1	b0
IFCLKSRC	**3048MHZ**	**IFCLKOE**	**IFCLKPOL**	**ASYNC**	**GSTATE**	**IFCFG1**	**IFCFG0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	0	0	0	0	0	0

Table 3.2.1: IFCONFIG Register.

In the code, IFCONFIG is set to 0xB3. By means of this, internal 30MHz clock and inverted clock out is enabled. The external interface is used in the slave mode. In this configuration, the selected endpoint's FIFO is filled by the data available at the FIFO pins with respect to the internal 30MHz clock. The interface clock configuration is the following:

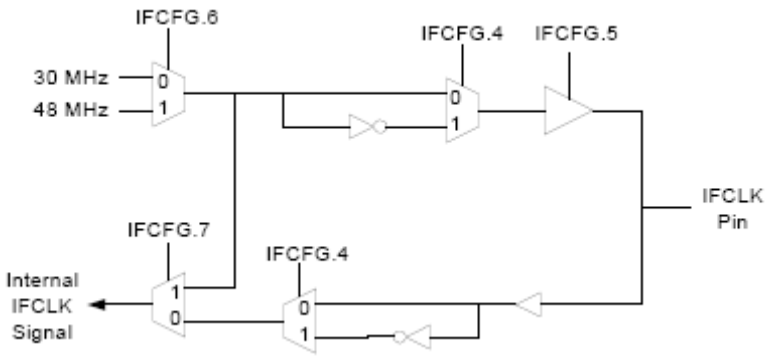


Figure 3.2.2:IFCLK Configuration.

The main clock comes from the development kit and and this clock is used by ADC to synchronize the EZ-USB peripheral. If the same clock is used by both the external interface and the ADC, there may be some indeterminate state and the FIFO content may be wrong. The situation is that ADC tries to sample the data and at the same time the external interface tries to write it to FIFO and this is quite critical to succeed. Fortunately, by inverting the output clock, the external interface clock and ADC clock can have 180^{o} phase difference and this prevents the data loss problem. In this setup, the data is sampled and after a half period time, the converted data is written to FIFO. If the conversion time of ADC and the writing time of FIFO are less than the half period of the interface clock, the problem is solved. In the following figure, the input clocks for the external interface and the ADC is shown.

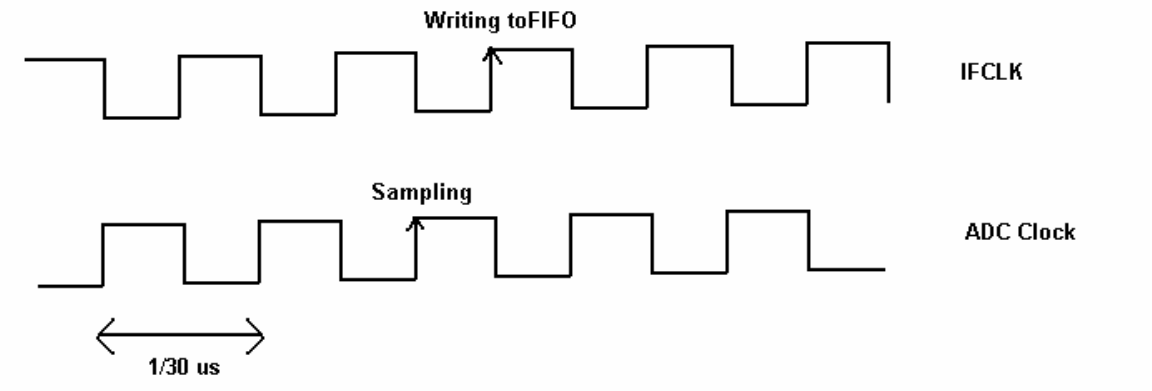


Figure 3.2.3: Used clocks by the external interface and ADC.

The second important setting made in the initialization routine is the endpoint configuration. As explained before, the endpoints should be configurated suitably in order to increase the transfer speed. Each endpoint is configurated by setting the EPxCFG registers. The bit definition of the EP2CFG is the following:

EP2CFG			Endpoint 2 Configuration				E612
b7	b6	b5	b4	b3	b2	b1	b0
VALID	DIR	TYPE1	TYPE0	SIZE	0	BUF1	BUF0
R/W	R/W	R/W	R/W	R/W	R	R/W	R/W
1	0	1	0	0	0	1	0

Table 3.2.2:EP2CFG

In order to activate the endpoint, Bit-7(VALID) should be set to 1 and otherwise, this endpoint is deactivated. The data transfer direction of the endpoint is determined by the Bit-6(DIR) and for IN endpoint, this bit should be set to 1 and otherwise, the endpoint is configurated as OUT. The data transfer types of the endpoint is set by the Bit5 and Bit4. The followig table shows the endpoint type setting scheme.

TYPE1	TYPE0	Endpoint Type
0	0	Invalid
0	1	Isochronous
1	0	Bulk (default)
1	1	Interrupt

Table 3.2.3: Endpoint Type

The size of the endpoint is set by Bit-3 of the register. If this bit set to 1, the size is 512 bytes and otherwise, it is 1024bytes. Buffering type is determined by the Bit-1 and Bit-2 of the endpoint configuration register.

BUF1	BUF0	Buffering
0	0	Quad
0	1	Invalid
1	0	Double
1	1	Triple

Table 3.2.4: Buffering

Other endpoint configurations are similar to above configuration scheme. In the initialization subroutine the last job is the initialization of the external interface. The external interface width is determined by the WORDWIDE bit in the EPxFIFOCFG registers. If this bit is set to 0, the width of the interface is 8bits and otherwise the external width is 16bits. In the code this bit is set to 0 (8 bit wide interface is used). The initialization subroutine is called once at the beginning the program flow.

The second function in the CYStream.c file is the TD_Poll() and this called repeatedly while the device is in the idle mode. The task of the this function is to control the data to transfer over the selected interface and endpoint.

Lastly, the main function is available at the fw.c file and again this file is included at the **appendix-** . The main function runs the initialization subroutine first and then repeatedly TD_Poll() is called while the device is in idle state.

In the application program, EZ-USB can be programmed first. The programming sequence is the following:

- If the user tries to use the EZ-USB before the downloading of the firmware, a warning message is created as follows:

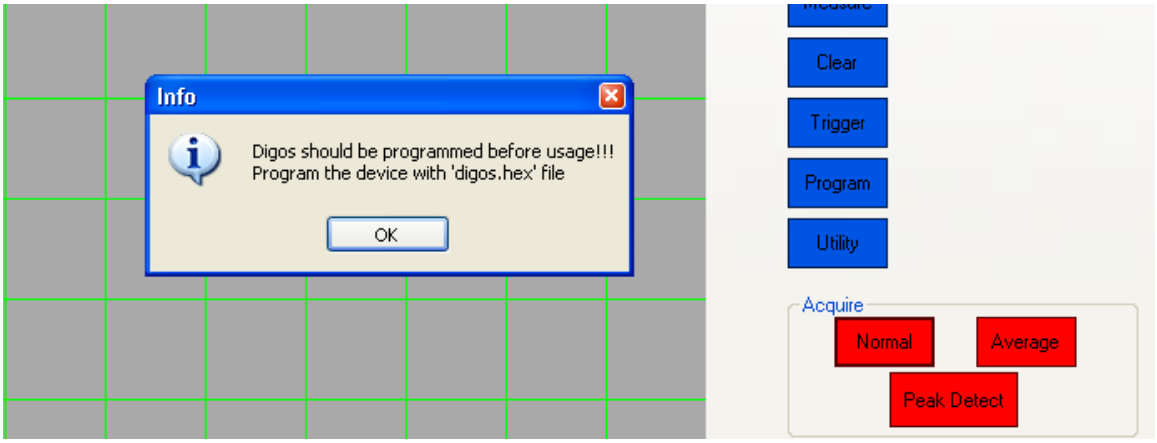


Figure 3.2.4: Programming Warning Message.

- If the user programs the EZ-USB, the following messages are prompted. The first message is for the removal of the Cypress device because, the attached Cypress device is removed first. The second message is the arrival of the new Cypress device. The new device and the old device have different VID and PID and therefore, they attach to the different driver files. After the programming, the enumeration is held again to establish a new channel between the host and the device.

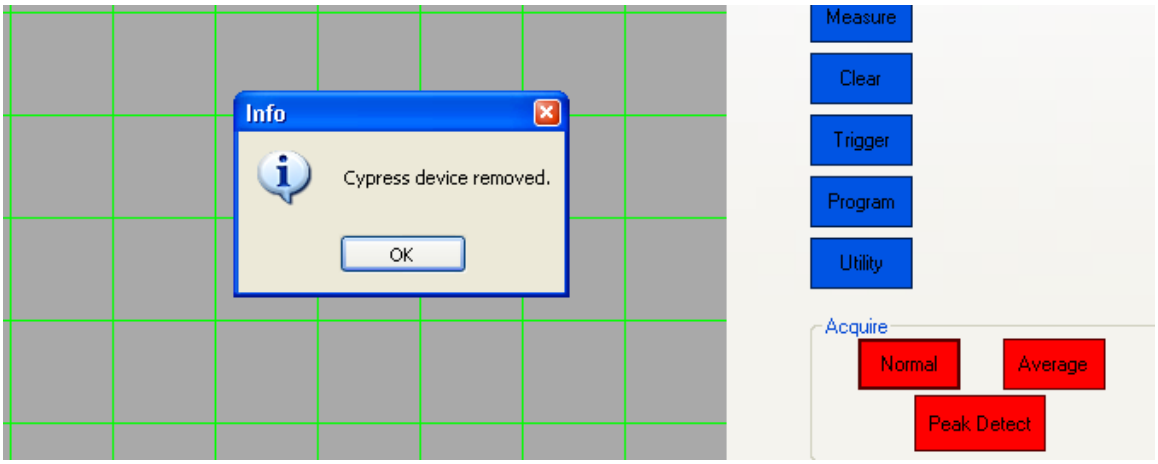


Figure 3.2.5: Device Removal Message.

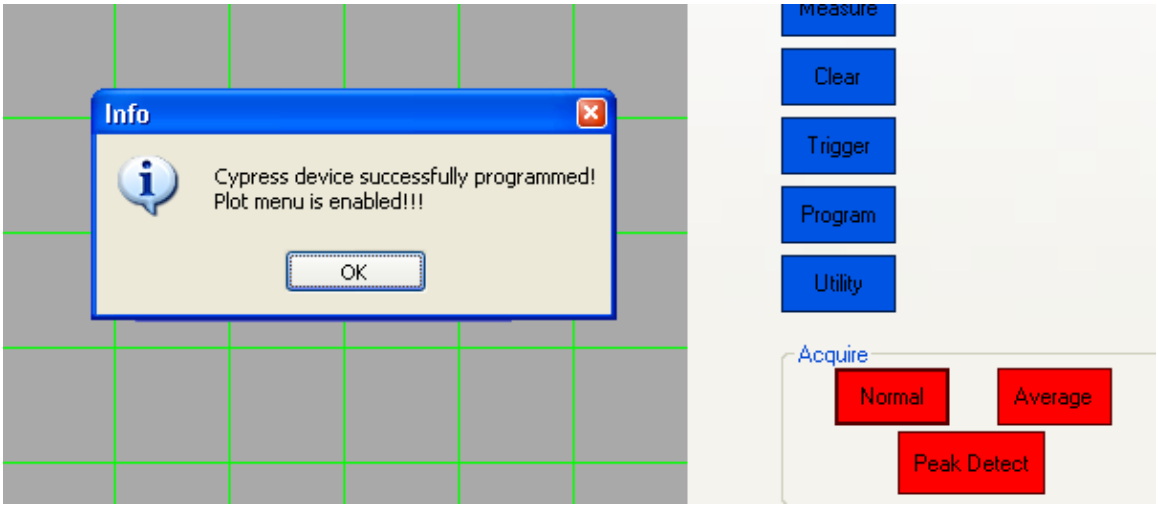


Figure 3.2.6: Device Arrival Message.

CHAPTER 4:USER INTERFACE

4.1. WHY .NET ENVIRONMENT AND C#

We have chosen .NET environment and C# for the interface part of the project. There are a lot of reasons why we have used C#. First of all, C# is a programming language that is widely used in a wide range of applications. It is very easy to learn.

Moreover, since we want to write a real time data plot algorithm, we need a programming language which has to include drawing classes. C# includes very rich drawing classes which are completely suitable for our applications. In addition, C# has very rich visual objects for data plotting.

We have used Cypress microcontroller to provide connection with the interface. There are USB libraries and sample codes written in C# in the CD which we obtained with the development kit. With the help of these libraries, it was very easy to write code to enable the connection between microcontroller and PC.

4.2. CYPRESS LIBRARIES

Cypress has a ‘dll’ file which includes USB libraries to connect USB devices to PC, to warn the user about the connection of the USB devices, to read data from the device connected to PC via USB connection. By adding the library to our C# project, we could use these libraries in C# environment.

Some library items and their syntax of use are the following:

- DEVICES_CYUSB: Finds the USB devices produced by Cypress Inc.
- CyConst.DEVICES_CYUSB: Send a byte value to a proper method according to the types of USB devices we are interested in.
- USBDeviceList: Lists all the USB devices of our interests.
- VariableName[0x????,0x????]: Send the Vendor ID and Product ID of the device to find the specific USB device.
- CyFX2Device: Defines the programmable part of the Cypress device.
- CyUSBEndPoint: Determines the end point of the Cypress USB device.

4.3. SOME SCREENSHOTS FROM THE GUI

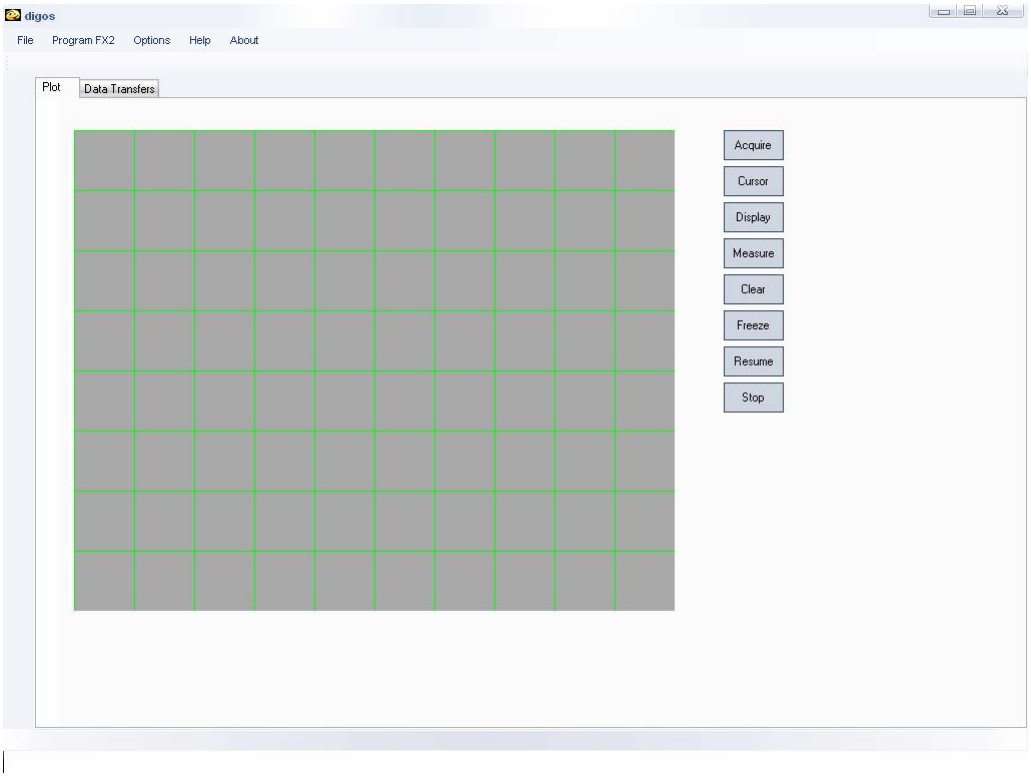


Figure 4.3.1: Initial page of the GUI.

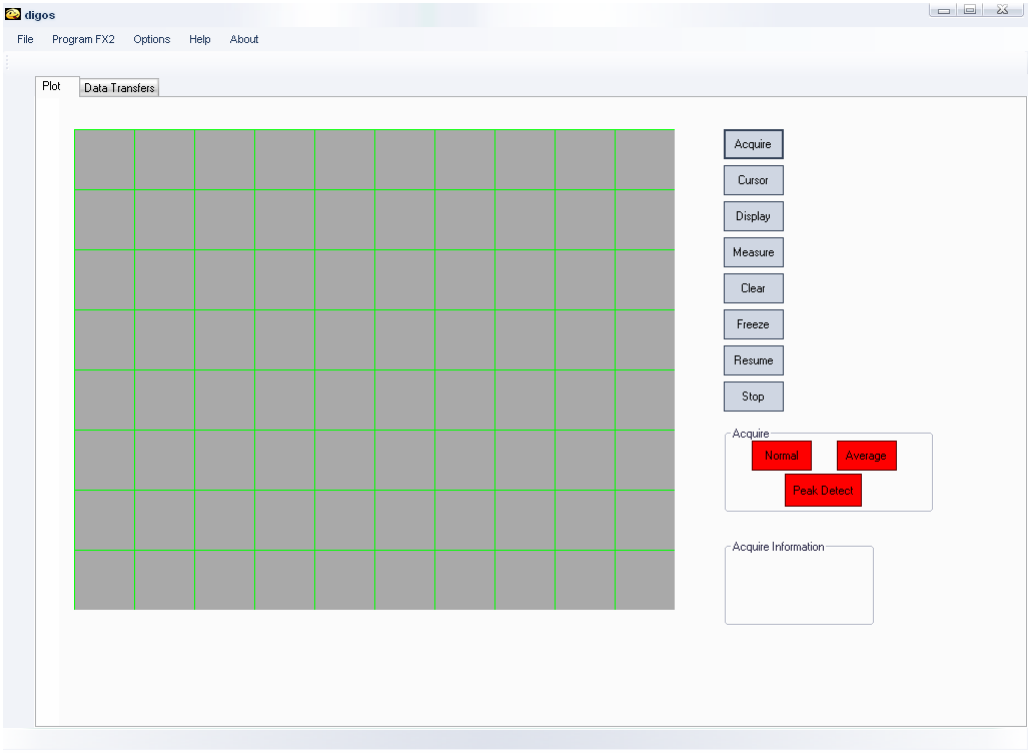


Figure 4.3.2: GUI when Acquire button is clicked.

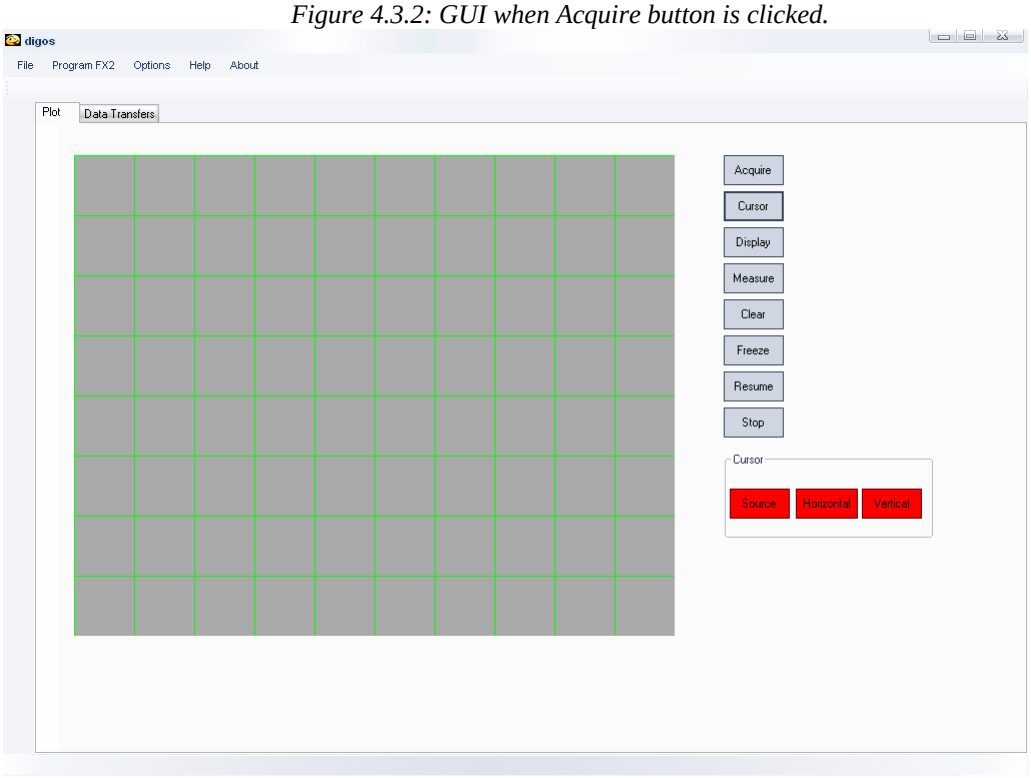


Figure 4.3.3: GUI when Cursor button is clicked.

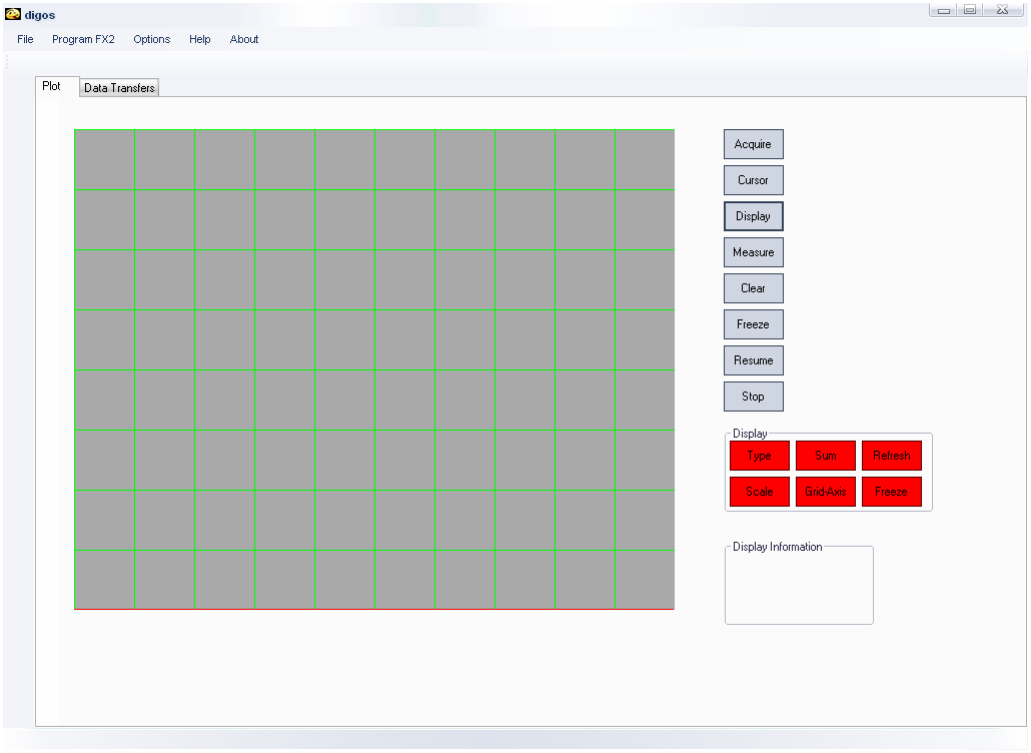


Figure 4.3.4: GUI when Display button is clicked.

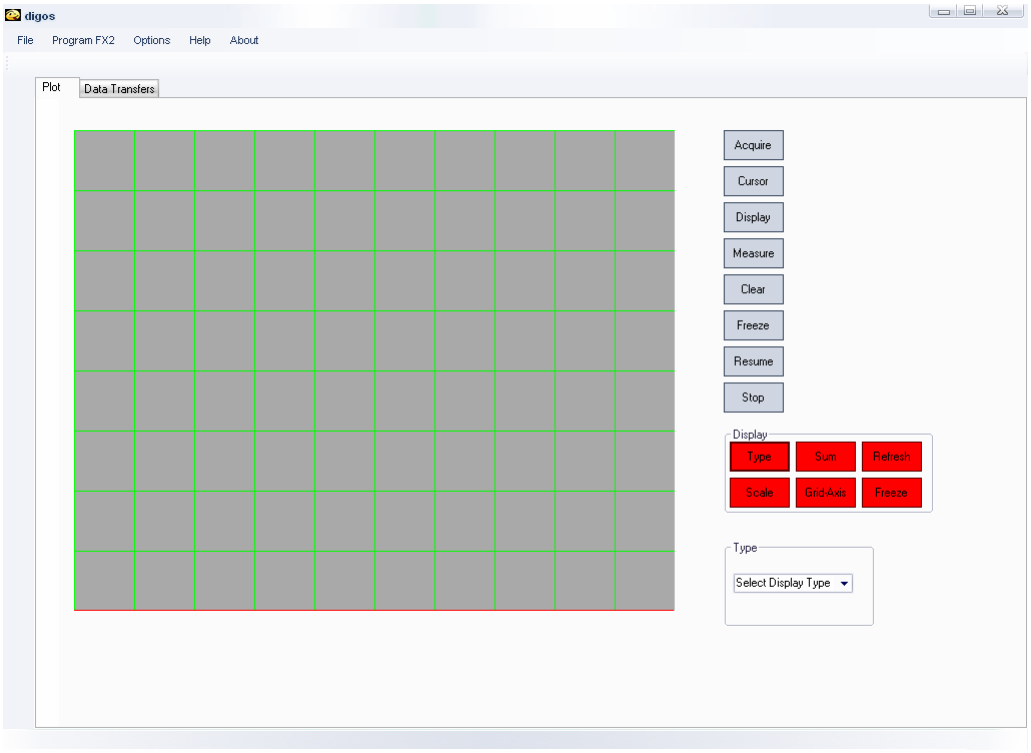


Figure 4.3.5: GUI when Display Type button is clicked.

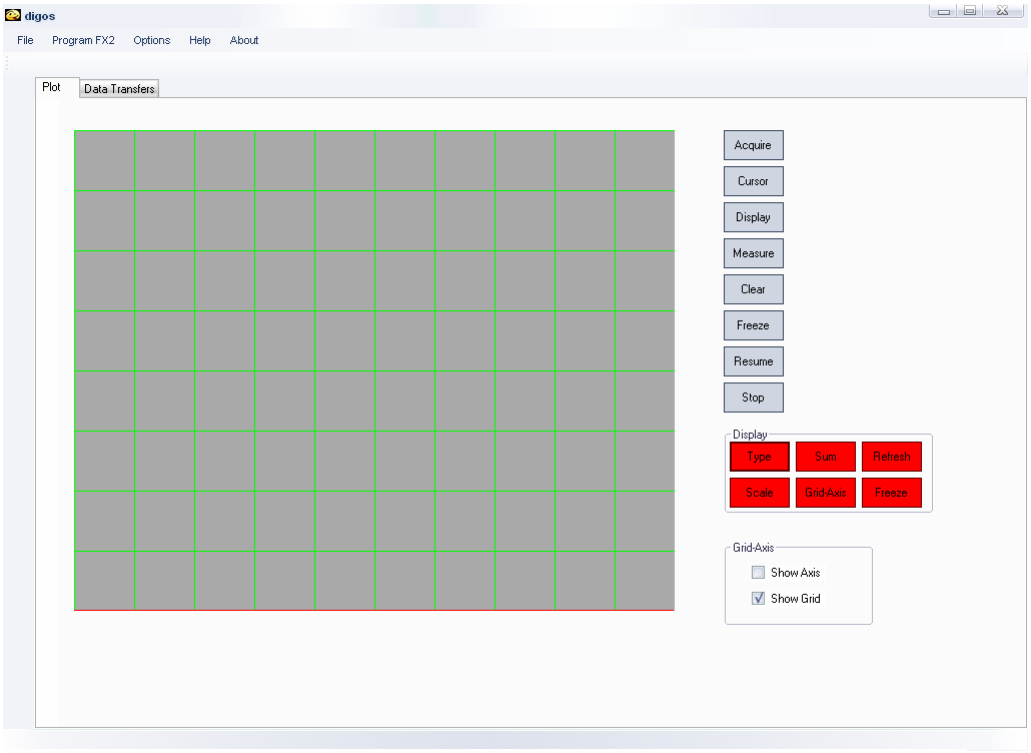


Figure 4.3.6: GUI when Display Grid-Axis button is clicked.

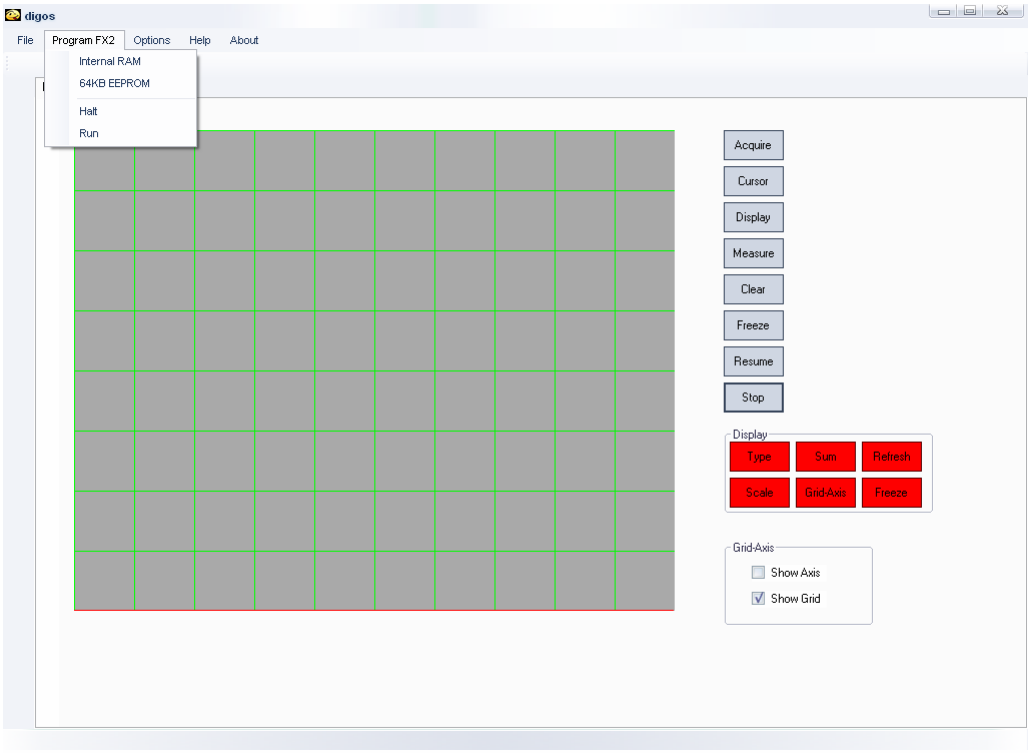


Figure 4.3.7: GUI when Program FX2 item is clicked.

4.4. ALGORITHM

Load the main form

Initialize the components in the main form

Try to find the USB devices of Cypress to read

If USB devices of Cypress couldn't be found to read

Try to find the USB devices of Cypress to program

Warn the user

Call the plug and unplug event handler

If the device is connected or disconnected

Inform the user about the situation

Try to find the USB devices of Cypress to read

If USB devices of Cypress could be found to read

Inform the user about the situation

When main form is being closed

Clean up memory, clean the USB devices added to memory.

Call a function to perform reading the data from Cypress periodically

Find the end point of the microcontroller

If the end point of the microcontroller is null

Warn the user

Else

Start the data transfer from the buffer of the microcontroller until the flag showing that data transfer is complete is interrupted

When Acquire Normal button is clicked

Call the function to activate the first data acquisition

Call the function to activate the first data acquisition

Define a timer to periodically call the data acquisition function(DAQ)

Alert the periodic data acquisition function at each tick of the timer

Set the call period of the DAQ function

Find the end point of the microcontroller

Set the expiration time for DAQ

If the end point of the microcontroller is null

Warn the user

Else

Start the data transfer from the buffer of the microcontroller until the flag showing that data transfer is complete is interrupted

Start the timer after the first DAQ

When the DAQ is completed

Send the data for plotting

Call the DAQ function again

When ‘Save’ item of the menu is clicked

Send the data coming to stream writer function to save it in a file

Until stop button is clicked on the save data form

Continue writing the data into a file

When ‘About’ item of the menu is clicked

Show the form including info about the project

When ‘Exit’ item of the menu is clicked

Clean up the memory

Close the application

When ‘Program FX2 Device’ item of the menu is clicked

Define theFX2 device to program

Choose the files with the extension ".hex" to program the device.

Program the device

When 'Halt' item of the menu is clicked

Halt the programming process

When the graph control area is loaded

Make all of the group boxes invisible

When the buttons are clicked, related group boxes will be visible

When a button is clicked

Hide all the group boxes

Show the group boxes of the active button

When the button to acquire the data in normal mode is clicked

Make the clear screen mode off

Make the start application mode on

Call the function to begin reading and plotting

When the button to clear the data is clicked

Make the clear screen mode on

Call the OnPaint method to redraw the data on the screen

When the check box controlling the axis visibility is checked

Update the color of the pen to draw the axis

Make the axis visibility control true

When the check box controlling the grid visibility is checked

Update the color of the pen to draw the grid

Make the grid visibility control true

When the check box controlling the scale of the data plotting is checked

Adjust the graph according to the scale

Excite the OnPaint method to redraw the data using new settings

When the combo box controlling the type of the data plotting is checked

If 'Vector' item is selected

Make the 'Display Type' boolean true

Connect the data points

Else

Make the 'Display Type' boolean false

Excite the OnPaint method to redraw the data using new settings

When the button to freeze the data flow is clicked

Make the 'freeze' boolean true

When the button to resume the data flow is clicked

Make the 'freeze' boolean false

When the button to stop the data flow is clicked

Make the 'stop' boolean true

Use it when necessary

Load the graph control

Initialize the components in the graph control

When the ReadData function is called

    Start the isynchronous data transfer

    Excite the OnPaint method to redraw the data using new settings

Continously call the OnPaint method

Translate the origin of the graph from left upper corner to left bottom corner

If ‘ShowGrid’ boolean is true

    Draw the grid lines

If ‘ShowAxis’ boolean is true

    Draw the axis lines

If ‘DisplayType’ boolean is true

    Connect the data dots on the screen

APPENDICES

APPENDIX A:

FIRMWARE CODE

//CYStream.c

```
#pragma NOIV               // Do not generate interrupt vectors
//-----------------------------------------------
#include "fx2.h"
#include "fx2regs.h"
#include "fx2sdly.h"           // SYNCDELAY macro

extern BOOL   GotSUD;        // Received setup data flag
extern BOOL   Sleep;
extern BOOL   Rwuen;
extern BOOL   Selfpwr;

#define LED_ADDR                  0x21

enum {
   Alt0_BulkIN = 0,
   Alt1_BulkOUT,
   Alt2_BulkINOUT,
   Alt3_IsocIN,
   Alt4_IsocOUT,
   Alt5_IsocIN,
   Alt6_IsocINOUT
};

enum {
   Full_Alt0_BulkIN = 0,
   Full_Alt1_BulkOUT,
   Full_Alt2_IsocIN,
   Full_Alt3_IsocOUT
};

BYTE xdata Digit[] = { 0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x98, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e };

BYTE    Configuration;      // Current configuration
BYTE    AlternateSetting = Alt5_IsocIN;// Alternate settings

//-----------------------------------------------------------------------------
// Task Dispatcher hooks
//   The following hooks are called by the task dispatcher.
//-----------------------------------------------------------------------------
```

```
WORD mycount;


void TD_Init(void)          // Called once at startup
{
  //int i,j;

  // set the CPU clock to 48MHz
  CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1) ;
  SYNCDELAY;
 // set the slave FIFO interface to 48MHz

  IFCONFIG=0xB3;
  SYNCDELAY;

  // Default interface uses endpoint 2, zero the valid bit on all others
  // Just using endpoint 2, zero the valid bit on all others
  EP1OUTCFG = (EP1OUTCFG & 0x7F);
  SYNCDELAY;
  EP1INCFG = (EP1INCFG & 0x7F);
  SYNCDELAY;
  EP4CFG = (EP4CFG & 0x7F);
  SYNCDELAY;
  EP6CFG = (EP6CFG & 0x7F);
  SYNCDELAY;
  EP8CFG = (EP8CFG & 0x7F);
  SYNCDELAY;

  EP2CFG = 0xE0;  // EP2 is DIR=IN, TYPE=BULK, SIZE=512, BUF=4x

 EP2FIFOCFG=EP2FIFOCFG&0xFE; //WORDWIDE=0; 8 bit data interface is used
  // We want to get SOF interrupts
  USBIE |= bmSOF;

  mycount = 0;

 Rwuen = TRUE;          // Enable remote-wakeup
 }


void TD_Poll(void)          // Called repeatedly while the device is idle
 {
    // ...FX2 in high speed mode
    if( EZUSB_HIGHSPEED( ) )
    {
      // Perform USB activity based upon the Alt. Interface selected
       switch (AlternateSetting)
       {
```

```
                case Alt0_BulkIN:
                  // Send data on EP2
                  while(!(EP2468STAT & bmEP2FULL))
                  {
                    EP2FIFOBUF[0] = LSB(mycount);
                    EP2FIFOBUF[1] = MSB(mycount);
                    EP2FIFOBUF[2] = USBFRAMEL;
                    EP2FIFOBUF[3] = USBFRAMEH;
                    EP2FIFOBUF[4] = MICROFRAME;

                    EP2BCH = 0x02;
                    EP2BCL = 0x00;

                    mycount++;
                  }
                break;

                case Alt2_BulkINOUT:
                  // Send data on EP2
                  while(!(EP2468STAT & bmEP2FULL))
                  {
                    EP2FIFOBUF[0] = LSB(mycount);
                    EP2FIFOBUF[1] = MSB(mycount);
                    EP2FIFOBUF[2] = USBFRAMEL;
                    EP2FIFOBUF[3] = USBFRAMEH;
                    EP2FIFOBUF[4] = MICROFRAME;

                    EP2BCH = 0x02;
                    EP2BCL = 0x00;

                    mycount++;
                  }

                  // check EP6 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is empty
                  if(!(EP2468STAT & bmEP6EMPTY))
                  {
                    EP6BCL = 0x80; // re(arm) EP6OUT
                  }
                break;

                case Alt3_IsocIN:
```

```
                case Alt5_IsocIN:
                  // Send data on EP2
                  while(!(EP2468STAT &
bmEP2FULL))
                  {
                    EP2FIFOBUF[0] =
LSB(mycount);
                    EP2FIFOBUF[1] =
MSB(mycount);
                    EP2FIFOBUF[2] =
USBFRAMEL;
                    EP2FIFOBUF[3] =
USBFRAMEH;
                    EP2FIFOBUF[4] =
MICROFRAME;

                    EP2BCH = 0x04;
                    EP2BCL = 0x00;

                    mycount++;
                  }
                break;


                case Alt1_BulkOUT:
                case Alt4_IsocOUT:
                                        //
check EP2 EMPTY(busy) bit in EP2468STAT
(SFR), core set's this bit when FIFO is empty
                    if(!(EP2468STAT &
bmEP2EMPTY))
                    {
                      EP2BCL = 0x80;
// re(arm) EP2OUT
                    }
                break;

                case Alt6_IsocINOUT:
                {
                  // Send data on EP2
                  while(!(EP2468STAT &
bmEP2FULL))
                  {
                    EP2FIFOBUF[0] =
LSB(mycount);
                    EP2FIFOBUF[1] =
MSB(mycount);
                    EP2FIFOBUF[2] =
USBFRAMEL;
                    EP2FIFOBUF[3] =
USBFRAMEH;
                    EP2FIFOBUF[4] =
MICROFRAME;

                    EP2BCH = 0x04;
```

```
                    EP2BCL = 0x00;

                    mycount++;
                  }

                  // check EP6
EMPTY(busy) bit in EP2468STAT (SFR), core
set's this bit when FIFO is empty
                  if(!(EP2468STAT &
bmEP6EMPTY))
                  {
                    EP6BCL = 0x80;
// re(arm) EP6OUT
                  }
                }
                break;
              }

            }

        /*      else    // Full Speed
            {
              // Perform USB activity based
upon the Alt. Interface selected
               switch (AlternateSetting)
              {
                case Full_Alt0_BulkIN:
                  // Send data on EP2
                  while(!(EP2468STAT &
bmEP2FULL))
                  {
                    EP2FIFOBUF[0] =
LSB(mycount);
                    EP2FIFOBUF[1] =
MSB(mycount);
                    EP2FIFOBUF[2] =
USBFRAMEL;
                    EP2FIFOBUF[3] =
USBFRAMEH;
                    EP2FIFOBUF[4] =
MICROFRAME;

                    EP2BCH = 0x00;
                    EP2BCL = 0x40;

                    mycount++;
                  }
                break;

                case Full_Alt1_BulkOUT:

                  // check EP2
EMPTY(busy) bit in EP2468STAT (SFR), core
set's this bit when FIFO is empty
                  while(!(EP2468STAT &
bmEP2EMPTY))
                  {
```

```
                    EP2BCL = 0x80;
// re(arm) EP2OUT
                }
            break;

            case Full_Alt2_IsocIN:
              // Send data on EP2
              while(!(EP2468STAT &
bmEP2FULL))
              {
                EP2FIFOBUF[0] =
LSB(mycount);
                EP2FIFOBUF[1] =
MSB(mycount);
                EP2FIFOBUF[2] =
USBFRAMEL;
                EP2FIFOBUF[3] =
USBFRAMEH;
                EP2FIFOBUF[4] =
MICROFRAME;

                EP2BCH = 0x03;
        // 1023
                EP2BCL = 0xFF;

                mycount++;
              }
            break;

            case Full_Alt3_IsocOUT:
              // check EP2
EMPTY(busy) bit in EP2468STAT (SFR), core
set's this bit when FIFO is empty
              while(!(EP2468STAT &
bmEP2EMPTY))
              {
                EP2BCL = 0x80;
// re(arm) EP2OUT
              }
            break;
          }
        }                   */
    }
```

BOOL TD_Suspend(void) // Called before the device goes into suspend mode

```
    {
      return(TRUE);
    }
```

BOOL TD_Resume(void) // Called after the device resumes

```
    {
      return(TRUE);
    }
```

```
//-----------------------------------------------
---------------------------
// Device Request hooks
//   The following hooks are called by the
end point 0 device request parser.
//-----------------------------------------------
---------------------------
```

BOOL DR_GetDescriptor(void)

```
{
  return(TRUE);
}
```

BOOL DR_SetConfiguration(void) // Called when a Set Configuration command is received

```
{
  Configuration = SETUPDAT[2];
  return(TRUE);          // Handled by user
code
}
```

BOOL DR_GetConfiguration(void) // Called when a Get Configuration command is received

```
{
  EP0BUF[0] = Configuration;
  EP0BCH = 0;
  EP0BCL = 1;
  return(TRUE);          // Handled by user
code
}
```

BOOL DR_SetInterface(void)

```
// Called when a Set Interface command is received
{
  BYTE          updateDisplay = TRUE;
  AlternateSetting = SETUPDAT[2];

    // ...FX2 in high speed mode
    if( EZUSB_HIGHSPEED( ) )
    {
      // Change configuration based
upon the Alt. Interface selected
      switch (AlternateSetting)
      {
        case Alt0_BulkIN:
          // Only using endpoint 2,
zero the valid bit on all others
          // Just using endpoint 2,
zero the valid bit on all others
          EP2CFG = 0xE0;  // EP2
is DIR=IN, TYPE=BULK, SIZE=512, BUF=4x
          SYNCDELAY;

          EP1OUTCFG =
(EP1OUTCFG & 0x7F);
```

```
                SYNCDELAY;
                EP1INCFG =
(EP1INCFG & 0x7F);
                SYNCDELAY;
                EP4CFG = (EP4CFG &
0x7F);
                SYNCDELAY;
                EP6CFG = (EP6CFG &
0x7F);
                SYNCDELAY;
                EP8CFG = (EP8CFG &
0x7F);
                SYNCDELAY;

                // Clear out any
committed packets
                FIFORESET = 0x80;
                SYNCDELAY;
                FIFORESET = 0x02;
                SYNCDELAY;
                FIFORESET = 0x00;
                SYNCDELAY;

                // Reset data toggle to 0
                TOGCTL = 0x12;  // EP2
IN
                TOGCTL = 0x32;  // EP2
IN Reset
                break;

            case Alt1_BulkOUT:
                // Only using endpoint 2,
zero the valid bit on all others
                EP2CFG = 0xA0;  // EP2
is DIR=OUT, TYPE=BULK, SIZE=512, BUF=4x
                SYNCDELAY;

                EP1OUTCFG =
(EP1OUTCFG & 0x7F);
                SYNCDELAY;
                EP1INCFG =
(EP1INCFG & 0x7F);
                SYNCDELAY;
                EP4CFG = (EP4CFG &
0x7F);
                SYNCDELAY;
                EP6CFG = (EP6CFG &
0x7F);
                SYNCDELAY;
                EP8CFG = (EP8CFG &
0x7F);
                SYNCDELAY;

                // OUT endpoints do NOT
come up armed
                EP2BCL = 0x80; // arm
first buffer by writing BC w/skip=1
```

```
                SYNCDELAY;
                EP2BCL = 0x80; // arm
second buffer by writing BC w/skip=1
                SYNCDELAY;
                EP2BCL = 0x80; // arm
third buffer by writing BC w/skip=1
                SYNCDELAY;
                EP2BCL = 0x80; // arm
fourth buffer by writing BC w/skip=1

                break;

            case Alt2_BulkINOUT:
                // Using endpoints 2 & 6,
zero the valid bit on all others
                EP2CFG = 0xE0; // EP2
is DIR=IN, TYPE=BULK, SIZE=512, BUF=4x
                SYNCDELAY;
                EP6CFG = 0xA0; // EP6
is DIR=OUT, TYPE=BULK, SIZE=512, BUF=4x
                SYNCDELAY;

                EP1OUTCFG =
(EP1OUTCFG & 0x7F);
                SYNCDELAY;
                EP1INCFG =
(EP1INCFG & 0x7F);
                SYNCDELAY;
                EP4CFG = (EP4CFG &
0x7F);
                SYNCDELAY;
                EP8CFG = (EP8CFG &
0x7F);
                SYNCDELAY;

                // Clear out any
committed packets
                FIFORESET = 0x80;
                SYNCDELAY;
                FIFORESET = 0x02;
                SYNCDELAY;
                FIFORESET = 0x00;
                SYNCDELAY;

                // Reset data toggle to 0
                TOGCTL = 0x12;  // EP2
IN
                TOGCTL = 0x32;  // EP2
IN Reset

                // OUT endpoints do NOT
come up armed
                EP6BCL = 0x80; // arm
first buffer by writing BC w/skip=1
                SYNCDELAY;
                EP6BCL = 0x80; // arm
second buffer by writing BC w/skip=1
```

```
                SYNCDELAY;
            break;

            case Alt3_IsocIN:
                // Only using endpoint 2,
zero the valid bit on all others
                EP2CFG = 0xD8;  // EP2
is DIR=IN, TYPE=ISOC, SIZE=1024, BUF=4x
                SYNCDELAY;

                EP1OUTCFG =
EP1INCFG = EP4CFG = EP6CFG = EP8CFG =
0x00;
                SYNCDELAY;

                // Clear out any
committed packets
                FIFORESET = 0x80;
                SYNCDELAY;
                FIFORESET = 0x02;
                SYNCDELAY;
                FIFORESET = 0x00;
                SYNCDELAY;

                // This register sets the
number of Isoc packets to send per
                // uFrame.  This register is
only valid in high speed.
                EP2ISOINPKTS = 0x03;
            break;

            case Alt4_IsocOUT:
            {
                // Only using endpoint 2,
zero the valid bit on all others
                EP1OUTCFG =
EP1INCFG = EP4CFG = EP6CFG = EP8CFG =
0x00;
                SYNCDELAY;
                EP2CFG = 0x98;  // EP2
is DIR=OUT, TYPE=ISOC, SIZE=1024, BUF=4x
                SYNCDELAY;

                // OUT endpoints do NOT
come up armed
                EP2BCL = 0x80; // arm
first buffer by writing BC w/skip=1
                SYNCDELAY;
                EP2BCL = 0x80; // arm
second buffer by writing BC w/skip=1        break;

            }
            break;

            case Alt5_IsocIN:
            {
```

```
                        // Only using endpoint 2,
zero the valid bit on all others
                        EP2CFG = 0xD8;  // EP2
is DIR=IN, TYPE=ISOC, SIZE=1024, BUF=4x
                        SYNCDELAY;

                        EP1OUTCFG =
EP1INCFG = EP4CFG = EP6CFG = EP8CFG =
0x00;
                        SYNCDELAY;

                        // Clear out any
committed packets
                        FIFORESET = 0x80;
                        SYNCDELAY;
                        FIFORESET = 0x02;
                        SYNCDELAY;
                        FIFORESET = 0x00;
                        SYNCDELAY;

                        // This register sets the
number of Isoc packets to send per
                        // uFrame.  This register is
only valid in high speed.
                        EP2ISOINPKTS = 0x01;
                    }
                    break;

                    case Alt6_IsocINOUT:
                    {
                        // Using endpoints 2 & 6,
zero the valid bit on all others
                        EP2CFG = 0xDA; // EP2
is DIR=IN, TYPE=ISOC, SIZE=1024, BUF=2x
                        SYNCDELAY;
                        EP6CFG = 0x9A; // EP6
is DIR=OUT, TYPE=ISOC, SIZE=1024, BUF=2x
                        SYNCDELAY;

                        EP1OUTCFG =
EP1INCFG = EP4CFG = EP8CFG = 0x00;
                        SYNCDELAY;

                        // Clear out any
committed packets
                        FIFORESET = 0x80;
                        SYNCDELAY;
                        FIFORESET = 0x02;
                        SYNCDELAY;
                        FIFORESET = 0x00;
                        SYNCDELAY;

                        // This register sets the
number of Isoc packets to send per
                        // uFrame.  This register is
only valid in high speed.
                        EP2ISOINPKTS = 0x01;
```

```
                        // OUT endpoints do NOT
come up armed
                        EP6BCL = 0x80; // arm
first buffer by writing BC w/skip=1
                        SYNCDELAY;
                        EP6BCL = 0x80; // arm
second buffer by writing BC w/skip=1

                    }
                    break;

                }
            }
        else
        {
            // Change configuration based
upon the Alt. Interface selected
            switch (AlternateSetting)
            {
              case Full_Alt0_BulkIN:
                // Only using endpoint 2,
zero the valid bit on all others
                // Just using endpoint 2,
zero the valid bit on all others
                EP2CFG = 0xE0;  // EP2
is DIR=IN, TYPE=BULK, SIZE=512, BUF=4x
                SYNCDELAY;
                EP1OUTCFG =
(EP1OUTCFG & 0x7F);
                SYNCDELAY;
                EP1INCFG =
(EP1INCFG & 0x7F);
                SYNCDELAY;
                EP4CFG = (EP4CFG &
0x7F);
                SYNCDELAY;
                EP6CFG = (EP6CFG &
0x7F);
                SYNCDELAY;
                EP8CFG = (EP8CFG &
0x7F);
                SYNCDELAY;

                // Clear out any
committed packets
                FIFORESET = 0x80;
                SYNCDELAY;
                FIFORESET = 0x02;
                SYNCDELAY;
                FIFORESET = 0x00;
                SYNCDELAY;

                // Reset data toggle to 0
                TOGCTL = 0x12;  // EP2
IN
```

```
                TOGCTL = 0x32;  // EP2
IN Reset

                break;

                case Full_Alt1_BulkOUT:
                  // Only using endpoint 2,
zero the valid bit on all others
                EP2CFG = 0xA0;  // EP2
is DIR=OUT, TYPE=BULK, SIZE=512, BUF=4x
                SYNCDELAY;

                EP1OUTCFG =
(EP1OUTCFG & 0x7F);
                  SYNCDELAY;
                  EP1INCFG =
(EP1INCFG & 0x7F);
                  SYNCDELAY;
                  EP4CFG = (EP4CFG &
0x7F);
                  SYNCDELAY;
                  EP6CFG = (EP6CFG &
0x7F);
                  SYNCDELAY;
                  EP8CFG = (EP8CFG &
0x7F);
                  SYNCDELAY;
                  // OUT endpoints do NOT
come up armed
                  EP2BCL = 0x80; // arm
first buffer by writing BC w/skip=1
                  SYNCDELAY;
                  EP2BCL = 0x80; // arm
second buffer by writing BC w/skip=1
                  SYNCDELAY;
                  EP2BCL = 0x80; // arm
third buffer by writing BC w/skip=1
                  SYNCDELAY;
                  EP2BCL = 0x80; // arm
fourth buffer by writing BC w/skip=1

                break;

                case Full_Alt2_IsocIN:
                  // Only using endpoint 2,
zero the valid bit on all others
                  EP2CFG = 0xD8;  // EP2
is DIR=IN, TYPE=ISOC, SIZE=1024, BUF=4x
                  SYNCDELAY;

                  EP1OUTCFG =
EP1INCFG = EP4CFG = EP6CFG = EP8CFG =
0x00;
                  SYNCDELAY;
```

```
                        // Clear out any
committed packets
                        FIFORESET = 0x80;
                        SYNCDELAY;
                        FIFORESET = 0x02;
                        SYNCDELAY;
                        FIFORESET = 0x00;
                        SYNCDELAY;

                    break;

                    case Full_Alt3_IsocOUT:
                    {
                        // Only using endpoint 2,
zero the valid bit on all others
                        EP1OUTCFG =
EP1INCFG = EP4CFG = EP6CFG = EP8CFG =
0x00;
                        SYNCDELAY;
                        EP2CFG = 0x98;  // EP2
is DIR=OUT, TYPE=ISOC, SIZE=1024, BUF=4x
                        SYNCDELAY;

                        // OUT endpoints do NOT
come up armed
                        EP2BCL = 0x80; // arm
first buffer by writing BC w/skip=1
                        SYNCDELAY;
                        EP2BCL = 0x80; // arm
second buffer by writing BC w/skip=1        break;

                    }
                    break;

                }
            }

        // Update the display to indicate the
currently selected alt. Interface
            if(updateDisplay)
            {

EZUSB_WriteI2C(LED_ADDR, 0x01,
&(Digit[AlternateSetting]));

EZUSB_WaitForEEPROMWrite(LED_ADDR);
                updateDisplay = FALSE;
            }

        return(TRUE);          // Handled by user
code
    }
```

```
BOOL DR_GetInterface(void)
// Called when a Set Interface command is received
{
  EP0BUF[0] = AlternateSetting;
  EP0BCH = 0;
  EP0BCL = 1;
  return(TRUE);          // Handled by user
code
}

BOOL DR_GetStatus(void)
{
  return(TRUE);
}

BOOL DR_ClearFeature(void)
{
  return(TRUE);
}

BOOL DR_SetFeature(void)
{
  return(TRUE);
}

BOOL DR_VendorCmnd(void)
{
  return(TRUE);
}

//-----------------------------------------------
---------------------------
// USB Interrupt Handlers
//   The following functions are called by
the USB interrupt jump table.
//-----------------------------------------------
---------------------------

// Setup Data Available Interrupt Handler
void ISR_Sudav(void) interrupt 0
{
  GotSUD = TRUE;          // Set flag
  EZUSB_IRQ_CLEAR();
  USBIRQ = bmSUDAV;       // Clear
SUDAV IRQ
}

// Setup Token Interrupt Handler
void ISR_Sutok(void) interrupt 0
{
  EZUSB_IRQ_CLEAR();
  USBIRQ = bmSUTOK;       // Clear
SUTOK IRQ
}

void ISR_Sof(void) interrupt 0
{
```

```
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSOF;          // Clear SOF
IRQ
  }
```

void ISR_Ures(void) interrupt 0

```
  {
    // Whenever we get a USB Reset, we
should revert to full speed mode
    pConfigDscr = pFullSpeedConfigDscr;
    ((CONFIGDSCR xdata *)
pConfigDscr)->type = CONFIG_DSCR;
    pOtherConfigDscr =
pHighSpeedConfigDscr;
    ((CONFIGDSCR xdata *)
pOtherConfigDscr)->type =
OTHERSPEED_DSCR;

    EZUSB_IRQ_CLEAR();
    USBIRQ = bmURES;         // Clear
URES IRQ
  }
```

void ISR_Susp(void) interrupt 0

```
  {
    Sleep = TRUE;
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmSUSP;
  }
```

void ISR_Highspeed(void) interrupt 0

```
  {
    if (EZUSB_HIGHSPEED())
    {
      pConfigDscr =
pHighSpeedConfigDscr;
      ((CONFIGDSCR xdata *)
pConfigDscr)->type = CONFIG_DSCR;
      pOtherConfigDscr =
pFullSpeedConfigDscr;
      ((CONFIGDSCR xdata *)
pOtherConfigDscr)->type =
OTHERSPEED_DSCR;

      // This register sets the number of Isoc
packets to send per
      // uFrame.  This register is only valid
in high speed.
      EP2ISOINPKTS = 0x03;
    }
    else
    {
      pConfigDscr = pFullSpeedConfigDscr;
      pOtherConfigDscr =
pHighSpeedConfigDscr;
    }
```

```
    EZUSB_IRQ_CLEAR();
    USBIRQ = bmHSGRANT;
}
void ISR_Ep0ack(void) interrupt 0
{
}
void ISR_Stub(void) interrupt 0
{
}
void ISR_Ep0in(void) interrupt 0
{
}
void ISR_Ep0out(void) interrupt 0
{
}
void ISR_Ep1in(void) interrupt 0
{
}
void ISR_Ep1out(void) interrupt 0
{
}

// ISR_Ep2inout is called on every OUT packet receieved.
// We don't do anything with the data.  We just indicate we are done with the buffer.
void ISR_Ep2inout(void) interrupt 0
{
  // Perform USB activity based upon the Alt. Interface selected
   switch (AlternateSetting)
   {
     case Alt1_BulkOUT:
     case Alt4_IsocOUT:
       // check EP2 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is empty
       if(!(EP2468STAT & bmEP2EMPTY))
       {
         EP2BCL = 0x80;          // re(arm) EP2OUT
       }
     break;

     case Alt2_BulkINOUT:
     case Alt6_IsocINOUT:
       // check EP6 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is empty
       if(!(EP2468STAT & bmEP6EMPTY))
       {
```

```
                    EP6BCL = 0x80;          // re(arm)
EP6OUT
                }
              break;
          }

      }
      void ISR_Ep4inout(void)
interrupt 0
      {
      }
      void ISR_Ep6inout(void)
interrupt 0
      {
      }
      void ISR_Ep8inout(void)
interrupt 0
      {
      }
      void ISR_Ibn(void) interrupt 0
      {
      }
      void ISR_Ep0pingnak(void)
interrupt 0
      {
      }
      void ISR_Ep1pingnak(void)
interrupt 0
      {
      }
      void ISR_Ep2pingnak(void)
interrupt 0
      {
      }
      void ISR_Ep4pingnak(void)
interrupt 0
      {
      }
      void ISR_Ep6pingnak(void)
interrupt 0
      {
      }
      void ISR_Ep8pingnak(void)
interrupt 0
      {
      }
      void ISR_Errorlimit(void)
interrupt 0
      {
      }
      void ISR_Ep2piderror(void)
interrupt 0
      {
      }
      void ISR_Ep4piderror(void)
interrupt 0
      {
      }
```

```
        void ISR_Ep6piderror(void)
interrupt 0
        {
        }
        void ISR_Ep8piderror(void)
interrupt 0
        {
        }
        void ISR_Ep2pflag(void)
interrupt 0
        {
        }
        void ISR_Ep4pflag(void)
interrupt 0
        {
        }
        void ISR_Ep6pflag(void)
interrupt 0
        {
        }
        void ISR_Ep8pflag(void)
interrupt 0
        {
        }
        void ISR_Ep2eflag(void)
interrupt 0
        {
        }
        void ISR_Ep4eflag(void)
interrupt 0
        {
        }
        void ISR_Ep6eflag(void)
interrupt 0
        {
        }
        void ISR_Ep8eflag(void)
interrupt 0
        {
        }
        void ISR_Ep2fflag(void)
interrupt 0
        {
        }
        void ISR_Ep4fflag(void)
interrupt 0
        {
        }
        void ISR_Ep6fflag(void)
interrupt 0
        {
        }
        void ISR_Ep8fflag(void)
interrupt 0
        {
        }
```

```
        void ISR_GpifComplete(void)
interrupt 0
        {
        }
        void ISR_GpifWaveform(void)
interrupt 0
        {
        }

        //   File:      fw.c
        //-----------------------------------------------
---------------------------
        #include "fx2.h"
        #include "fx2regs.h"

        //-----------------------------------------------
---------------------------
        // Constants
        //-----------------------------------------------
---------------------------
        #define DELAY_COUNT   0x9248*8L  //
Delay for 8 sec at 24Mhz, 4 sec at 48
        #define _IFREQ  48000           // IFCLK
constant for Synchronization Delay
        #define _CFREQ  48000           //
CLKOUT constant for Synchronization Delay

        //-----------------------------------------------
---------------------------
        // Random Macros
        //-----------------------------------------------
---------------------------
        #define   min(a,b) (((a)<(b))?(a):(b))
        #define   max(a,b) (((a)>(b))?(a):(b))

        // Registers which require a
synchronization delay, see section 15.14
        // FIFORESET        FIFOPINPOLAR
        // INPKTEND         OUTPKTEND
        // EPxBCH:L         REVCTL
        // GPIFTCB3         GPIFTCB2
        // GPIFTCB1         GPIFTCB0
        // EPxFIFOPFH:L
EPxAUTOINLENH:L
        // EPxFIFOCFG       EPxGPIFFLGSEL
        // PINFLAGSxx       EPxFIFOIRQ
        // EPxFIFOIE        GPIFIRQ
        // GPIFIE           GPIFADRH:L
        // UDMACRCH:L       EPxGPIFTRIG
        // GPIFTRIG

        // Note: The pre-REVE EPxGPIFTCH/L
register are affected, as well...
        //      ...these have been replaced by
GPIFTC[B3:B0] registers

        #include "fx2sdly.h"            // Define
_IFREQ and _CFREQ above this #include
```

```
//-----------------------------------------------
---------------------------
// Global Variables
//-----------------------------------------------
---------------------------
volatile BOOL   GotSUD;
BOOL     Rwuen;
BOOL     Selfpwr;
volatile BOOL   Sleep;              // Sleep
mode enable flag

WORD   pDeviceDscr;   // Pointer to
Device Descriptor; Descriptors may be moved
WORD   pDeviceQualDscr;
WORD   pHighSpeedConfigDscr;
WORD   pFullSpeedConfigDscr;
WORD   pConfigDscr;
WORD   pOtherConfigDscr;
WORD   pStringDscr;

//-----------------------------------------------
---------------------------
// Prototypes
//-----------------------------------------------
---------------------------
void SetupCommand(void);
void TD_Init(void);
void TD_Poll(void);
BOOL TD_Suspend(void);
BOOL TD_Resume(void);

BOOL DR_GetDescriptor(void);
BOOL DR_SetConfiguration(void);
BOOL DR_GetConfiguration(void);
BOOL DR_SetInterface(void);
BOOL DR_GetInterface(void);
BOOL DR_GetStatus(void);
BOOL DR_ClearFeature(void);
BOOL DR_SetFeature(void);
BOOL DR_VendorCmnd(void);

// this table is used by the epcs macro
const char code
EPCS_Offset_Lookup_Table[] =
{
  0,    // EP1OUT
  1,    // EP1IN
  2,    // EP2OUT
  2,    // EP2IN
  3,    // EP4OUT
  3,    // EP4IN
  4,    // EP6OUT
  4,    // EP6IN
  5,    // EP8OUT
  5,    // EP8IN
};
```

```
        // macro for generating the address of an
endpoint's control and status register (EPnCS)
        #define epcs(EP)
(EPCS_Offset_Lookup_Table[(EP & 0x7E) | (EP >
128)] + 0xE6A1)

        //-----------------------------------------------
---------------------------
        // Code
        //-----------------------------------------------
---------------------------

        // Task dispatcher
        void main(void)
        {
          DWORD   i;
          WORD   offset;
          DWORD   DevDescrLen;
          DWORD   j=0;
          WORD   IntDescrAddr;
          WORD   ExtDescrAddr;

          // Initialize Global States
          Sleep = FALSE;          // Disable
sleep mode
          Rwuen = FALSE;          // Disable
remote wakeup
          Selfpwr = FALSE;        // Disable self
powered
          GotSUD = FALSE;         // Clear
"Got setup data" flag

          // Initialize user device
          TD_Init();

          // The following section of code is used
to relocate the descriptor table.
          // Since the SUDPTRH and SUDPTRL
are assigned the address of the descriptor
          // table, the descriptor table must be
located in on-part memory.
          // The 4K demo tools locate all code
sections in external memory.
          // The descriptor table is relocated by the
frameworks ONLY if it is found
          // to be located in external memory.
          pDeviceDscr = (WORD)&DeviceDscr;
          pDeviceQualDscr =
(WORD)&DeviceQualDscr;
          pHighSpeedConfigDscr =
(WORD)&HighSpeedConfigDscr;
```

```
            pFullSpeedConfigDscr =
(WORD)&FullSpeedConfigDscr;
            pStringDscr = (WORD)&StringDscr;

            if ((WORD)&DeviceDscr & 0xe000)
            {
              IntDescrAddr =
INTERNAL_DSCR_ADDR;
              ExtDescrAddr =
(WORD)&DeviceDscr;
              DevDescrLen = (WORD)&UserDscr -
(WORD)&DeviceDscr + 2;
              for (i = 0; i < DevDescrLen; i++)
                *((BYTE xdata *)IntDescrAddr+i) =
0xCD;
              for (i = 0; i < DevDescrLen; i++)
                *((BYTE xdata *)IntDescrAddr+i) =
*((BYTE xdata *)ExtDescrAddr+i);
              pDeviceDscr = IntDescrAddr;
              offset = (WORD)&DeviceDscr -
INTERNAL_DSCR_ADDR;
              pDeviceQualDscr -= offset;
              pConfigDscr -= offset;
              pOtherConfigDscr -= offset;
              pHighSpeedConfigDscr -= offset;
              pFullSpeedConfigDscr -= offset;
              pStringDscr -= offset;
            }

            EZUSB_IRQ_ENABLE();          //
Enable USB interrupt (INT2)
            EZUSB_ENABLE_RSMIRQ();          //
Wake-up interrupt

            // What is INT2 is for USB & INT4 is
for the Slave FIFOs
            INTSETUP |= (bmAV2EN |
bmAV4EN);    // Enable INT 2 & 4 autovectoring

            // I don't think we care about Setup PIDs
only the Setup data; commented out
            // bmSUTOK but we want bmSUDAV.
            USBIE |= bmSUDAV | bmSUTOK |
bmSUSP | bmURES | bmHSGRANT;   // Enable
selected interrupts

            // Global interrupt enable. Controls
masking of all interrupts except USB wakeup
            // (resume). EA = 0 disables all
interrupts except USB wakeup. When EA = 1,
interrupts are
            // enabled or masked by their individual
enable bits.
            EA = 1;              // Enable 8051
interrupts

          #ifndef NO_RENUM
```

```
            // Renumerate if necessary.  Do this by
checking the renum bit.  If it
            // is already set, there is no need to
renumerate.  The renum bit will
            // already be set if this firmware was
loaded from an eeprom.
            if(!(USBCS & bmRENUM))
            {
               EZUSB_Discon(TRUE);   //
renumerate
            }
          #endif

            // unconditionally re-connect.  If we
loaded from eeprom we are
            // disconnected and need to connect.  If
we just renumerated this
            // is not necessary but doesn't hurt
anything
            USBCS &=~bmDISCON;

            // The three LSBs of the Clock Control
Register (CKCON, at SFR location 0x8E) control
the stretch
            // value; stretch values between zero and
seven may be used. A stretch value of zero adds
zero
            // instruction cycles, resulting in MOVX
instructions which execute in two instruction
cycles.
            CKCON = (CKCON&(~bmSTRETCH))
| FW_STRETCH_VALUE; // Set stretch to 0 (after
renumeration)

            // clear the Sleep flag.
            Sleep = FALSE;
                  EZUSB_InitI2C();
                  // Initialize EZ-USB I2C
controller

            // Task Dispatcher
            while(TRUE)            // Main Loop
            {
              if(GotSUD)          // Wait for SUDAV
              {
                SetupCommand();         // Implement
setup command
                  GotSUD = FALSE;           // Clear
SUDAV flag
              }

            // Poll User Device
            // NOTE: Idle mode stops the
processor clock.  There are only two
            // ways out of idle mode, the
WAKEUP pin, and detection of the USB
```

```
            // resume state on the USB bus.  The
timers will stop and the
            // processor will not wake up on any
other interrupts.
            if (Sleep)
            {
               if(TD_Suspend())
               {
                  Sleep = FALSE;          // Clear
the "go to sleep" flag.  Do it here to prevent any
race condition between wakeup and the next sleep.
                  do
                  {
                     EZUSB_Susp();        // Place
processor in idle mode.
                  }
                  while(!Rwuen &&
EZUSB_EXTWAKEUP());
                   // Must continue to go back into
suspend if the host has disabled remote wakeup
                   // *and* the wakeup was caused
by the external wakeup pin.

                  // 8051 activity will resume here
due to USB bus or Wakeup# pin activity.
                  EZUSB_Resume();   // If source is
the Wakeup# pin, signal the host to Resume.
                  TD_Resume();
               }
               TD_Poll();
            }
         }

         // Device request parser
         void SetupCommand(void)
         {
            void   *dscr_ptr;


            switch(SETUPDAT[1])
            {
               case SC_GET_DESCRIPTOR:
// *** Get Descriptor
                  if(DR_GetDescriptor())
                     switch(SETUPDAT[3])
                     {
                        case GD_DEVICE:           //
Device
                           SUDPTRH =
MSB(pDeviceDscr);
                           SUDPTRL =
LSB(pDeviceDscr);
                           break;
                        case
GD_DEVICE_QUALIFIER:           // Device
Qualifier
                           SUDPTRH =
MSB(pDeviceQualDscr);
```

```
                    SUDPTRL =
LSB(pDeviceQualDscr);
                    break;
                  case GD_CONFIGURATION:
// Configuration
                    SUDPTRH =
MSB(pConfigDscr);
                    SUDPTRL =
LSB(pConfigDscr);
                    break;
                  case
GD_OTHER_SPEED_CONFIGURATION:  //
Other Speed Configuration
                    SUDPTRH =
MSB(pOtherConfigDscr);
                    SUDPTRL =
LSB(pOtherConfigDscr);
                    break;
                  case GD_STRING:          //
String
                    if(dscr_ptr = (void
*)EZUSB_GetStringDscr(SETUPDAT[2]))
                    {
                      SUDPTRH = MSB(dscr_ptr);
                      SUDPTRL = LSB(dscr_ptr);
                    }
                    else
                      EZUSB_STALL_EP0();   //
Stall End Point 0
                    break;
                  default:          // Invalid request
                    EZUSB_STALL_EP0();     //
Stall End Point 0
                }
              break;
            case SC_GET_INTERFACE:
// *** Get Interface
              DR_GetInterface();
              break;
            case SC_SET_INTERFACE:
// *** Set Interface
              DR_SetInterface();
              break;
            case SC_SET_CONFIGURATION:
// *** Set Configuration
              DR_SetConfiguration();
              break;
            case SC_GET_CONFIGURATION:
// *** Get Configuration
              DR_GetConfiguration();
              break;
            case SC_GET_STATUS:              //
*** Get Status
              if(DR_GetStatus())
                switch(SETUPDAT[0])
                {
                  case GS_DEVICE:          //
Device
```

```
                    EP0BUF[0] = ((BYTE)Rwuen
<< 1) | (BYTE)Selfpwr;
                    EP0BUF[1] = 0;
                    EP0BCH = 0;
                    EP0BCL = 2;
                    break;
                  case GS_INTERFACE:        //
Interface
                    EP0BUF[0] = 0;
                    EP0BUF[1] = 0;
                    EP0BCH = 0;
                    EP0BCL = 2;
                    break;
                  case GS_ENDPOINT:         //
End Point
                    EP0BUF[0] = *(BYTE xdata
*) epcs(SETUPDAT[4]) & bmEPSTALL;
                    EP0BUF[1] = 0;
                    EP0BCH = 0;
                    EP0BCL = 2;
                    break;
                  default:           // Invalid
Command
                    EZUSB_STALL_EP0();      //
Stall End Point 0
                }
              break;
            case SC_CLEAR_FEATURE:
// *** Clear Feature
            if(DR_ClearFeature())
              switch(SETUPDAT[0])
              {
                case FT_DEVICE:           //
Device
                  if(SETUPDAT[2] == 1)
                    Rwuen = FALSE;       //
Disable Remote Wakeup
                  else
                    EZUSB_STALL_EP0();   //
Stall End Point 0
                  break;
                case FT_ENDPOINT:         //
End Point
                  if(SETUPDAT[2] == 0)
                  {
                    *(BYTE xdata *)
epcs(SETUPDAT[4]) &= ~bmEPSTALL;

EZUSB_RESET_DATA_TOGGLE(
SETUPDAT[4] );
                  }
                  else
                    EZUSB_STALL_EP0();   //
Stall End Point 0
                  break;
              }
            break;
```

```
            case SC_SET_FEATURE:
// *** Set Feature
              if(DR_SetFeature())
                switch(SETUPDAT[0])
                {
                  case FT_DEVICE:           //
Device
                    if(SETUPDAT[2] == 1)
                      Rwuen = TRUE;      // Enable
Remote Wakeup
                    else if(SETUPDAT[2] == 2)
                      // Set Feature Test Mode.
The core handles this request.  However, it is
                      // necessary for the firmware
to complete the handshake phase of the
                      // control transfer before the
chip will enter test mode.  It is also
                      // necessary for FX2 to be
physically disconnected (D+ and D-)
                      // from the host before it will
enter test mode.
                      break;
                    else
                      EZUSB_STALL_EP0();   //
Stall End Point 0
                    break;
                  case FT_ENDPOINT:         //
End Point
                    *(BYTE xdata *)
epcs(SETUPDAT[4]) |= bmEPSTALL;
                    break;
                }
              break;
            default:                    // *** Invalid
Command
              if(DR_VendorCmnd())
                EZUSB_STALL_EP0();          //
Stall End Point 0
          }

          // Acknowledge handshake phase of
device request
          EP0CS |= bmHSNAK;
        }

        // Wake-up interrupt handler
        void resume_isr(void) interrupt
WKUP_VECT
        {
          EZUSB_CLEAR_RSMIRQ();
        }
```

APPENDIX B

CODE OF THE MAIN FORM:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.IO;
using CyUSB;
using System.Threading;
```

```
namespace CyControl
{
   public partial class Form1 : Form
   {
```

```
      USBDeviceList usbDevices;
```

```
        CyUSBDevice myDevice;

        CyUSBEndPoint curEndpt;
        App_PnP_Callback evHandler;

        byte AdmDev = CyConst.DEVICES_CYUSB;
        string dataCaption;
        bool progOK = true;
        public string FileNameToBeSaved=null;

        bool ChangeAxColor = false;
        bool ChangeGrColor = false;
        public ColorDialog AxCh;    //To make the dialog result public
        public ColorDialog GrCh;    //They will used in click menu items

        public int keeptime, keepvolt;
        public float timeScale = 1, voltScale = 1;    //Useless scales.

        public int bytes = 512;           //# of bytes for one reading.
        public byte[] buffer = new byte[512];
        public bool bXferCompleted = false;
        int counter = 0;

public Color buttonColor1 = SystemColors.ActiveCaption;
//For easy modification of the button colors.
            public Color buttonColor2 = Color.BlanchedAlmond;
//Will be used.
            public Color buttonColor3 = Color.BurlyWood;
```

```
System.Windows.Forms.Timer MyTimer = new System.Windows.Forms.Timer();
```

```
public Form1()
        {
            InitializeComponent();
            evHandler = new App_PnP_Callback(PnP_Event_Handler);
            usbDevices = new USBDeviceList(AdmDev, evHandler);
            myDevice = (CyUSBDevice)usbDevices[0x04B4, 0x1004];
        //Get the first device having VendorID == 0x04B4 and ProductID ==0x1004

            if (myDevice == null)
            {
                progOK = false;
                myDevice = (CyUSBDevice)usbDevices[0x04B4, 0x8613];
                //Connect the device part which will be programmed.
                //VendorID == 0x04B4 and ProductID == 0x8613
                if (myDevice != null)
                {
                    MessageBox.Show("Cypress connected
now."+Environment.NewLine+"But it should be programmed before usage!" +
Environment.NewLine + "Program the device with 'digos.hex' file", "Info",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    return;
                }

            }
            Form1_Resize(this, null);
        }
```

```
public void PnP_Event_Handler(IntPtr pnpEvent, IntPtr hRemovedDevice)
        {
           if (pnpEvent.Equals(CyConst.DBT_DEVICEREMOVECOMPLETE))
           {
              usbDevices.Remove(hRemovedDevice);     //Warn when device is removed.
                    MessageBox.Show("Cypress device removed.", "Info",
           MessageBoxButtons.OK, MessageBoxIcon.Information);
           }
           if (pnpEvent.Equals(CyConst.DBT_DEVICEARRIVAL))
           {
              usbDevices.Add();       //Warn when device id added.
              MessageBox.Show("Cypress device successfully programmed!" +
Environment.NewLine + "Plot menu is enabled!!!", "Info", MessageBoxButtons.OK,
MessageBoxIcon.Information);
              myDevice = (CyUSBDevice)usbDevices[0x04B4, 0x1004];
              if (myDevice!=null)
                 progOK = true;
        //Now, programmed part of the device is connected.Make progOK=true.
           }
        }
```

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
           if (usbDevices != null) usbDevices.Dispose();
        //Clean up memory, clean the USB devices added to memory.
        }
```

```
private void AboutMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show(Util.Assemblies, Text);
            //Show info about the program.
        }
```

```
private void Periodic_Read(object sender, System.EventArgs e)
        {
            curEndpt = myDevice.EndPointOf(0x86);
//Catch the endpoint of the microcontroller.
            CyBulkEndPoint bulkEpt = curEndpt as CyBulkEndPoint;
            //Define the endpoint of the device.

            if (curEndpt == null)
            {
                MessageBox.Show("Be sure that 'digos' dis connected.");
                //Warn when the device is disconnected.
                return;
            }

            bXferCompleted = bulkEpt.XferData(ref buffer, ref bytes);
            //Send a message when data transfer is completed.
        }
```

```
private void GetData()//Gets one point to plot.
        {
```

```
            MyTimer.Tick += new EventHandler(Periodic_Read)
        //Each tick,each fixed time interval elapsed, call the Periodic_Read to get info from
the device.
            MyTimer.Interval = 1;
        //For now, time interval is fixed at the minimum time interval the Timer function
permit.

            curEndpt = myDevice.EndPointOf(0x86);
        //Catch the endpoint of the microcontroller.

            CyBulkEndPoint bulkEpt = curEndpt as CyBulkEndPoint;
        //Define the endpoint of the device.

            curEndpt.TimeOut = 2000;
        //If there is no response until a fixed time elapsed, warn the user about the problem.

            if (curEndpt == null)
            {
               MessageBox.Show("Be sure that 'digos' dis connected.");
               return;
            }

        //This part is to fill the info output text box to be informed about the data coming.
        //Will be deleted at the end.
            BuildDataCaption();
            OutputBox.Text += dataCaption;
            OutputBox.SelectionStart = OutputBox.Text.Length;
            OutputBox.ScrollToCaret();
```

```
        if (bulkEpt != null)
        {
            bXferCompleted = false;//for each 512.
            bXferCompleted = bulkEpt.XferData(ref buffer, ref bytes);
//Send a message when data transfer is completed.
            MyTimer.Start();
//Start the timer to konuw how many seconds elapsed.

            if (bXferCompleted)
                graphControl1.ReadData(ref buffer, bytes);
//buf and bCnt sent
//When the flag for data transfer complete stage is enabled,
//send the data to ReadData function for plotting.

        }
//For isynchoronous data acquisition.
        CyIsocEndPoint isocEpt = curEndpt as CyIsocEndPoint;
        if (isocEpt != null)
            bXferCompleted = isocEpt.XferData(ref buffer, ref bytes);

        DisplayInfoWriteFile(buffer, bytes);
//To fill the output text file with the data coming.
//To enable the data save function.

    }
```

```
private void BuildDataCaption()
```

```
    {
        StringBuilder dataStr = new StringBuilder();

        switch (curEndpt.Attributes)
        {
            case 0: dataStr.Append("CONTROL ");
                break;
            case 1: dataStr.Append("ISOC ");
                break;
            case 2: dataStr.Append("BULK ");
                break;
            case 3: dataStr.Append("INTERRUPT ");
                break;
        }

        if (curEndpt.bIn)
            dataStr.Append("IN transfer ");
        else
            dataStr.Append("OUT transfer ");

        dataCaption = dataStr.ToString();
    }
```

```
public void DisplayInfoWriteFile(byte[] buf, int bCnt)
    {
        StringBuilder dataStr = new StringBuilder();
        string resultStr = "";
        if (bCnt > 0)
```

```
            resultStr = dataCaption + "completed\r\n";
        else
            resultStr = dataCaption + "failed\r\n";

//When save file item clicked in the menu, this will be activated.
        if(FileNameToBeSaved!=null)
using(StreamWriter myWriter = new StreamWriter(FileNameToBeSaved))
        {
            for(int j=0;j<bCnt;j++)
            myWriter.WriteLine((int)buf[j]);

        }

//Prepare the text file which will be written in the output text file.
        for (int i = 0; i < bCnt; i++)
            {
                if ((i % 16) == 0) dataStr.Append(string.Format("\r\n{0:X4}", i));
                dataStr.Append(string.Format(" {0:X2}", buf[i]));
            }

//Fill the output text on the form.
        OutputBox.Text += dataStr.ToString() + "\r\n" + resultStr + \r\n";
OutputBox.SelectionStart = OutputBox.Text.Length;
OutputBox.ScrollToCaret();
    }
```

```
private void Form1_Resize(object sender, EventArgs e)
    {
```

```
        bool bControlEpt = ((curEndpt != null) && (curEndpt.Attributes == 0));
        int oBoxAdj = bControlEpt ? 200 : 100;
        OutputBox.SetBounds(0, 0, 5, XferTab.Size.Height - oBoxAdj);
    }
```

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Close();
        //Exit the application.
    }
```

```
private void ProgE2Item_Click(object sender, EventArgs e)
    {//Function to program the fX2 device using the .hex file.

        CyFX2Device fx2 = myDevice as CyFX2Device;
//Take the adress of the FX2 device using the adress of the connected USB device.

        string tmpFilter = FOpenDialog.Filter;
        if (sender == ProgE2Item)
            FOpenDialog.Filter = "Hex Files (*.hex) | *.hex";

 if ((fx2 != null) && (FOpenDialog.ShowDialog() == DialogResult.OK))
            {
                bool bResult = false;

                if (sender == ProgE2Item)
                    {
```

```
            StatLabel.Text = "Programming EEPROM of " + fx2.FriendlyName;
            Refresh();
            bResult = fx2.LoadEEPROM(FOpenDialog.FileName);
            }

        else
            {
            StatLabel.Text = "Programming RAM of " + fx2.FriendlyName;
            Refresh();
            bResult = fx2.LoadRAM(FOpenDialog.FileName);
            }

        StatLabel.Text = "Programming " + (bResult ? "succeeded." : "failed.");
        Refresh();
    }

    FOpenDialog.Filter = tmpFilter;
}
```

```
private void HaltItem_Click(object sender, EventArgs e)
{
  //Call this function to halt the data write process.
  CyFX2Device fx2 = myDevice as CyFX2Device;

  if (fx2 != null)
    if (sender == HaltItem)
      fx2.Reset(1);
    else
```

```
            fx2.Reset(0);
        }
```

```
    private void graphControl1_Load(object sender, EventArgs e)
        {
            hideAllGroupBoxes();

            //for easy color change of the buttons
            btnAcquire.BackColor = buttonColor1;
            btnCursor.BackColor = buttonColor1;
            btnDisplay.BackColor = buttonColor1;
            btnMeasure.BackColor = buttonColor1;
            btnResume.BackColor = buttonColor1;
            btnStop.BackColor = buttonColor1;
        }
```

```
    private void AcNormal_Click(object sender, EventArgs e)
        {
            graphControl1.Clear = false;
            graphControl1.StartApp = true;
            GetData();
    //Get the data from the device.
        }
```

```
    public void hideAllGroupBoxes()
```

```
//Hide all of the groupboxes.
        {
            groupBoxAcquire.Hide();  //menu
            groupBoxCursor.Hide();    //menu
            groupBoxDisplay.Hide();  //menu
            groupBoxDiInformation.Hide();      //comenu
            groupBoxAcInformation.Hide();     //comenu
            groupBoxCuInformation.Hide();     //comenu
            groupBoxGridAxis.Hide();            //submenu
            groupBoxScale.Hide();               //submenu
            groupBoxDiTypee.Hide();             //cosubmenu
            groupBoxDiAccumulate.Hide();      //cosubmenu
            groupBoxMeOneInformation.Hide();       //comenu
            groupBoxMeTwoInformation.Hide();       //comenu
        }
```

```
private void btnAcquire_Click(object sender, EventArgs e)
        {
            hideAllGroupBoxes();
//Hide previously used groupboxes.
            groupBoxAcquire.Show();
//Make the acquire group visible.
            groupBoxAcInformation.Show();
        }
```

```
private void btnCursor_Click(object sender, EventArgs e)
        {
```

```
            hideAllGroupBoxes();
    //Hide previously used groupboxes.
            groupBoxCursor.Show();
    //Make the cursor group visible.
            groupBoxCuInformation.Show();
        }
```

```
    private void btnDisplay_Click(object sender, EventArgs e)
        {
            hideAllGroupBoxes();
    //Hide previously used group boxes.
            groupBoxDisplay.Show();
    //Make the display group visible
            groupBoxDiInformation.Show();
    //Simultaneously show the info box
        }
```

```
    private void DiGridAxis_Click(object sender, EventArgs e)
        {
            hideAllGroupBoxes();
    //Hide previously used group boxes.
            groupBoxDisplay.Show();
            groupBoxGridAxis.Show();
    //Show submenu of comenu info of display
        }
```

```
private void DiScale_Click(object sender, EventArgs e)
    {
        hideAllGroupBoxes();
//Hide previously used group boxes.
        groupBoxDisplay.Show();
        groupBoxScale.Show();
//Show submenu of comenu info of display
    }
```

```
private void DiRefresh_Click(object sender, EventArgs e)
    {
        graphControl1.Clear=true;
        Invalidate();           //Activate the OnPaint function.
    }
```

```
private void checkBoxShowAxis_CheckedChanged(object sender, EventArgs e)
    {
        if (ChangeAxColor)
            graphControl1.AxPenColorStatus(AxCh.Color);
        //Immedeately change the color of the axis pen, send the realted data.

        graphControl1.SetAxisStatus(checkBoxShowAxis.Checked);
        //Send the data that shows the checkbox status change.
    }
```

```
private void checkBoxShowGrid_CheckedChanged(object sender, EventArgs e)
```

```
        {
            if (ChangeGrColor)
                graphControl1.GrPenColorStatus(GrCh.Color);
            //Immedeately change the color of the grid pen, send the realted data.

            graphControl1.SetGridStatus(checkBoxShowGrid.Checked);
            //Send the data that shows the checkbox status change.
        }
```

```
private void DiType_Click(object sender, EventArgs e)
        {
            hideAllGroupBoxes();
//Hide previously used group boxes.
            groupBoxDisplay.Show();
            groupBoxDiTypee.Show();
        }
```

```
private void DiSum_Click(object sender, EventArgs e)
        {
            hideAllGroupBoxes();
//Hide previously used group boxes.
            groupBoxDisplay.Show();

        }
```

```
private void checkedListBox1_SelectedIndexChanged(object sender,EventArgs e)
        {
            keeptime = checkedListBox1.SelectedIndex;
//Uncheck other indices
            for (int i = 0; i < checkedListBox1.Items.Count; i++)
                checkedListBox1.SetItemChecked(i, false);
            checkedListBox1.SetItemChecked(keeptime, true);
//Keep selected index.
            scaling();
//Send the necessary scaling values to the related places.
        }
```

```
private void checkedListBox2_SelectedIndexChanged(object sender, EventArgs e)
        {
            keepvolt = checkedListBox2.SelectedIndex;
//Uncheck other indices
            for (int i = 0; i < checkedListBox2.Items.Count; i++)
                checkedListBox2.SetItemChecked(i, false);
            checkedListBox2.SetItemChecked(keepvolt, true);
//Keep selected index.
            scaling();
//Send the necessary scaling values to the related places.
        }
```

```
private void scaling()
        {
            graphControl1.DispType=false;
```

```
        if (comboBox1.SelectedIndex == 0)
        {
            graphControl1.DispType=true;
        }

switch (checkedListBox1.SelectedIndex)
//Set the scale, will be editted acc. to the length of the table.
        {
            case 0:
                timeScale = 2;
                break;
            case 1:
                timeScale = 3;
                break;
            case 2:
                timeScale = 4;
                break;
            case 3:
                timeScale = 5;
                break;
            case 4:
                timeScale = 6;
                break;
            case 5:
                timeScale = 7;
                break;
            case 6:
                timeScale = 8;
```

```
    break;
case 7:
    timeScale = 9;
    break;
case 8:
    timeScale = 10;
    break;
case 9:
    timeScale = 11;
    break;
case 10:
    timeScale = 12;
    break;
case 11:
    timeScale = 13;
    break;
case 12:
    timeScale = 14;
    break;
case 13:
    timeScale = 15;
    break;
case 14:
    timeScale = 16;
    break;
case 15:
    timeScale = 17;
    break;
case 16:
```

```
        timeScale = 18;
        break;
    case 17:
        timeScale = 19;
        break;

}

switch (checkedListBox2.SelectedIndex)
{
    case 0:
        voltScale = 2;
        break;
    case 1:
        voltScale = 3;
        break;
    case 2:
        voltScale = 4;
        break;
    case 3:
        voltScale = 5;
        break;
    case 4:
        voltScale = 6;
        break;
    case 5:
        voltScale = 7;
        break;
    case 6:
```

```
    voltScale = 8;
    break;
case 7:
    voltScale = 9;
    break;
case 8:
    voltScale = 10;
    break;
case 9:
    voltScale = 11;
    break;
case 10:
    voltScale = 12;
    break;
case 11:
    voltScale = 13;
    break;
case 12:
    voltScale = 14;
    break;
case 13:
    voltScale = 15;
    break;
case 14:
    voltScale = 16;
    break;
case 15:
    voltScale = 17;
    break;
```

```
            case 16:
                voltScale = 18;
                break;
            case 17:
                voltScale = 19;
                break;

        }

        graphControl1.TimeScale = timeScale;

//Send the scale values to the graphcontrol part.
        graphControl1.VoltScale = voltScale;
        Invalidate();
//Send interrupt to the OnPaint function.
    }
```

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (comboBox1.SelectedIndex == 0)
//Vector draw mode ON
            graphControl1.DispType=true;
        else if (comboBox1.SelectedIndex == 1)
//Vector draw mode OFF
            graphControl1.DispType=false;
        Invalidate();
//Send interrupt to the OnPaint function.
    }
```

```
private void btnClear_Click(object sender, EventArgs e)
    {
        graphControl1.Clear = true;
//Clear the screen.
        Invalidate();
//Send interrupt to the OnPaint function.
    }
```

```
private void btnFreeze_Click(object sender, EventArgs e)
    {
        graphControl1.freeze = true;
        //Freeze the data flow on the screen.
    }
private void btnResume_Click(object sender, EventArgs e)
    {
        graphControl1.freeze = false;
        //Activate the data flow on the screen.
    }
```

```
private void btnStop_Click(object sender, EventArgs e)
    {
        //Stop the data acquisition and the data flow on the screen.
    }
```

```
    }
}
```

CODE OF THE GRAPH CONTROL PART

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.IO;
```

```
namespace digos2
{
    public partial class GraphControl : UserControl
    {
```

```
        public bool ShowGrid = true;
//Default values of showgrid
```

```
        public bool ShowAxis = false;
//and show axis are off.
        public bool StartApp = false;
        public bool Clear = false;
        public bool UpScale = false;
        public bool DispType = false;
//Vector or dot drawing

        public bool freeze = false;
        long currentDot = 0;
        int timeAx;
        int myIndex;
        public int NumberOfPoints=600;
        public float TimeScale=1,VoltScale=1;
//For scaling Real scales

        public struct CoOrds
//Decleration of the struct type for the coordinates.
        {
                public float x, y;
                public CoOrds(float p1, float p2)
                        {
                                x = p1;
                                y = p2;
                        }

}
```

```
Pen MyAxPen = new Pen(Color.Azure);
Pen MyGrPen = new Pen(Color.Lime);
Pen MyDrawPen = new Pen(Color.Red, 2);
CoOrds[] MyData = new CoOrds[600];
//Holds all data for one screen view.
```

```
public GraphControl()
{
   InitializeComponent();
}
```

```
public void SetGridStatus(bool Status)
{
   ShowGrid = Status;
   Invalidate();
}
```

```
public void SetAxisStatus(bool Status)
{
   ShowAxis = Status;
   Invalidate();
}
```

```
public void AxPenColorStatus(Color AxPenColor)
    {
       MyAxPen.Color = AxPenColor;
    }
```

```
    public void GrPenColorStatus(Color GrPenColor)
    {
       MyGrPen.Color = GrPenColor;
    }
```

```
    public void ReadData(ref byte[] buf, int bCnt)
        {
    //Takes one bulk(512 items),writes its average to the last point, shifts the graph to the
left.

          myIndex = 1;//Since we use "myIndex-1" in the array.
          while (myIndex < NumberOfPoints)
          {
             currentDot = 0;
    //Since wee add the buffer items up, we start from 0 for each cycle.

             for (int i = 0; i < bCnt; i+=8)
    //Get the average of some of the data of 1024 bytes => 512 item.
                currentDot = currentDot + (480 * (buf[i] / 255));
    //480/FF:Normalize the data.
```

```
            MyData[myIndex - 1].y = MyData[myIndex].y;
//Shift the values in the array to the left.
            myIndex++;
//Go along the array.
        }
        MyData[NumberOfPoints].y = currentDot/128;
//Our new point.
        Invalidate();
//call the OnPaint method to refresh the graph.


    }
protected override void OnPaint(PaintEventArgs e)
    {

        base.OnPaint(e);

        e.Graphics.TranslateTransform(0, this.Height );
        e.Graphics.ScaleTransform(1, -1);//Translate the origin of the area



        if (UpScale)
            StartApp = true;

        if (Clear)
            StartApp = false;



        if (ShowGrid)
```

```
        {
            //Horizontal Lines
            e.Graphics.DrawLine(MyGrPen, new PointF(0, 0 * this.Height / 8), new
PointF(this.Width, 0 * this.Height / 8));
            e.Graphics.DrawLine(MyGrPen, new PointF(0, 1 * this.Height / 8), new
PointF(this.Width, 1 * this.Height / 8));
            e.Graphics.DrawLine(MyGrPen, new PointF(0, 2 * this.Height / 8), new
PointF(this.Width, 2 * this.Height / 8));
            e.Graphics.DrawLine(MyGrPen, new PointF(0, 3 * this.Height / 8), new
PointF(this.Width, 3 * this.Height / 8));

            e.Graphics.DrawLine(MyGrPen, new PointF(0, 5 * this.Height / 8), new
PointF(this.Width, 5 * this.Height / 8));
            e.Graphics.DrawLine(MyGrPen, new PointF(0, 6 * this.Height / 8), new
PointF(this.Width, 6 * this.Height / 8));
            e.Graphics.DrawLine(MyGrPen, new PointF(0, 7 * this.Height / 8), new
PointF(this.Width, 7 * this.Height / 8));
            e.Graphics.DrawLine(MyGrPen, new PointF(0, 8 * this.Height / 8), new
PointF(this.Width, 8 * this.Height / 8));

            //Vertical Lines
            e.Graphics.DrawLine(MyGrPen, new PointF(0 * this.Width / 10, 0), new
PointF(0 * this.Width / 10, this.Height));
            e.Graphics.DrawLine(MyGrPen, new PointF(4 * this.Width / 10, 0), new
PointF(4 * this.Width / 10, this.Height ));
            e.Graphics.DrawLine(MyGrPen, new PointF(3 * this.Width / 10, 0), new
PointF(3 * this.Width / 10, this.Height ));
            e.Graphics.DrawLine(MyGrPen, new PointF(2 * this.Width / 10, 0), new
PointF(2 * this.Width / 10, this.Height ));
```

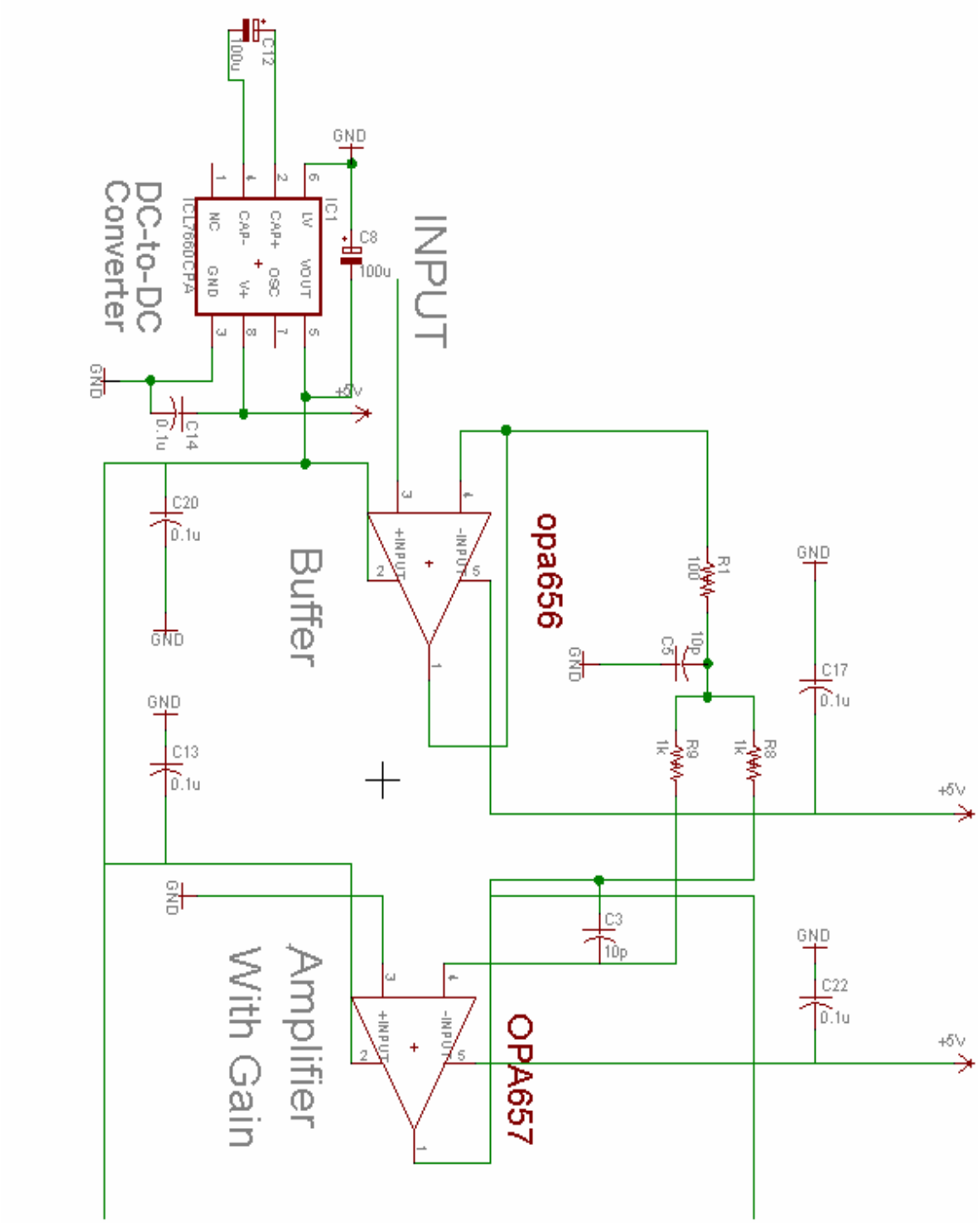
```
                e.Graphics.DrawLine(MyGrPen, new PointF(1 * this.Width / 10, 0), new
PointF(1 * this.Width / 10, this.Height ));

                e.Graphics.DrawLine(MyGrPen, new PointF(6 * this.Width / 10, 0), new
PointF(6 * this.Width / 10, this.Height ));
                e.Graphics.DrawLine(MyGrPen, new PointF(7 * this.Width / 10, 0), new
PointF(7 * this.Width / 10, this.Height ));
                e.Graphics.DrawLine(MyGrPen, new PointF(8 * this.Width / 10, 0), new
PointF(8 * this.Width / 10, this.Height ));
                e.Graphics.DrawLine(MyGrPen, new PointF(9 * this.Width / 10, 0), new
PointF(9 * this.Width / 10, this.Height ));
                e.Graphics.DrawLine(MyGrPen, new PointF(10 * this.Width / 10, 0), new
PointF(10 * this.Width / 10, this.Height ));

                //Axis Lines
                e.Graphics.DrawLine(MyGrPen, new PointF(0, 4 * this.Height / 8), new
PointF(this.Width, 4 * this.Height / 8));
                e.Graphics.DrawLine(MyGrPen, new PointF(5 * this.Width / 10, 0), new
PointF(5 * this.Width / 10, this.Height));

            }



        if (ShowAxis)
            {
                e.Graphics.DrawLine(MyAxPen, new Point(this.Width / 2, 0), new
Point(this.Width / 2, this.Height));
```

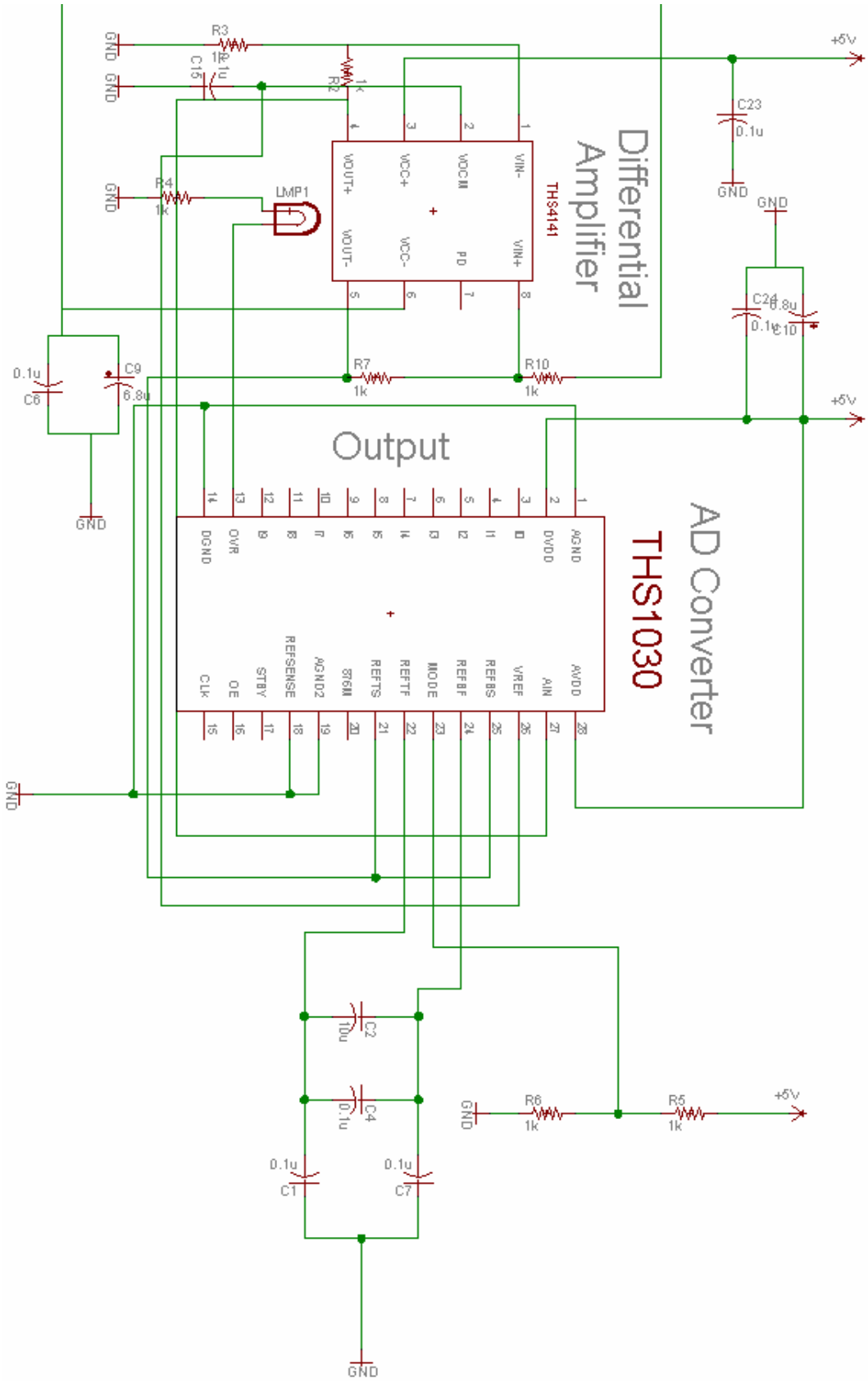
```
                e.Graphics.DrawLine(MyAxPen, new Point(0, this.Height / 2), new
Point(this.Width, this.Height / 2));

            }


            if (StartApp & DispType)
                for (timeAx = 0; timeAx < NumberOfPoints;timeAx++ )
                    e.Graphics.DrawEllipse(MyDrawPen, timeAx / TimeScale,
(MyData[timeAx].y) / VoltScale, 1, 1);

            if (StartApp)
                for (timeAx = 0; timeAx < NumberOfPoints-1; timeAx++)
        //connect the points
                    e.Graphics.DrawLine(MyDrawPen, timeAx / TimeScale,
MyData[timeAx].y / VoltScale, (timeAx + 1) / TimeScale, MyData[timeAx + 1].y /
VoltScale);


        }
```

```
    }
}
```

APPENDIX C

SCHEMATIC


DC-to-DC
Converter
TCL7660CPA
IC1
C12
100u
C8
100u
GND
+5v
C14
0.1u
INPUT
Buffer
C20
0.1u
opa656
+INPUT
-INPUT
R1
100
C5
10p
C17
0.1u
R9
1k
R8
1k
C13
0.1u
Amplifier
With Gain
C3
10p
OPA657
C22
0.1u


Differential
Amplifier
THS4141
VOUT+
VCC+
VOCM
VIN-
VOUT-
VCC-
PD
VIN+
LMP1
R3
C15
R2
R4
C23
C24
C10
R7
R10
C6
C9
Output
AD Converter
THS1030
DGND
OVR
DVDD
AGND
CLK
OE
STBY
REFSENSE
AGND2
STBM
REFTS
REFTF
MODE
REFBF
REFBS
VREF
AIN
AVDD
C2
C4
C1
C7
R6
R5
GND
+5V
0.1u
6.8u
1k
10u

BIBLIOGRAPHY

[1] http://en.wikipedia.org/wiki/USB

[2]http://developer.apple.com/documentation/DeviceDrivers/Conceptual/USBBook/USBOverview/chapter_2_section_3.html

[3] www.usb.org/developers/whitepapers/usb_20t.pdf

[4] http://www.lvr.com/usbcenum.htm

[5] http://www.beyondlogic.org/usbnutshell/usb5.htm

[6] http://www.beyondlogic.org/usbnutshell/usb4.htm#Control