STUDY ON DATA ACQUISITION CARD AND ITS APPLICATION ON THE BASIS OF ANALOG TO DIGITAL CONVERSION

STUDY ON DATA ACQUISITION CARD AND ITS APPLICATION IN SOME SIDES ON THE BASIS OF ANALOG TO DIGITAL CONVERSION

Thesis Submitted to---------------------------------------

Department of Electrical & Electronic Engineering
Bangladesh University of Engineering &Technology
Dhaka-1000, Bangladseh

As a part of completion of B.Sc. Engineering

Session:- "2002-2003"

Md. Nayeem Arafat, student no. --9906005

Mohd. Abu Wasib, student no.-- 9906042

Shamsul Arafin, student no.—9906043

DEDICATION

Dedicated to----------------------

Dr. Mohammad Ali Chowdhury

Head of the Department
Department of Electrical & Electronic Engineering
BUET, Dhaka-1000

with great respect

Acknowledgements

Firstly we would like to thank our respected Thesis and Project Teacher **Mohammad Ali Chowdhury** for his constant guidance, helpful suggestion, constructive criticism and endless patience throughout the development of this thesis.

We would also thank to **Mr. Sanaullah** and **Mr. Debu**, Laboratory assistant of Power Electronic Laboratory and Electrical Machine Laboratory respectively.

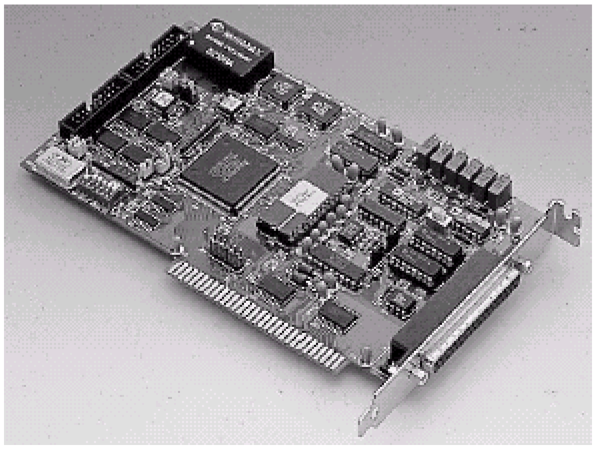
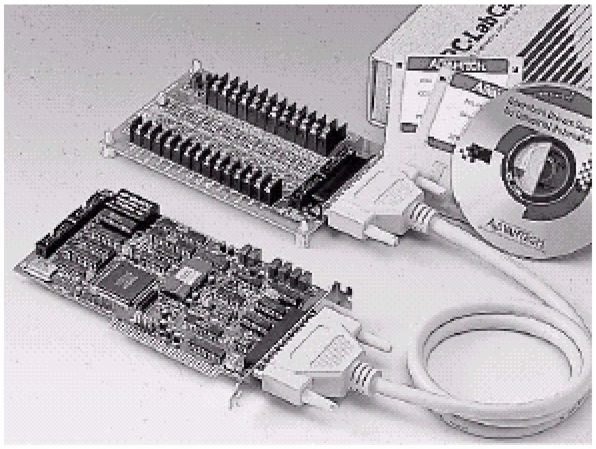
Finally ALL PRAISES TO ALLAH.

PREFACE

In this thesis we have tried to show some applications of DATA ACQUISITION CARD on the basis of analog to digital conversion scheme. Here we have used PCL-818HG, high performance with programmable gain DAS card. There are plenty of schemes to be implemented by using this card. Analog to Digital conversion is one of them.
Mainly this thesis is the outgrowth of our experience in learning the usage of data acquisition card.

PCL-818HG

High Performance
DAS Card with Programmable Gain

Why DAS (DATA ACQUISITION SYSTEM):::::::::

Data acquisition systems via analog signals are used in communication, electronic and medical applications. Conversions to digitized systems is widely used today because complex circuits are low cost, accurate, simple to implement. Data Acquisition systems are mainly used to measure and record analog signals.

What is DAC (DATA ACUISITION CARD):::::::::::::::::

To implement the data acquisition systems a device is needed by which plenty of conditions and requirements of DAS is served. That device is called DAC.

What is PCL-818HG:::::::::::::::::::::::::

The PCL-818HG is a very-high-speed, high-performance multifunction plug-in DAS card for the IBM PC/AT and compatible computers. It features a 100 KHz 12-bit analog-to-digital converter, on board 1 K word FIFO buffer, two 12-bit D/A output channels, 16 digital input channels, 16 digital output channels and one 16-bit counter channel.

A programmable-gain instrument amplifier (x0.5, 1.5,10,50,100,500,or1000) lets you acquire very low level input signals without external signal conditioning.
The PCL-818HG also includes a 16-channel, 8-bit analog comparator which you can use as an analog watchdog to monitor the card's 16 analog input signals. If a signal goes above or below a triggering value, the PCL-818HG can generate an interrupt and transfer data. The card provides DMA data transfer on one of its two 12-bit D/A output channels. This channel lets you perform synchronous data output at up to 100 kHz, excellent for waveform generation. All these sophisticated functions make the PCL-818HG an ideal data acquisition system for your laboratory applications, which require very-high speed and powerful triggering capabilities.

The PCL-818HG is hardware and software compatible with its popular processors, the PCL-818. This puts rich software support and a wide variety of external signal conditioning boards at your disposal.

Features::::

• 100 KHz 12-bit A/D converter
• 16 single-ended or eight differential inputs
• Built-in 1 K word FIFO buffer
• 16-channel analog comparator
• 100 KHz data transfer rate with FIFO
• 200 KHz data transfer rate with DMA
• 12-bit analog output with DMA
• Unipolar/bipolar input with programmable gain
• 16 digital inputs and 16 digital outputs
• Furnished with menu-driven scope software package

Specifications:::::

Analog input (A/D converter)
Channels: 16 single ended or 8 differential, switch selectable
Resolution: 12 bits
On board FIFO: 1K words
Input ranges (V_{dc}):
Bipolar:±0.05, ±0.01, ±0.05, ±0.1, ±0.5, ±1, ±5, ±10

Unipolar:0 to 0.01,0to 0.1, 0 to1, 0 to 10
All input ranges are software programmable
Ovrevoltage: Continuous ±30 V max
Conversion Type: Successive approximation
Conversion rate: 100 kHz (depends on gain and input signal
Linearity: ±1 bit
Trigger mode: Software, on board programmable pacer or external
External trigger: TTL compatible
Load is .4 ma max at .5 volt and –0.05 ma at 2.7 volt.
Data transfer: Program, interrupt or DMA.

Analog input::::::::

The PCL-818HG features a 12-bit high-speed A/D converter with 2.5 msec. conversion time. It also includes an on-board sample-and hold circuit with software programmable input range. You can trigger the A/D conversion from your program, the on-board programmable pacer or an external trigger. The PCL-818HG can select either eight differential or 16 single ended analog inputs by switch. Like Advantech's other high-performance cards the PCL-818HG offers auto-channel/gain scanning. This feature allows high-speed multichannel sampling with DMA (up to 200 KHz) and different gain for each channel.

On-board 1 K word FIFO with programmable index:::::

To increase the throughput and improve performance under Windows the PCL-818HG includes a 1 K word FIFO (First In First Out) buffer. A programmable index with one-step resolution lets you acquire anywhere from one to 1024 analog-to-digital-conversion data elements at a time. For example, if you set the FIFO's index to 25, after the card completes 25 A/D conversions, it will generate an interrupt and transfer just 25 data elements. You don't need to wait until the FIFO is full or half-full to receive your data.

Data transfer
The PCL-818HG can transfer the A/D conversion data in four different ways: software polling, interrupt service routine, DMA and FIFO. The method you use determines the speed of the data transfer, as shown below:

Method	**Max. A/D throughput**
Software polling	10-20 KHz (depending on PC speed)
Interrupt	10-30 KHz
DMA	200 KHz
FIFO w/repeat	330 KHz

input string

Analog input

• **Channels**: 16 single ended, eight differential (selectable)
• **Resolution**: 12 bit
• **Conversion time**: 3 msec
• **Input ranges**:
Bipolar: ±10 V, ±5 V, ±2.5 V, ±1.25 V and ±0.625 V
Unipolar: 0~10 V, 0~5 V, 0~2.5 V and 0~1.25 V
• **Automatic channel gain/scan**
• **Trigger modes**:
Software, pacer or external trigger pre-, postand position-triggering
• **Data transfer**:
Program-controlled, interrupt (IRQ 2, 4, 5, 7, 10, 11, 12 or 15) or 16-bit DMA transfer (DRQ 5 to 7) selected by software
• **FIFO size**: 1 K word
• **Data transfer rate**: 200 KHz with DMA 330 KHz with FIFO
• **Accuracy**: 0.01% of FSR ±1 LSB
• **Temperature coefficient**: 25 PPM/oC

General
• **I/O port**: 32 consecutive bytes
• **Bus**: 16-bit ISA (AT)
• **Dimensions**: 4.8" x 8.6" (122 mm x 218 mm)
• **Power consumption**: +5 V @ 600 mA max.;
+12 V @ 200 mA max.; -12 V @ 15 mA typical

Daughterboards:::::

We offer a wide variety of optional daughterboards to help you get the most from your PCl-818HG. You will need the PCLD-774 Analog Expansion Board to make connections.

PCLD-789 Amplifier/Multiplexer Board

This analog input-signal conditioning board multiplexes 16 differential inputs to one A/D input channel. A high-grade instrumentation amplifier provides switch selectable gains of 1, 2, 10, 50, 100, 200,1000.

PCLD-788 Relay Multiplexer Board

This board multiplexes up to 16 differential inputs to one analog output channel. It offers isolated break-before-make high voltage switching and a CJC circuit for thermocouple measurement.

PCLD-787Channel simultaneous sample and hold board:::::

This board lets you simultaneously acquire up to eight analog input with less than 30 nsec of channel-to-channel sample time uncertainty.

HARDWARE INSTALLATION::::::::::

Initial inspection::::

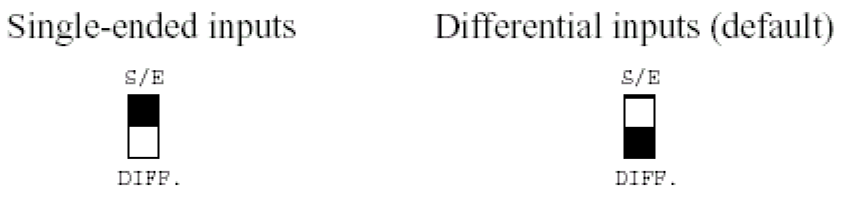
We carefully inspected the PCL-818HG both mechanically and electrically before we shipped it. It should be free of marks and scratches and in perfect order on receipt. As you unpack the PCL-818HG, check it for signs of shipping damage (damaged box, scratches, dents, etc.). It is damaged or fails to meet specifications, notify our service department or your local sales representative immediately. Also, call the carrier immediately and retain the shipping carton and packing material for inspection by the carrier. We will then make arrangements to repair or replace the unit. Discharge any static electricity on your body before you touch the board by touching the back of the system unit (grounded metal). Remove the PCL-818HG card from its protective packaging by grasping the rear metal panel. Handle the card only by its edges to avoid static electric discharge which could damage its integrated circuits. Keep the antistatic package. Whenever you remove the card from the PC, store the card in this package for protection.
You should also avoid contact with materials that hold static electricity such as plastic, vinyl and styrofoam.

Switch and jumper settings
You set the basic configuration of the PCL-818HG through two function switches and four jumpers, as summarized in the following table:

Option	**Setting**
1) Channel configuration, S. E. or diff.	SW1
2) Base address selection	SW2
3) Internal voltage reference, -10 V or -5 V	JP1
4) TRIG0 and GATE0 Selection	JP2
5) D/A reference voltage, int./ext.	JP3, JP4

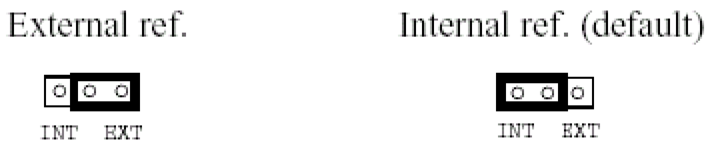
Channel configuration, S. E. or diff. (SW1)::

The PCL-818HG offers 16 single-ended or eight differential analog input channels. SW1 changes the channels between single-ended or differential input. Slide the switch to the upper position, marked DIFF, for eight differential inputs. Slide the switch to the upper position, marked S/E, for 16 single ended inputs. DIFF is the default.



D/A reference voltage, int./ext. (JP2):::::

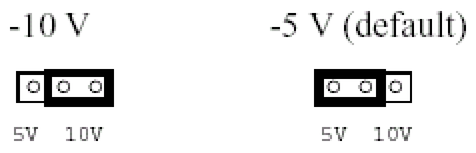
Jumpers JP2 select the reference voltage source for the PCL- 1800's D/A converter channels. You can use the card's internal reference or supply an external reference.



When you set a JP2 to INT, the channel takes itsreference voltage input from the card's on-board reference. Jumper JP1 sets the on-board reference to either -5 V or -10 V. With the jumper set to INT, the D/A channel then has an output range of 0 V to +5 V or 0 V to +10 V.
When you set the channel's jumper to EXT, the D/A converter takes its reference voltage input from connector CN3 pin 31 (for channel 0) or pin 12 (for channel 1). You can apply any voltage between -10 V and +10 V to function as the external reference. The reference input can be either DC or AC (< 100 KHz). When you use an external reference with voltage V_{ref}, you can program the D/A channel to output from 0 V to - V_{ref}. You can also use the D/A converter as a programmable attenuator. The attenuation factor between reference input and analog output is:
Attenuation factor = G / 4095
where G is a value you write to the D/A registers between 0 and 4095. For example, if you set G to 2048, then the attenuation factor is 0.5. A sine wave of 10 V amplitude applied to the reference input will generate a sine wave of 5 V amplitude on the analog output.

Internal voltage reference, -10 V or -5 V (JP3):::::

If you use an internal reference voltage (set with JP2), the PCL-818HG provides a choice of DC internal reference voltage sources: -5V and -10V. JP3 selects the sources, Jumper settings appear below:



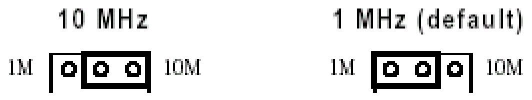
Connector pin assignments:::::

The PCL-818HG has two on-board 20-pin flat-cable connectors (insulation displacement, mass termination) and a DB-37 connector accessible from the card bracket. See the figure in Appendix B for the location of each connector.
Pin assignments for each connector appear in the following sections.

Abbreviations	
A/D S	Analog input (single-ended)
A/D H	Analog input high (differential)
A/D L	Analog input low (differential)
A.GND	Analog ground
D/A	Analog output
D/O	Digital output
D/I	Digital input
D.GND	Digital and power supply ground
CLK	Clock input for the 8254
GATE	Gate input for the 8254
OUT	Signal output of the 8254
VREF	Internal voltage reference
REFIN	External voltage reference input

Timer Clock selection (JP4):::::

Jumper of JP4 controls the input clock frequency for the 8254 programmable clock/timer of the module. You have two choices: 10MHz and 1MHz. This lets you generate pacer output frequencies from 2.5 MHz to 0.00023 Hz (71 minutes/pulse).
The following equation gives the pacer rate:
Pacer rate = Fclk / (Div1 * Div2)
Fclk is 1MHz or 10MHz, is set by JP4 as illustrated above. Div1 and Div2 are dividers set in counter 1 and counter 2 in the Intel 8254 counter.

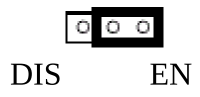


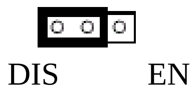
FIFO enable/disable (JP5):::::

When you enable the PCL-818HG’s FIFO (First In First Out) buffer each time the card makes an A/D raeding, it wills tore the data in both the A/D output registers (accessed at addresses BASE +0/1) and in the (accessed at BASE +23/24). When you enable the FIFO the PCL-818HG will require 32 consecutive I/O addresses.

When you disable the FIFO bufefr, you can only access the converted data from the A/D output registers at BASE +0/1. The PCL-818HG will only require consecutive I/O addresses.
JP5 jumper settings apper below:::

FIFO enabled (default) **FIFO disabled**





Hardware installation::::::

Warning! Disconnect power from the PC whenever you install or remove the PCL-818HG or its cables.

Installing the card in your computer::::

1. Turn the computer off. Turn the power off to any peripheral devices such as printers and monitors.
2. Disconnect the power cord and any other cables from the back of the computer.
3. Remove the system unit cover (see the user's guide for your chassis if necessary).
4. Locate the expansion slots at the rear of the unit and choose any unused slot.
5. Remove the screw that secures the expansion slot cover to the system unit. Save the screw to secure the interface card-retaining bracket.
6. Carefully grasp the upper edge of the PCL-818HG card. Align the hole in the retaining bracket with the hole on top of the expansion slot and align the gold striped edge connector with the expansion slot socket. Press the board firmly into the socket.
7. Secure the PCL-818HG using the screw you removed in step 5.
8. Attach any accessories (using DB-37 cable, etc.) to the PCL-818HG.
9. Replace the system unit cover. Connect the cables you removed in step 2. Turn the computer power on.

Hardware installation is now complete.

Software Installation:::::::

The PCL-818HG includes a floppy disk with utility software.
The floppy disk contains:

1.A comprehensive I/O driver for A/D and digital I/O applications. This driver lets you use standard functions, written in common programming languages, to operate the PCl-

818HG. You do not need to perform detailed register programming. The driver supports the following languages: Microsoft Visual Basic, Visual C++, Borland C++, C++ Builder and Delphi. Please refer to the Software Driver User's Manual for more information.

2.Demonstration programs

3. A calibration program

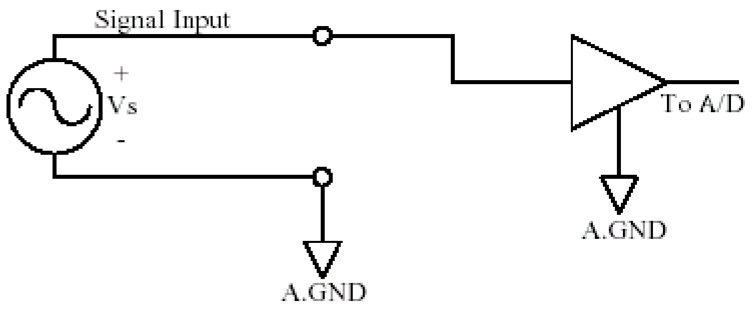
4. A test program

Signal Connections:::::::

Correct signal connections are one of the most important factors to consider if your application is to send and receive data with accuracy. Good signal connections can also avoid a lot of unnecessary damage to your valuable PC and other hardware. This chapter provides information on signal connections for different types of data acquisition applications.

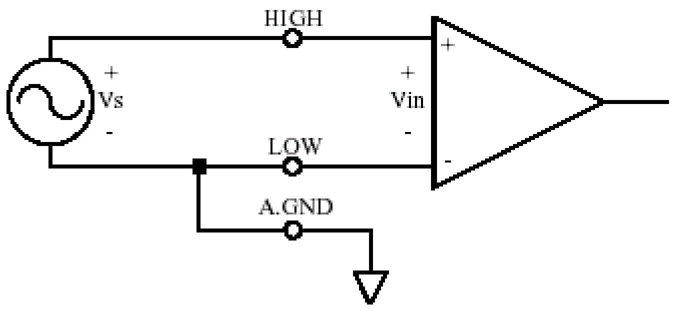
Analog input connections:::::

The PCL-818HG supports either 16 single-ended or 8 differential analog inputs. Switch SW1 selects the input channel configuration. Single-ended channel connections Single-ended connections use only one signal wire per channel. The voltage on the line references to the common ground on the card. A signal source without a local ground is called a "floating" source. It is fairly simple to connect a single ended channel to a floating signal source. A standard wiring diagram looks like this:



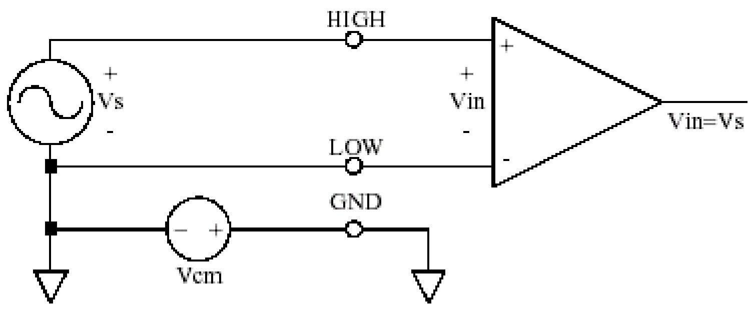
Differential channel connection::::::

Differential input connections use two signal wires per channel. The card measures only the voltage difference between these two wires, the HI wire and the LOW wire. If the signal source has no connection to ground, it is called a "floating" source. A connection must exist between LOW and ground to define a common reference point for floating signal sources. To measure a floating source connect the input channel as shown below:

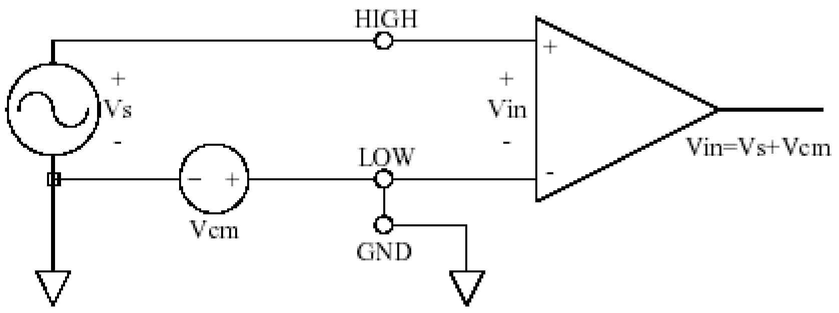


If the signal source has one side connected to a local ground, the signal source ground and the PCL-818HG ground will not be at exactly the same voltage, as they are connected through the ground return of the equipment and building wiring. The difference between the ground voltages forms a common-mode voltage. To avoid the ground loop noise effect caused by common-mode voltages, connect the signal ground to the LOW input. Do not connect the LOW input to the PCL-818HG ground directly. In some cases you may also need a wire connection between the PCL-818HG
ground and the signal source ground for better grounding. The following two diagrams show correct and incorrect connections for a differential input with local ground:

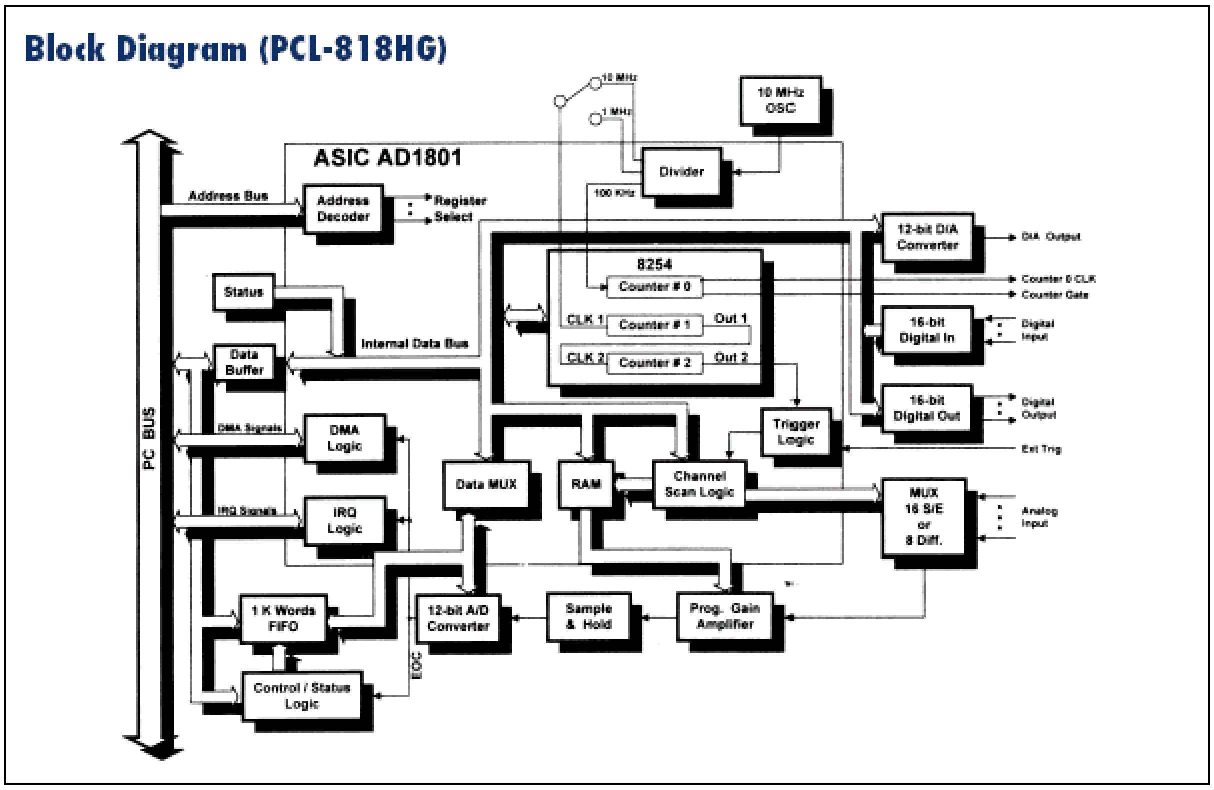
Correct connection::::::



Incorrect connection:::::::



BLOCK DIAGRAM of PCL-818HG::::::



PCL-818 Series Quick-reference Table

Model	A/D speed	Unipolar input (V)	Bipolar input (V)	Onboard memory	D/A chan.	Connector	Size	On-board DC/DC	Power consumption
PCL-818HG	100 KHz	0 – 10, 0 – 1, 0 – 0.1, 0 – 0.01	±10, ±5, ±1, ±0.5, ±0.1, ±0.05, V0.01, ±0.005	1 K samples FIFO	1	DB-37	185 x 100 mm	Yes	<2.8 W

PCL-818HG:::::::(COLOURED VIEW)

Expanding analog inputs:::::

You can expand any or all of the PCL-818HG's A/D input channels using multiplexing daughterboards. Most daughterboards require the PCLD- 774 Analog Expansion Board or the PCLD-8115 Screw Terminal Board for connections.

The PCLD-789 Amplifier and Multiplexer multiplexes 16 differential inputs to one A/D input channel. You can cascade up to eight PCLD-789s to the PCL-818HG for a total of 128 channels.

The PCLD-774 Analog Expansion Board accommodates multiple external signal-conditioning daughterboards, such as PCLD-779 and PCLD-789. It features five sets of on-board 20-pin header connectors. A special star-type architecture lets you cascade multiple signal conditioning boards without the signal-attenuation and current loading problems of normal cascading.

The PCLD-8115 Screw Terminal Board makes wiring connections easy. It provides 20-pin flat cable and DB-37 cable connectors. It also includes CJC (Cold Junction Compensation) circuits, which let you directly measure thermocouples with your PCL-818HG. You can handle all types of thermocouples with software compensation and linearization.

Special circuit pads on the PCLD-8115 accommodate passive signal conditioning components. You can easily implement a low-pass filter, attenuator or current shunt by adding resistors and capacitors.

Register structure and format::::::

This chapter shows the format for each of the PCL-818HG's registers. Register-level programming for the card's more advanced functions, such DMA data transfer or pre-triggering, is extremely complicated. In practice you will need to use the card's software driver. We provide the register layout information only to help you understand what the functions in the software driver are doing. On the other hand, register layout and programming for most of the card's standard functions, such as digital I/O, are the same as for other cards in the PCL-818 Series. Any programs you have written which directly access the registers should work with few modifications. Notes in the text indicate functions which require the software driver. Register format The PCL-818HG requires 32 consecutive addresses in the PC's I/O space. Each address corresponds to a card register. The address of each register is specified as an offset from the card's base address. For example, BASE+0 is the card's base address, and BASE+7 is the base address + seven bytes. Read address BASE+0 (A/D conversion data) and write address BASE+26 (D/A output channel 1 data) are 16-bit access. All other addresses are 8-bit access.

I/O port address map:::::

The following table shows the function of each register or driver and its address relative to the card's base address.

I/O port address assignments

Address	Read	Write
BASE+0	A/D data and channel (16-bit)	Software A/D trigger
BASE+1	N/A	A/D range control

A/D data register-BASE +0/1::::::

A 16-bit register at BASE+0 holds data from each A/D conversion and identifies the source A/D channel number. Bits 15 (MSB) to 4 (LSB) hold the 12 bits of data from the conversion. Bits 3 to 0 hold the source A/D channel number.

BASE+0 A/D conversion data and channel number (read)

Bit	D15	D14	D13	D12	D11	D10	D9	D8
Value	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4
Bit	D7	D6	D5	D4	D3	D2	D1	D0
Value	AD3	AD2	AD1	AD0	C3	C2	C1	C0

AD11 to AD0 ::::

Analog to digital conversion data. AD0 is the least significant bit (LSB) of the A/D data, and AD11 is the most significant bit (MSB).

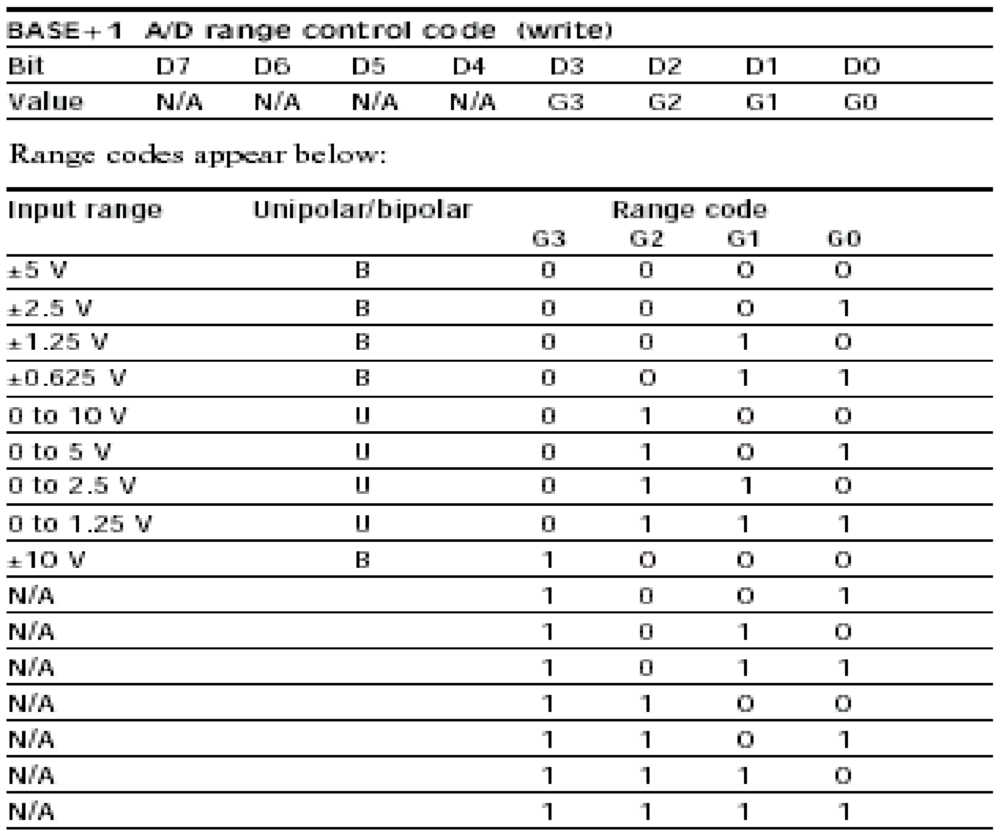
C3 to C0::::::
A/D channel number which supplied the data. C3 is the MSB and C0 is the LSB.

Software A/D triggering -BASE+0:::::::::

You can trigger an A/D conversion from software, the card's on-board pacer or an external pulse. If you select software triggering, write to the register BASE+0 with any value to trigger an A/D conversion. Bits 1 and 0 of register BASE+9 select the trigger source.

A/D range control-BASE+1::

Each A/D channel has its own individual input range, controlled by a range code stored in on-board RAM. If you want to change the range code for a given channel, select the channel as the start channel in register BASE+2, MUX scan, then write the range code to bits 0 and 3 of BASE+1.



BASE+1 A/D range control code (write)

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Value	N/A	N/A	N/A	N/A	G3	G2	G1	G0

Range codes appear below:

Input range	Unipolar/bipolar	Range code			
		G3	G2	G1	G0
±5 V	B	0	0	0	0
±2.5 V	B	0	0	0	1
±1.25 V	B	0	0	1	0
±0.625 V	B	0	0	1	1
0 to 10 V	U	0	1	0	0
0 to 5 V	U	0	1	0	1
0 to 2.5 V	U	0	1	1	0
0 to 1.25 V	U	0	1	1	1
±10 V	B	1	0	0	0
N/A		1	0	0	1
N/A		1	0	1	0
N/A		1	0	1	1
N/A		1	1	0	0
N/A		1	1	0	1
N/A		1	1	1	0
N/A		1	1	1	1

A/D conversion:::::

This chapter explains how to use the PCL-818HG's A/D conversion functions. The PCL-818HG gives you a wide array of options for triggering and data transfer. You can trigger the data conversions from software, the card's onboard

pacer or an external signal. You can transfer the acquired data using software, interrupt or DMA. In addition, you can use the card's FIFO buffer to store the data then transfer it through any of the above methods except DMA.
A/D data format and status register A 16-bit read-only register at BASE+0 holds data from each A/D
conversion and identifies the source A/D channel number. Bits D15 (the MSB) to D4 (the LSB) hold the 12 bits of data from the conversion. Bits D3 (the MSB) to D0 (the LSB) hold the source A/D channel number.

A/D Data Format and Status Register::::::::

Since the PCM-3718H/3718HG uses 12-bit A/D conversions, a single 8-bit register will not accommodate all the data. The PCM-3718H/3718HG therefore stores A/D data in two registers located at addresses BASE+0 and BASE+1. It stores the A/D low byte data in bits D4 to D7 (AD0 to AD3) of BASE+0 and high byte data in bits D0 to D7 (AD4 to AD11) of BASE+1. The least significant bit is AD0 and the most significant bit is AD11. You can read the source channel number corresponding to the A/D data from bits D0 to D3 (C0 to C3) of BASE+0.

A/D data register format is:

BASE+0 (read only) - A/D low byte & channel number								
Bit	D7	D6	D5	D4	D3	D2	D1	D0
Value	AD3	AD2	AD1	AD0	C3	C2	C1	C0

BASE+1 (read only) - A/D high byte								
Bit	D7	D6	D5	D4	D3	D2	D1	D0
Value	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4

The A/D status register at BASE+8 (read only) gives information on A/D configuration and operation.

A/D status register format is:

BASE+8 - A/D status								
Bit	D7	D6	D5	D4	D3	D2	D1	D0
Value	EOC	N/A	MUX	INT	CN3	CN2	CN1	CN0

Bits in this register indicate the end of conversion status, single ended/differential input, interrupt status and the number of the channel to be converted next.

Trigger Mode::::::

You can trigger an A/D conversion from software, from the module's on-board pacer or from an external signal. Bits 1 and 0 of register BASE+9 select the trigger source.

1. If you select software triggering, write to register BASE+0 with any value to trigger an A/D conversion. High-speed A/D applications do not normally use software triggering because the triggering rate is too slow.
2.The PCM-3718H/3718HG’s on-board Intel 8254 programmable interval timer/counter can generate periodic timing signals. Counters 1 and 2 of the Intel 8254 provide A/D converter trigger pulses with precise periods. You can select pacer output between 2.5 MHz and 71 minutes per pulse. Chapter 7 cover the details of the Intel 8254 timer/counter. Pacer triggering is ideal for interrupt and DMA data transfer, normally used in A/D applications which require higher conversion speeds.
3.You can also trigger the A/D conversion from an external signal. Wire the external signal to pin 1 on connector P2 and switch jumper JP3 to EXT. You would normally use external triggering if your application requires A/D conversions not periodically, but conditionally, e. g., to measure a voltage when a limit switch closes. The A/D conversion starts at the rising edge of the external trigger pulse.

Trigger mode:::::::

You can trigger an A/D conversion from software, the card's on-board pacer or an external pulse (CN3, pin 35). Select the trigger source by programming bits D0 and D1 of register BASE+9.

Software::::

If you select software triggering, write to register BASE+0 with any value to trigger the A/D conversion. You would not use software triggering in high speed A/D applications because the triggering rate is too slow.

External trigger:::

You can provide an external signal to trigger the A/D conversion. Connect the external signal to A/D EXT. TRIG (pin 35 on connector CN3). You would normally use external triggering if your application requires A/D conversions conditionally, not periodically, e.g., measuring a voltage when a limit switch closes. The A/D conversion starts at the rising edge of the external trigger pulse.

A/D data transfer:::

The PCL-818HG can transfer A/D data by program control, interrupt routine or DMA. You can transfer the data from each conversion to the PC immediately, or you can store data from a series of conversions in the FIFO (First In
First Out) buffer and copy it all at once. In addition, you can use the PCL-818HG's watchdog comparator circuit

to transfer the collected data only when it meets specific conditions. Program- and interrupt-controlled data transfer are straightforward. DMA data transfer, however, can be quite complicated, because the PCL-818HG uses dual-buffer 16-bit DMA. Unless you are extremely experienced with the PC and PCL-818HG (and have a high tolerance for pain) we recommend that you use the PCL-818HG software driver.

Program-controlled data transfer:

(without FIFO)

In program controlled data transfer your program polls the card to see when the card has finished an A/D conversion, then accesses the data by reading it from the card's I/O port (BASE+0). If you use software triggering, your program should trigger an A/D conversion (by writing any value to BASE+0), then poll the EOC (end of conversion) bit in the A/D status register (BASE+8). When EOC is off, the converter has finished processing the data. Your program can then read the data from the A/D data registers. If you use pacer or externally triggered A/D, your application program should check the EOC bit (end of conversion) of the A/D status register (BASE+8). If this bit is off (0), an A/D conversion is in progress. When the EOC bit is on, the conversion is finished and the data has been placed in the A/D data register (BASE+0). Your program can then read the data. You can also use the PCL-818HG's FIFO buffer to store the data, as described in the following section.

FIFO

General information::::::::::

The PCL-818HG's FIFO (First In First Out) buffer It lets you acquire data at up to 330 KHz and protects you from data loss with high data rates, especially under multitasking operating systems like Windows. Normally, the PCL-818HG transfers data from A/D conversions one element at a time. It performs a conversion, writes the data to the data output register, then transfers the data using DMA or an interrupt service routine. Without the FIFO, the each time the card performs a conversion, the data overwrites existing data in the output register. If the old data is not removed before the new data arrives, the old data is lost. With the FIFO enabled (BASE+21, bit 6 = 1) the new data simply fills the second element in the buffer, leaving the original data intact. Data from each further conversion fills the other elements in the FIFO. The FIFO holds data from up to 1024 conversions.

When you want to remove data from the FIFO, simple read a data element from the data register (BASE+0). This pulls the earliest data from the FIFO; the next data element in the FIFO takes its place. You can transfer data elements from the FIFO buffer at any time. The card continues saving data to the FIFO even as you are removing the old data, preventing data loss. You can also wait until the FIFO becomes full or half full, then remove all the elements at once. This method allows extremely fast data transfer, because it minimizes the number of instructions the CPU must perform. The RIS (repeat input string) command in the 80x86 instruction set allows you to repeatedly transfer data elements with one command.

Software-controlled data transfer::::::::

Programming with the FIFO is similar to standard software-controlled data transfer. Three flags in BASE+21 indicate the status of the FIFO: the empty flag (EF), half-full flag (HF) and full-flag (FF). If you want to remove data as the card acquires it, poll EF. If EF is 0, one or more conversions have occurred and data is available in the FIFO. Your program should then read each element in turn from BASE+0, checking EF to determine when the FIFO is empty. You can also wait for the FIFO to become full or half full before you empty it. Simply poll the HF or FF flag.

FIFO vs. Non FIFO operation::::

The pcl-8181HG 's buffer can protect us from data loss. In normal operation, the pcl-818HG generates an interrupt when it has finished converting the A/D data, and our ISR transfers the data. A problem occurs, however, when a higher priority system interrupt blocks the PCL –818hG's interrupt. Our ISR must wait until the system interrupt finished before it can transfer the data. If the wait is too long, the PCL-818HG may perform another A/D conversion and overwrite the data in the A/D registers before our routine can transfer the old data. The new A/D conversion will call our ISR again, but it will only transfer the new data.
The PCL-818Hg's FIFO buffer solves this problem. The FIFO stores the data element from each successive A/D conversion. If our ISR cannot transfer the first element in the buffer before the next conversion, the FIFO simply stores the data from the second conversion at the next position.
The programming procedure with the FIFO is basically the same as with the standard data registers. The only difference is that our IST should transfer, as many data elements as are in the FIFO buffer not just a single element. It should read an element from the data register (BASE +23/24) then check the FIFO empty flag, bit D0 of BASE +25. When D0 is 1, the buffer is empty.
If we want to flash the FIFO, write any value to BASE +25. This will empty the FIFO and set the empty flag to 1.

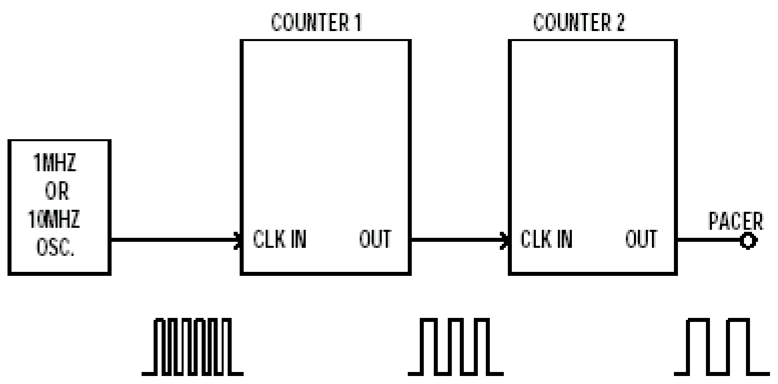
<u>How to Make an A/D Conversion::::::</u>

To perform A/D conversion, you can write all I/O port instructions directly in your program, or you can take advantage of the PCM-3718H/3718HG driver. We suggest that you apply the driver functions in your program. This will make your programming job easier and improve the program performance. See the User's Manual of the software driver for more information. Do the following to perform software trigger and program controlled
data transfer without the PCM-3718H/3718HG driver:
1. Set the input range for each A/D channel.
2. Set the input channel by specifying the MUX scan range.
3.Trigger the A/D conversion by writing to the A/D low byte register (BASE+0) with any value.
4. Check for the end of the conversion by reading the A/D status register (BASE+8) INT bit.
5. Read data from the A/D converter by reading the A/D data registers (BASE+0 and BASE+1).
6.Convert the binary A/D data to an integer.

Programmable Pacer:

The Intel 8254:::

The PCM-3718H/3718HG uses the Intel 8254 programmable interval timer/counter version 2. The popular 8254 offers three independent 16-bit down counters. Each counter has a clock input, control gate and an output. You can program each counter for maximum count values from 2 to 65535. Version 2 of the 8254 has a maximum input clock frequency of 10 MHz. The PCM-3718H/3718HG provides 1 MHz and 10 MHz input frequencies to the 8254 from an on-board crystal oscillator. Jumper JP1 controls the input frequency..
Counters 1 and 2 on the 8254 are cascaded and operated in a fixed divider configuration. Counter 1 input is connected to the 1 MHz or 10 MHz clock frequency, and the output of Counter 1 is connected to the input of Counter 2. The output of Counter 2 is internally configured to provide trigger pulses to the A/D converter, as shown below:



The Intel 8254 has six operational modes, from Mode 0 through Mode 5. To generate a pacer clock, program both Counter 1 and Counter 2 for Mode 3 (square wave generation).
Note!: Counter 0 is reserved for future development.

Counter Applications:

The 8254 programmable Interval timer/counter on your PCM-3718H/ 3718HG interface module is a very useful device. You can program counters 1 and 2 as pacers to generate A/D conversion trigger pulses. Counter 0 is reserved for later module development.

Setting the Pacer Rate:

The following equation gives the pacer rate:
Pacer rate = FCLK / (C1 *C2)
FCLK is either 1 MHz or 10 MHz, as set by jumper JP1.

A/D Calibration:::::::::::

Regular and accurate calibration procedures ensure the maximal accuracy. The CALB.EXE calibration program leads you through the whole A/D offset and gain adjustment procedure. The basic steps are outlined below:
1.Connect an external DC voltage source with value of 0.5 LSB to A/ D Channel 0 (pin 1 on connector P1).
2. Adjust VR2 until the output from the card's A/D converter flickers between 0 and 1.
3. Connect an external DC voltage source with a value of 4094.5 LSB
to A/D channel 0.
4. Adjust VR1 until the A/D reading flickers between 4094 and 4095.
5. Repeat steps 2 to step 4, adjusting VR1 and VR2.

6. Select unipolar input configuration. Connect an external DC voltage source with a value of 0.5 LSB to A/D channel 0. Adjust VR3 until the reading of the A/D flickers between 0 and 1.

PCL-818HG SOFTWARE DRIVER::::::::

Introduction: Each Advantech PC-Labcard data acquisition and control card comes with a software driver that allows us to control the card's function using high level languages. We standardized all our drivers so that we use the same functions and parameters. This lets us write a single program and use it with any PC-Labcard . We need only link the new card's driver to the program.
In addition our drivers make programming easier because each function pulls its parameters from a common parameter table. You don't need to specify function parameters each time you call it only when the parameter changes.
The standard driver specification covers many functions. Among them we worked on **A/D conversion**

Most cards don't support the full functionality of the driver, but we wrote the driver functions so that they would allow you take full advantage of each card's capabilities.

Supported languages:::::::

We supply driver interfaces for –
1) BASIC (BASIC A, GWBASIC, QUICK BASIC 4.0/4.5)
2) C (Microsoft C, Turbo C/C++)
3) Turbo Pascal

Running the Driver:::::::::::::

The drivers are Terminate and Stay Resident (TSR) programs which run in the background while our application runs in the foreground. Before we start our application program, we must load the driver into our system's memory. To do this type the driver's name at the DOS prompt. For example, type **`PCL-818`** to load the driver for the PCL-818.
Each driver supports up to two cards at one time, the PCL-818 driver simultaneously supports two PCL-818 cards. If you want to use more than one kind of card, you must load the driver for each card into memory. The drivers will not load themselves into memory more than once.

Parametre Table:::::

Many older software drivers require that you issue a long list of function calls to perfrom a simple A/D conversion. For example, we might need to do the following to perform an A/D data transfer.

1) Use function 1 to initialize the hardware
2) Use function 2 to set the input range
3) Use function 4 to set start and stop scanning channel
4) Use function 3 to set the A/D trigger mode
5) Use function 5 to set the pacer clock rate
6) Use function 6 to set the IRQ level
7) Use function 7 to set the DMAQ level
8) Use function 8 to perform A/D conversion “N’ times
9) Use function 9 to check the operation status

Performing these procedures each time you write our program becomes very time consuming and frustrating. Why should we have to remember so amny function calls and their parameters?

The Pc-Labcard software driver simplifies our programming because its functions cover whole operations , FUNCTION 6 “A/D conversion with DMA data transfer””. In addition, all functions take their parameters from a common parameter table.

The parameter table holds parameters, masks, minimum/maximum values and other specific information. Each tiem we call the driver, we simply pass it the function number and a memory address pointer to the parameter table. The driver will pick the parameters the function needs from the parameter table, then execute the function.

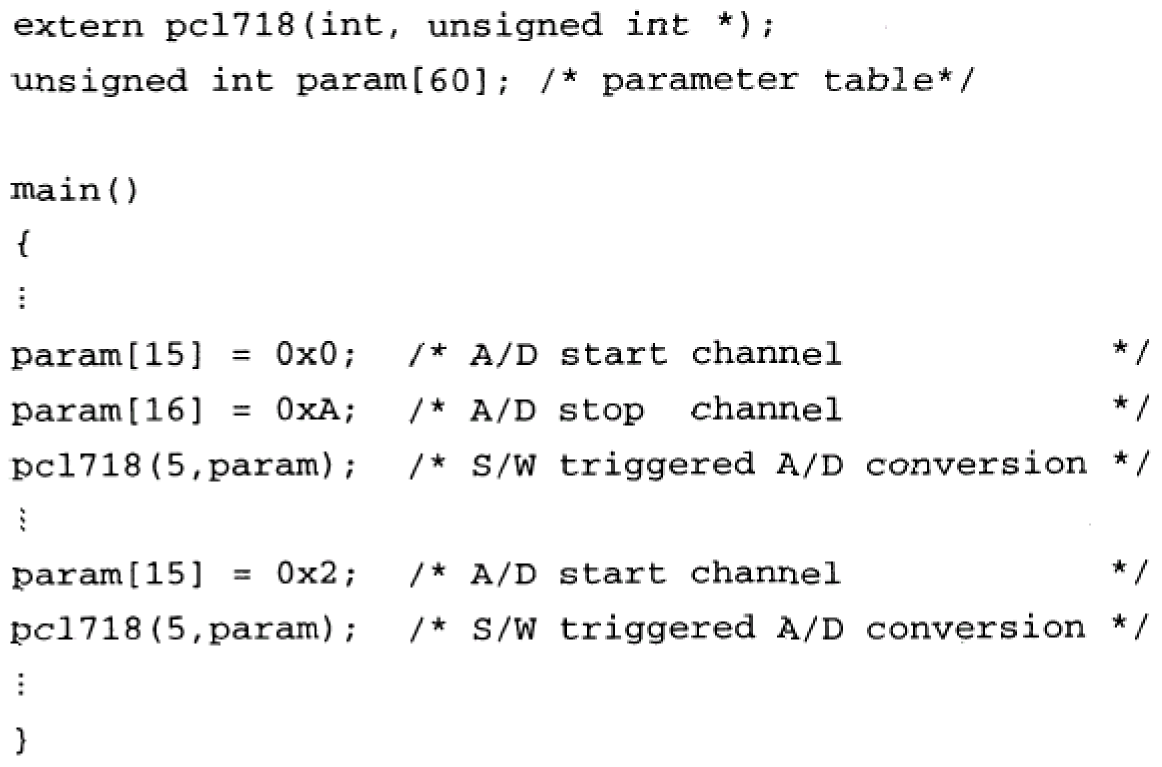
If we need to call the function with new values, we simply update the parameter table, then call the driver. We can also setup two or more parameter tables with different settings. When we call the driver, you then pass the driver the address of the parameter table we wish to use.

Programming Example:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>

extern pcl818HG(int, unsigned int *);
unsigned int param[60];         /* If two boards installed, need to
declare the second parameter array     */
unsigned    int    data[100];    /*    Conversion    data    buffer
*/
unsigned int far * dat;
main()
{
            unsigned int i;
            float       DataBuf;
            FILE *fp1;
            fp1=fopen("c:\\plot3.txt","wt");
            clrscr() ;
 /*PARAMETER TABLE*/

            dat= data;
            param[0]  = 0;            /* Board number
*/
            param[1]  = 0x300;        /* Base I/O address
*/
            param[5]  = 50;           /* Pacer rate = 10M / (50 * 100)
= 2000 Hz  */
            param[6]  = 100;
            param[7]  = 0;                /* Trigger mode, 0 : pacer
trigger  */
           param[10] = FP_OFF(dat);  /* Offset of A/D data buffer A
*/
            param[11] = FP_SEG(dat);  /* Segment of A/D data buffer A
*/
            param[12] = 0;            /* Data buffer B address, if not
used,    */
            param[13] = 0;            /* must set to 0.
*/
            param[14] = 110;           /* A/D conversion number
*/
            param[15] = 2;            /* A/D conversion start channel
*/
            param[16] = 2;            /* A/D conversion stop channel
*/
            param[17] = 0;            /* Overall gain code, 0 : +/- 5V
*/

            /* param[18] = FP_OFF(gain_array);
                 param[19] = FP_SEG(gain_array);  */

            /* param[45] : Error code
                 param[46] : Return value 0
                 param[47] : Return value 1 */
/*ACQUIRE DATA*/

            pcl818HG(3, param);        /* Func 3 : Hardware
initialization       */
            if (param[45] != 0) {
                      printf(" DRIVER INITIALIZATION FAILED !");
                      exit(1);
                      }

            pcl818HG(4, param);        /* Func 4 : A/D initialization
*/
            if (param[45] != 0) {
                      printf(" A/D INITIALIZATION FAILED !");
                      exit(1);
                      }
            /* Functin 5 will use FIFO, it must enable hardware FIFO.
*/
            pcl818HG(5, param);        /* Func 5 : "N" times of A/D
trigger      */
            if (param[45] != 0) {
            printf(" A/D SOFTWARE DATA TRANSFER FAILED !");
            exit(1);
            }
/*DISPLAY ACQUIRED DATA FROM BUFFER*/
            for (i = 0; i < param[14]; i++)    /* Display data
*/
      {
            DataBuf = data[i] & 0xFFF;
            DataBuf =( (5 - (-5)) * DataBuf / 4096) + (-5);
            /*
                 (5 - (-5)) : A/D input range (-5V  to  5V)
                 4096       : Full scale 12 bit A/D data
                 DataBuf    : A/D input data
                 (-5)       : A/D input range "-5" V
            */
            printf("\ndata[%3d] = % 1.2f V ", i, DataBuf);
 }
}
```

Modifying the parameter table::::

We can change a function’s parameters in two ways. We can change the parameters in the parameter table or we can specify a different parameter table with a set of values.
Modifying parameters directly
If we want to change the A/D start channel number, we don’t need to issue any function calls. Simply change the corresponding parameter in the parameter table, the n call the driver functions.
See the following example in C language::::



```
extern pcl718(int, unsigned int *);
unsigned int param[60]; /* parameter table*/

main()
{
⋮
param[15] = 0x0;   /* A/D start channel            */
param[16] = 0xA;   /* A/D stop  channel            */
pcl718(5,param);   /* S/W triggered A/D conversion */
⋮
param[15] = 0x2;   /* A/D start channel            */
pcl718(5,param);   /* S/W triggered A/D conversion */
⋮
}
```

Creating a new parameter table::::

If we need to switch back and forth between different settings or block of setting, create two copies of the parameter table. We can then select the proper parameter table then we call the driver functions.
Note that the driver can only access one parameter table at a time. The driver performs its operations based on specified parameter table.
The following C language example uses two parameter tables, PARAM 1 and PARAM2. The two parameter tables are same except for the start channel, stop channel and gain setting. When the program performs the operation for job1 it uses parameter table PARAM 1. For job2 it uses PARAM2.

```
extern pcl718(int, unsigned int *);
unsigned param1[60];/* Parameter Table 1          */
unsigned param2[60];/* Parameter Table 2          */

main()
{
:
/* Job 1 */
param1[15] = 0x0; /* A/D start channel           */
param1[16] = 0xA; /* A/D stop  channel           */
param1[17] = 0x0; /* gain code                   */
pcl718(5,param1); /* S/W triggered A/D conversion */
:
/* Job 2 */
param2[15] = 0x2; /* A/D start channel           */
param2[16] = 0x8; /* A/D stop  channel           */
param2[17] = 0x5; /* gain code                   */
pcl718(5,param2); /* S/W triggered A/D conversion */
:
}
```

Using two cards:::::

We can use two cards simultaneously with each card driver. We simply declare separate parameter tables for each card. In the first card's parameter table, set parameter 0, board number to 0. In the second card's parameter table, set parameter 0 to 1. We will also need to set parameter 1, base I/O address, in each parameter table to match the setting of the corresponding card. When we want to operate the first card, pass the driver the address of the first parameter table and vice versa.

Data Buffers:

Most driver function read data from or writes data to a data buffer instead of variable. The driver uses separate buffers for each operation (A/D) and it supports up to two buffers for each operation.

A data buffer is nothing more than a block of consecutive memory. We can setup this block using any method our language support. For example we can declare and reserve the memory space for an integer array in advance or allocate a block of memory dynamically as we need it. The data buffer can be any size from two bytes up to 64K bytes. We can locate it anywhere in conventional memory in PC real mode.
No matter what size our data buffer is, before we perform any function call. We must assign the buffer segment and offset addresses to the corresponding parameters in our parameter table.
Although the buffer can be up to 64 Kb in size, make sure that the data buffer doesn't overflow the segment. For example, if the offset address is 8000h and buffer size is A000h, the buffer would extend from 8000h to 12000h crossing the segment boundary (8000h+A000h=12000h >10000h). The overflow makes the address point to the wrong memory location. We can avoid this problem by reducing the buffer size to allocating another buffer with a smaller offset.
Make sure that the buffer is large enough to hold all of our data.Each element of data returned from an A/D conversion will occupy two bytes of space. Each element of data output by a D/A conversion will occupy two bytes. In the both cases, the first byte, in the buffer will be the low byte and the next high byte.
Each time we perform an operation, the driver will reads or writes the data to or from sequential location in the memory buffer until the driver ha s performed specified number of conversions.
If we are using a card with a FIFO buffer the minimum size of the memory buffer depends on the sampling frequency, as given by the following equation:

$$n \geq \frac{50}{(1000/f)-1.3}$$

Where n is the number of conversions in the memory buffer.
F= the sampling frequency in kHz

For example, if the sampling rate is 330Khz, the data buffer should be greater than 30 samples (60 bytes) in size.

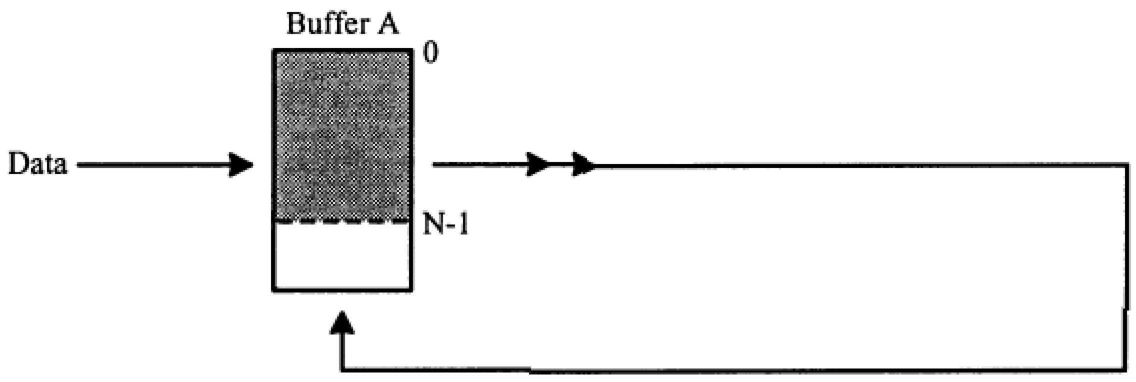
Double buffer data transfer:

We can define two buffers for each operation (A/D) allowing us to process the data while the driver is reading or writing it. The first buffer is designated BUFFER A, the second, BUFFER B. Both the buffers must be same size, each large enough to hold the number of the conversions set by PARAM 14.
For example, with double buffer A/D conversion the driver will first stores converted data to buffer A. When buffer A fills up the driver begins storing data to buffer B. We can then access the data in buffer A. When buffer B fills, the driver

switches back to buffer A and we can access in buffer B. We can tell when the driver switches buffer through a status function, Function 7 for standard A/D. The driver continues saving data until it has performed the specified number of conversions.
We can use this feature to operate “near” real time data acquisition applications. For example, we can acquire data to one buffer in the background and simultaneously display data from the other buffer in the foreground. The method does have its drawback, however. Since it takes time to switch between buffers, some of our data may become lost during the switching period. Needless to say, we should use this method of data storage with caution.

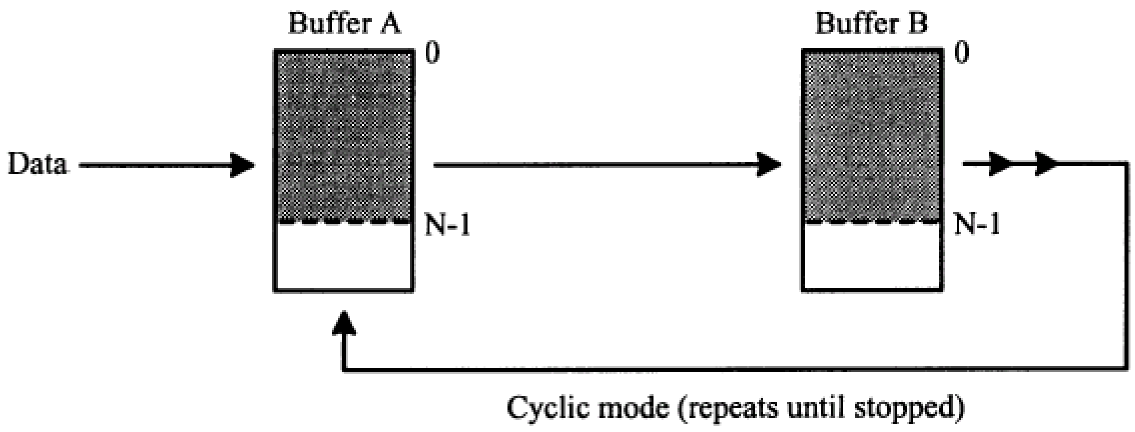
Cyclic mode:::::

The software driver can perform A/D operation in cyclic or non-cyclic mode.
In non-cyclic mode the driver performs a specified number of operations, then stops. With A/D, for example, it would perform a series of conversions and save the acquired data at successive memory locations in the buffer, the driver will continue to write or read data at memory locations outside the buffer. This could crash our program or cause strange behavior.
In cyclic mode the driver performs operations until we issue a STOP command. When it reaches the end of the buffer, it starts over at the beginning of the buffer. With A/D it saves data elements at successive locations in the buffer. When it reaches the end if the buffer, it begins saving data at the beginning of the buffer. The new data overcomes the old data at each location.



Cyclic mode (repeats until stopped)

Cyclic mode also works with double buffers, described in the previous section. The driver fills Buffer A, then Buffer B , then cycles back to Buffer A. The process continues until we issue a STOP command.



Parameter Description:

PARAM [0]-Card number

0=Card number 1
1= Card number 2

This parameter specifies the card to which the parameter table applies each card driver supports up to two cards. We can install and unlimited number of drivers in our system's simultaneously.

In the first card parameter table, set parameter 0 to 0. In the second card's parameter table, set parameter 0 to 1. When we want to operate the first card, pass the driver the address of the first parameter table and vice versa.

We will also need to set to parameter 1, base I/O address in each parameter table to match the setting of the corresponding card.

PARAM [1]- Base I/O address

We can select any address from 200 to 3F0 hex. This value must match the jumper setting of the card. We specify card may occupy either 16 to 32 I/O port address.

PARAM [5] and PARAM [6]-Pacer rate divider constants

Hold constants C1 and C2, which set the rate of the cards on board and timer. PARAM [5] holds C1 and PARAM [6] hold C2. The pacer rate is as follows:
Pacer rate = Card base frequency/(C1*C2)
Acceptable values for each constant is 0 to 65535

PARAM [7]-Trigger mode, Internal/External

0= Internal trigger (Pacer trigger)
1= External trigger (not Allowed for D/I functions)

PARAM [14]-number of A/D conversions

Range is 1 to 32767
This parameter holds number of A/D conversions to be performed. If we specify a number of conversions larger than our data buffer the driver will write data outside the data buffer. This can cause our program to crash.

PARAM [15]-A/D start channel

The channel number where the card starts reading. The card will scan from the start channel (PARAM [15]) to stop channel (PARAM [16]), then repeat. To read from a single channel set PARAM [15] and PARAM [16] to the same value.

PARAM [16]- A/D stop channel

Set the A/D stop channel number

PARAM [17]-Channel gain

This parameter sets the gain code for all the cards input channels at once. If we need different gain values for each channel, set PARAM [17] to FF hex and use a gain array table.

PARAM[45]-Error Number

PARAM[46]-Return value 0

PARAM[47]-Return value 1

ANALOG TO DIGITAL CONVERSION

Basic operation

Software data transfer (FUNCTION 5)

The function an A/D conversion, transfers the data to buffer A then waits for a trigger pulse from the cards on –board clock (pacer). When it receives the pulse it performs another conversion. The process continues until it has performed the number of conversion (N) specified in PARAM [14]. Only the does the driver returns control to our application program.
If we set PARAM [14] to 1, the function performs a single conversion then returns to our application immediately without waiting for a trigger signal. This function does not support double buffer or cyclic modes.

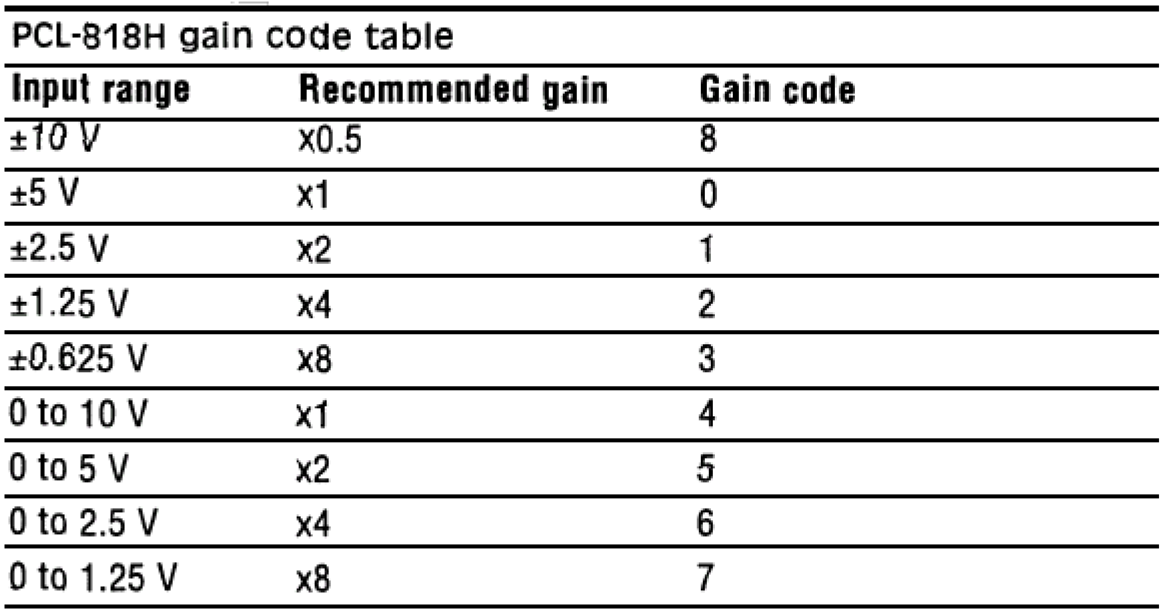
A/D start channel and stop channel::::::

The driver can sequentially scan through our cards from analog input channel. It will begin with the start channel and scan until it reads the stop channel. The driver will stop scanning when it has acquired specific number of readings.
For example, if the start channel number is 3, and the stop channel number is A hex, then the driver will scan channels 3,4,5,6,7,8,9,A, 3,4,…………

Gain array table:

Before we perform A/D conversions, we must set each channel’s gain to a specific value. For the highest possible resolution we should amplify the input signal so that the maximum voltage equals the maximum input range of the A/D converter.

If the gain is the same for all the channels, simply set PARAM [17] in the parameter table to the gain code. If the gain values are different for each channel, set PARAM [17] to FF (hex), and the driver will take the channel gain values from the gain table and array of words 16 bit values in memory.
Function 3: Driver initialization:
The Pclabcard is initialized according to the parameter's definitions. It will stop all functions, release all sources. It should be called before any other functions.



PCL-818H gain code table

Input range	Recommended gain	Gain code
±10 V	x0.5	8
±5 V	x1	0
±2.5 V	x2	1
±1.25 V	x4	2
±0.625 V	x8	3
0 to 10 V	x1	4
0 to 5 V	x2	5
0 to 2.5 V	x4	6
0 to 1.25 V	x8	7

Programming procedures:

This section outlines the parameters we will need to set and functions. We will need to call to perform A/D conversions operations. The function reference section gives detail information for each function.

A/D conversion with software data transfer:

1) Set parameter 0,1,5,6,7,10,11,12,13,14,15,16,17,18 and 19

2) Call function 3, Driver initialization

3) Call function 4, A/D initialization

4) Call function5 A/D conversion with S/w data transfer

5) If we want to perform more conversions, just call function 5 again. We don’t have to repeat the initialization

Set PARAM [14] to the number of conversion. Each time we call function 5, the card will perform this many conversions. It will perform each conversion then wait for the next trigger pulse. When the driver has performed all of the conversions, it will return control to our application program.

A/D functions

Parameters used

PARAM [0]	Card number
PARAM [1]	Base I/O address
PARAM [5]	Pacer rate divider constant C1
PARAM [6]	Pacer rate divider constant C2
PARAM [7]	Trigger mode, internal/external
PARAM [10]	A/D data buffer A address-offset
PARAM [11]	A/D data buffer A address-segment
PARAM [12]	A/D data buffer B address-offset
PARAM [13]	A/D data buffer B address-segment
PARAM [14]	Number of A/D conversions
PARAM [15]	A/D start channel
PARAM [16]	A/D stop channel
PARAM [17]	Overall gain code

Function reference

Function 3 Driver initialization

Function 4 A/D initialization

Function 5 A/D conversion with software data transfer

Function 3

Driver initialization:::::

This function initializes the driver according to the parameters in the parameter table. Our program must call it before we call any other driver function. It will also stop all functions and release all resources, so we may want to call this function if our program is exiting abnormally.

Function 4

A/D initialization

This function initializes the PC-Labcard's A/D function according to settings the parameter table/ Call it before we use any other A/D functions. We should already have called the function 3, driver initialization.

Function 5

A/D conversion with software data transfer

This function performs A/D conversion with software data transfer. It does not return control to our program until all the conversions have finished. This function does not double buffer mode - it saves data in buffer A only. It also does not support cyclic mode. Specify the number of conversions in PARAM [14].

Our performed works

Our total works was divided into some parts. All of the works was done on the basis of analog to digital conversion. We performed our works by maintaining the following steps:::::::
1)Sampling of sinusoidal signal and read those data
2)View of sinusoidal real time signal from stored data
3)Measurement of power
4)Measurement of energy
5)Development of computerized standard energy meter

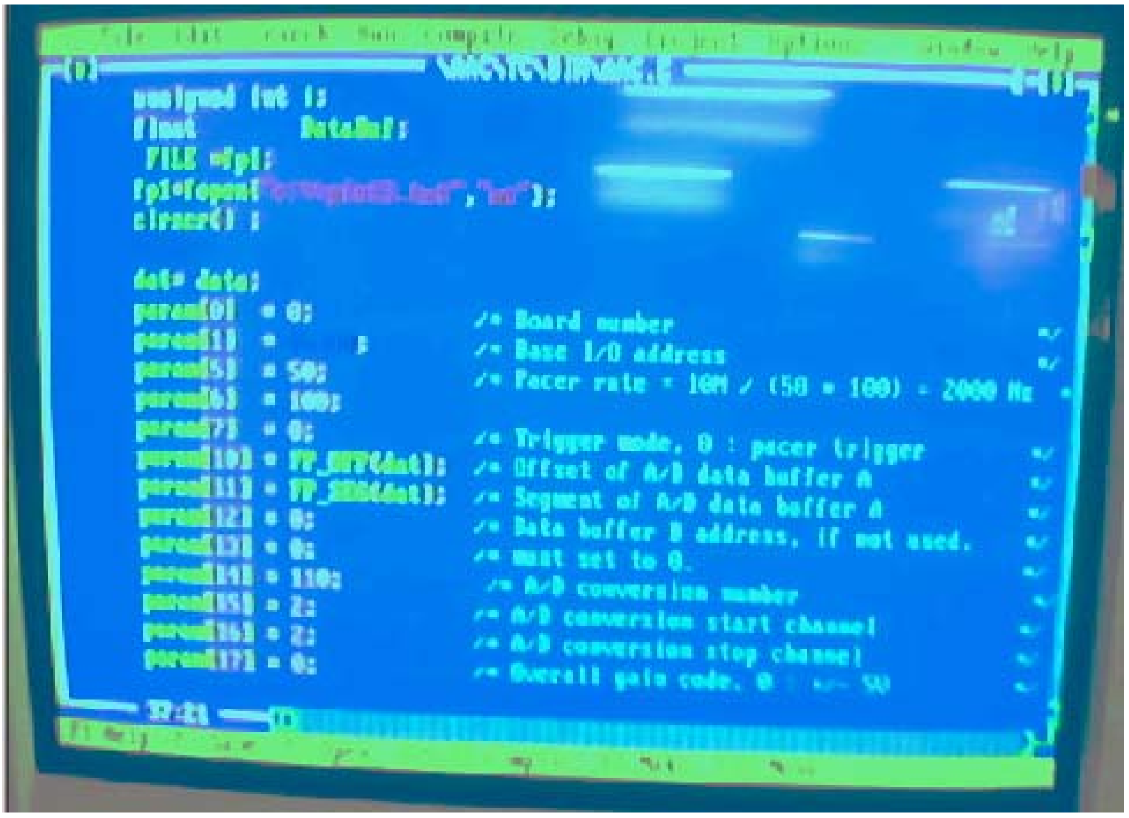
First operation::::

1) Sampling of sinusoidal signal and read those data

Software::::::

All calculation is done by program, which is written in C/C++ language. We used software driver program for initialization and setting of different parameters for data acquisition card.

Program:::::



```
unsigned int i;
float        DataBuf;
 FILE *fp1;
fp1=fopen("[illegible]","w");
clrscr() ;

dat* data;
param[0]  = 0;                 /* Board number                               */
param[1]  =      ;             /* Base I/O address                           */
param[5]  = 50;                /* Pacer rate = 10M / (50 * 100) = 2000 Hz    */
param[6]  = 100;
param[7]  = 0;                 /* Trigger mode, 0 : pacer trigger            */
param[10] = FP_OFF(dat);       /* Offset of A/D data buffer A                */
param[11] = FP_SEG(dat);       /* Segment of A/D data buffer A               */
param[12] = 0;                 /* Data buffer B address, if not used,        */
param[13] = 0;                 /* must set to 0.                             */
param[14] = 110;                /* A/D conversion number                     */
param[15] = 2;                 /* A/D conversion start channel               */
param[16] = 2;                 /* A/D conversion stop channel                */
param[17] = 0;                 /* Overall gain code, 0 : +/- 5V              */
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>

extern pcl818HG(int, unsigned int *);
unsigned int param[60];            /* If two boards installed, need to declare the second
parameter array     */
unsigned int data[100]; /* Conversion data buffer                    */
unsigned int far * dat;
main()
{
                unsigned int i;
                float       DataBuf;
                FILE *fp1;
                fp1=fopen("c:\\plot3.txt","wt");
                clrscr() ;

                dat= data;
                param[0]  = 0;          /* Board number                        */
                param[1]  = 0x300;       /* Base I/O address                   */
                param[5]  = 50;          /* Pacer rate = 10M / (50 * 100) = 2000 Hz  */
                param[6]  = 100;
                param[7]  = 0;              /* Trigger mode, 0 : pacer trigger  */
               param[10] = FP_OFF(dat);  /* Offset of A/D data buffer A           */
                param[11] = FP_SEG(dat);  /* Segment of A/D data buffer A          */
                param[12] = 0;          /* Data buffer B address, if not used,    */
                param[13] = 0;          /* must set to 0.                       */
                param[14] = 110;          /* A/D conversion number                 */
                param[15] = 2;          /* A/D conversion start channel          */
                param[16] = 2;          /* A/D conversion stop channel           */
                param[17] = 0;          /* Overall gain code, 0 : +/- 5V         */
```

```
        /* param[18] = FP_OFF(gain_array);
            param[19] = FP_SEG(gain_array);  */

        /* param[45] : Error code
            param[46] : Return value 0
            param[47] : Return value 1 */

        pcl818HG(3, param);      /* Func 3 : Hardware initialization      */
        if (param[45] != 0) {
                printf(" DRIVER INITIALIZATION FAILED !");
                exit(1);
                }

        pcl818HG(4, param);      /* Func 4 : A/D initialization          */
        if (param[45] != 0) {
                printf(" A/D INITIALIZATION FAILED !");
                exit(1);
                }

        /* Functin 5 will use FIFO, it must enable hardware FIFO.          */
        pcl818HG(5, param);      /* Func 5 : "N" times of A/D trigger     */
        if (param[45] != 0) {
        printf(" A/D SOFTWARE DATA TRANSFER FAILED !");
        exit(1);
        }
        for (i = 0; i < param[14]; i++)   /* Display data                */
    {
        DataBuf = data[i] & 0xFFF;
        DataBuf =( (5 - (-5)) * DataBuf / 4096) + (-5);
        /*
            (5 - (-5)) : A/D input range (-5V  to  5V)
            4096       : Full scale 12 bit A/D data
            DataBuf    : A/D input data
            (-5)       : A/D input range "-5" V
        */
        printf("\ndata[%3d] = % 1.2f V ", i, DataBuf);
        fprintf(fp1,"\n%f",DataBuf);
    }
    fclose(fp1);

}
```

Program ourput:

Summary of the operation:::::

2)View of two real time sinusoidal signals ::::::

Program::::::

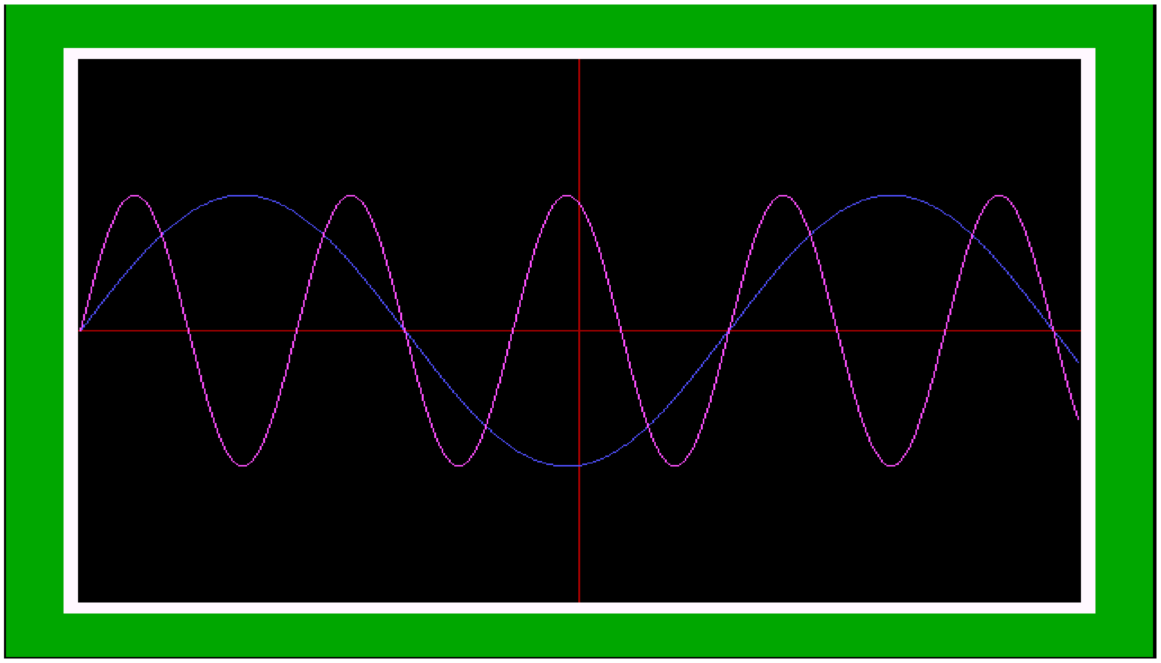
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include<graphics.h>

extern pcl818HG(int, unsigned int *);
unsigned int param[60];      /* If two boards installed, need to declare
                                                                the second
parameter array              */
unsigned int data[2000];

                    /* Conversion data buffer              */
unsigned int far * dat;



///=================================================================
        /// graph start 1
float rectangl_draw(void)
{
        /* request auto detection */
        int gdriver = DETECT, gmode, errorcode;
        int left, top, right, bottom;
        int ip;
                int x1,x2,y1,y2,m,n,c,d,k,w;
        float u,v;
        /* initialize graphics and local variables */
        initgraph(&gdriver, &gmode, "C:\\AAC\\TC\\BGI");

        /* read result of initialization */
        errorcode = graphresult();
        if (errorcode != grOk)  /* an error occurred */
        {
                printf("Graphics error: %s\n", grapherrormsg(errorcode));
                printf("Press any key to halt:");
                getch();
                exit(1); /* terminate with an error code */
        }
        //===============================================================
============

 //     int ip;
        left = getmaxx();
        top = getmaxy();
        right =getmaxx();
        bottom=getmaxy();
        left=left/512;
        right=right*(.998046875);
        top=top/512;
        bottom=bottom*(.998046875);
 //     printf("t=%d/b=%d/r=%d/L=%d",top,bottom,right,left);
        /* draw a rectangle */

        for(ip=0;ip<40;ip++)
        {
        if(ip<32)
        setcolor(2);
        else
        setcolor(15);
        rectangle(left+ip,top+ip,right-ip,bottom-ip);
        }



        //===============================================================
==

        m=left+ip;
        n=right-ip;
        c=top+ip;
        d=bottom-ip;
        k=n-m;
        w=d-c;
        u=k/2;
        v=w/2;
        setcolor(4);
        line(m,v+c,n,v+c);
        line(u+m,c,u+m,d);
        return(m);
        /* clean up */
        }


         /// graph end1


///=============================================================
void main()
{
                unsigned int i;
                float       DataBuf1[1000];
                float ud;
                        int x1,y1;
                        float m1;


               // clrscr() ;
                FILE *fp1;
                FILE *fp2;
                fp1=fopen("c:\\plot1.txt","wt");
                fp2=fopen("C:\\plot2.txt","wt");
                clrscr();
                dat= data;
                param[0]  = 0;          /* Board number                    */
                param[1]  = 0x300;        /* Base I/O address                  */
                param[5]  = 10;          /* Pacer rate = 10M / (50 * 100) = 2000 Hz  */
                param[6]  = 10;
                param[7]  = 0;          /* Trigger mode, 0 : pacer trigger        */
                param[10] = FP_OFF(dat);  /* Offset of A/D data buffer A            */
                param[11] = FP_SEG(dat);  /* Segment of A/D data buffer A           */
                param[12] = 0;           /* Data buffer B address, if not used,    */
                param[13] = 0;           /* must set to 0.                  */
                param[14] = 900;          /* A/D conversion number              */
                param[15] = 2;           /* A/D conversion start channel          */
                param[16] = 3;           /* A/D conversion stop channel           */
                param[17] = 0;           /* Overall gain code, 0 : +/- 5V         */

                /* param[18] = FP_OFF(gain_array);
                        param[19] = FP_SEG(gain_array);  */

                /* param[45] : Error code
                        param[46] : Return value 0
                        param[47] : Return value 1 */
```

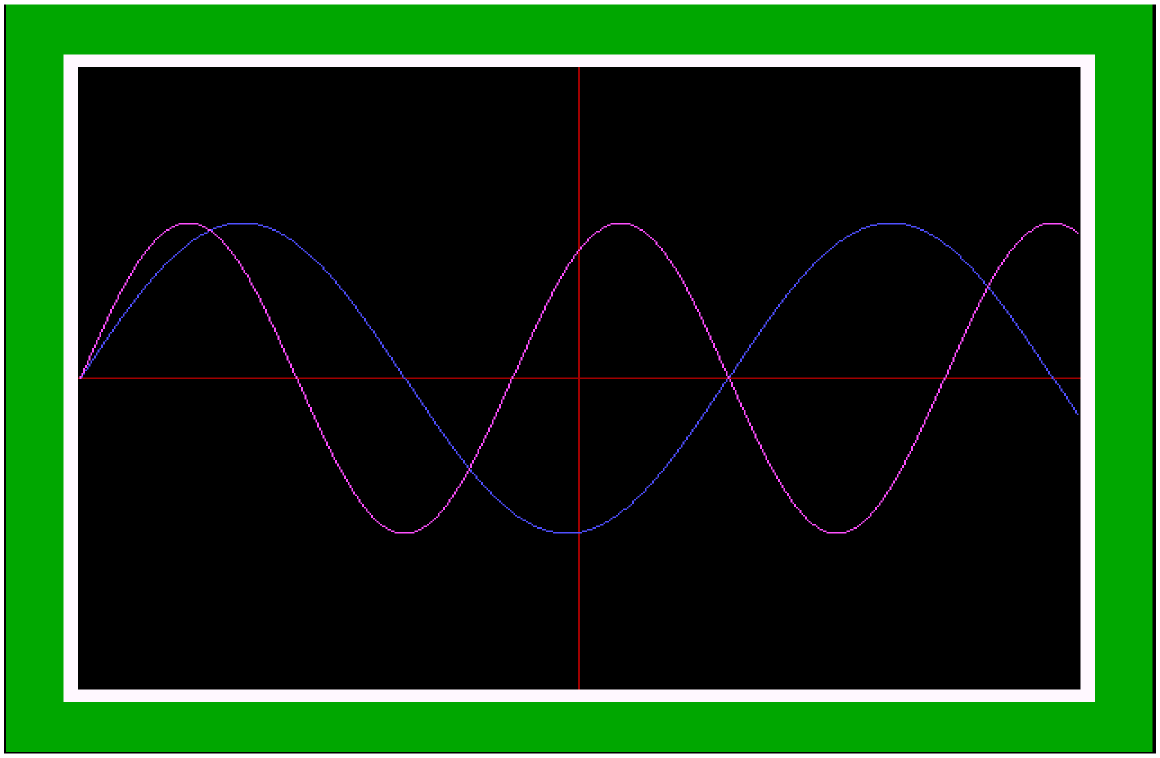
```
                    pcl818HG(3, param);       /* Func 3 : Hardware initialization      */
                    if (param[45] != 0) {
                                printf(" DRIVER INITIALIZATION FAILED !");
                                exit(1);
                                }


                    pcl818HG(4, param);       /* Func 4 : A/D initialization          */
                    if (param[45] != 0) {
                                printf(" A/D INITIALIZATION FAILED !");
                                exit(1);
                                }


/* Functin 5 will use FIFO, it must enable hardware FIFO.            */
                    do
                    {
                    pcl818HG(5, param);       /* Func 5 : "N" times of A/D trigger     */
                    if (param[45] != 0) {
                    printf(" A/D SOFTWARE DATA TRANSFER FAILED !");
                    exit(1);
                    }



///==================================================================
                    //graph start2

          m1=rectangl_draw();
          //=============================================================



          x1=getmaxx();
          x1=x1/2;
          y1=getmaxy();
          y1=y1/2;
          setcolor(9);

                    //        setcolor(13);
          //for(p=1;p<555;p++)
          //{
          //q[0]=0;
 //       q[p]=100*cos(p*3.14/180);
 //       line(p+m1,y1-q[p-1],m1+p+1,y1-q[p]);
 //       }

 //       getch();
 //       closegraph();
```

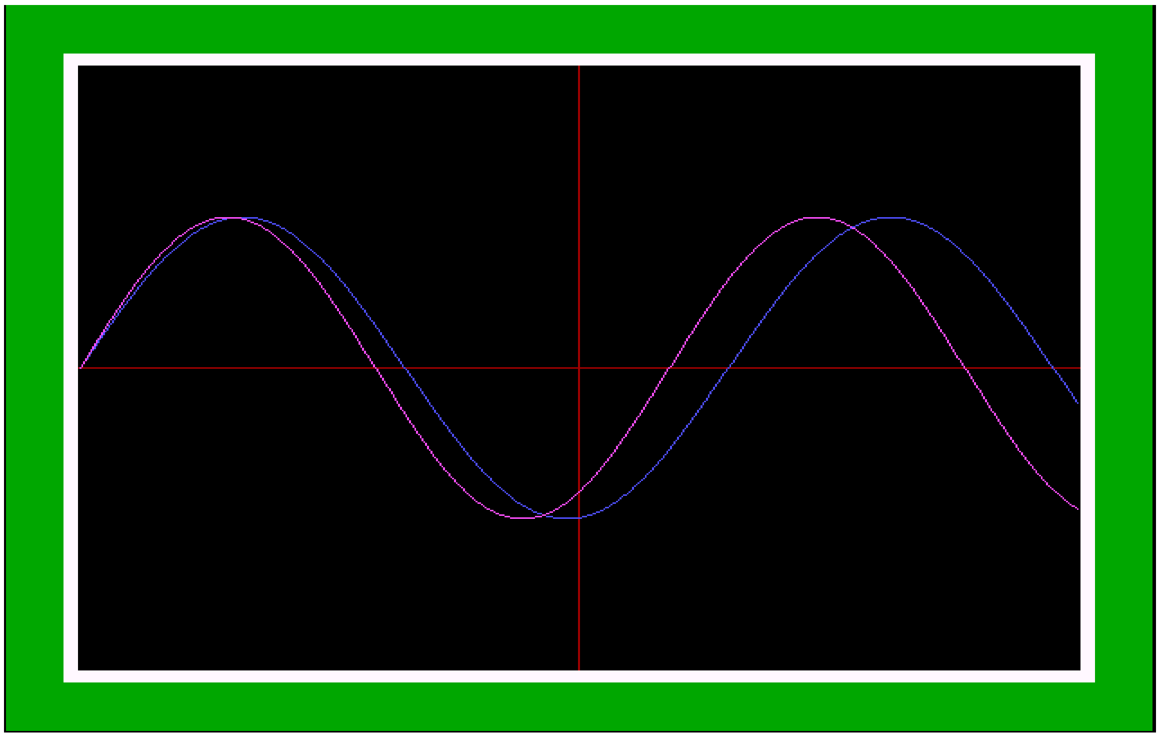
```
            //================

            // ud=rectangl_draw();

             for (i = 0; i < param[14]; i++)    /* Display data
                                                                    */
 {
 if(i==0||i%2==0)

 {

             DataBuf1[i] = data[i] & 0xFFF;
             DataBuf1[i] =(((5 - (-5)) * DataBuf1[i] / 4096) + (-5));
             fprintf(fp1,"\n%f",DataBuf1[i]);
  }
            else
            {

             DataBuf1[i] = data[i] & 0xFFF;
             DataBuf1[i] =(((5 - (-5)) * DataBuf1[i] / 4096) + (-5));

             /*
                         (5 - (-5)) : A/D input range (-5V  to  5V)
                         4096       : Full scale 12 bit A/D data
                         DataBuf    : A/D input data
                         (-5)       : A/D input range "-5" V
             */
            // printf("\ndata[%3d] = %f V ", i, DataBuf1[i]);
             fprintf(fp2,"\n%f",DataBuf1[i]);
             }
  }




  //==========================
 for(i=0;i<555;i++)
             {
             if(i==0||i%2==0)
             {

DataBuf1[0]=0;
             setcolor(13);
line(i+m1,y1-25*DataBuf1[i-1],m1+i+2,y1-25*DataBuf1[i+1]);
              }
              else
              {
              setcolor(14);
line(i+m1+1,y1-25*DataBuf1[i-1],m1+i+3,y1-25*DataBuf1[i+1]);
```

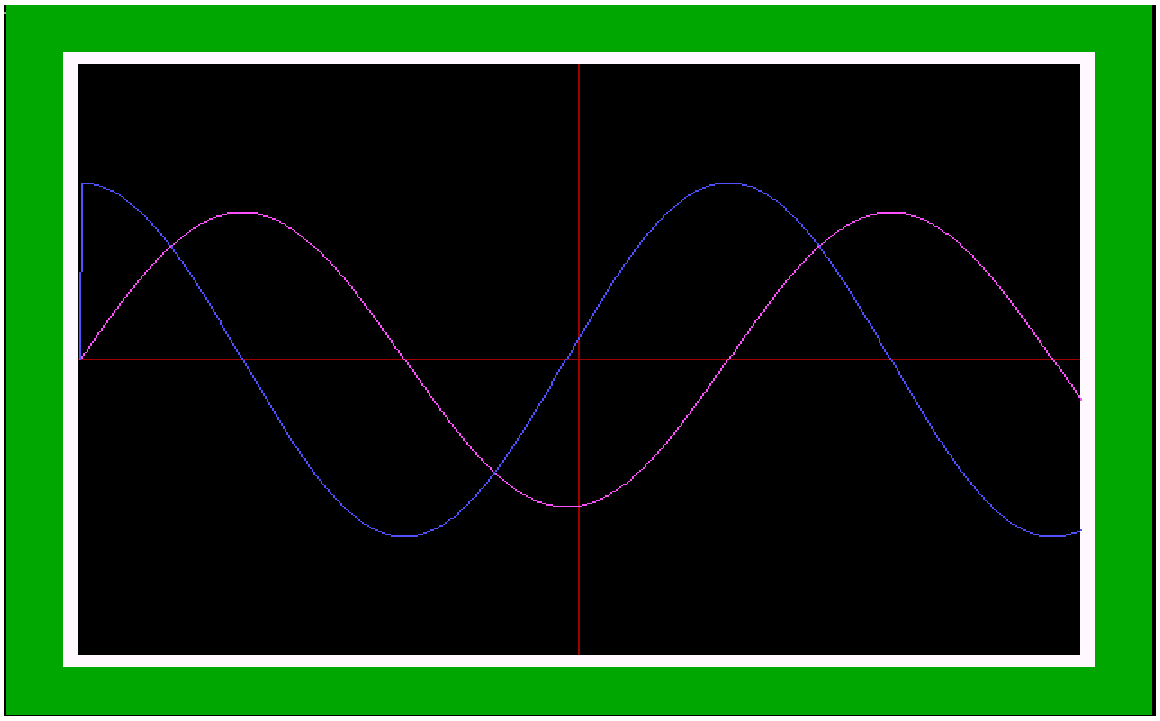
```
                    }
                    }



          //==============================
          getch();
          }
          while(kbhit()==0);

          fclose(fp1);
}
```

Program output::::::









Summary of the operation:

Formation of power curve::

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include<graphics.h>

extern pcl818HG(int, unsigned int *);
unsigned int param[60];       /* If two boards installed, need to declare
                                                                                                                                        the second
parameter array              */
unsigned int data[2000];

                        /* Conversion data buffer                  */
unsigned int far * dat;



///=====================================================================
            /// graph start 1
float rectangl_draw(void)
{
            /* request auto detection */
            int gdriver = DETECT, gmode, errorcode;
            int left, top, right, bottom;
            int ip;
                        int x1,x2,y1,y2,m,n,c,d,k,w;
            float u,v;
            /* initialize graphics and local variables */
            initgraph(&gdriver, &gmode, "C:\\AAC\\TC\\BGI");

            /* read result of initialization */
            errorcode = graphresult();
            if (errorcode != grOk)  /* an error occurred */
            {
                        printf("Graphics error: %s\n", grapherrormsg(errorcode));
                        printf("Press any key to halt:");
                        getch();
                        exit(1); /* terminate with an error code */
            }
            //=====================================================================
======

  //        int ip;
            left = getmaxx();
            top = getmaxy();
            right =getmaxx();
            bottom=getmaxy();
            left=left/512;
            right=right*(.998046875);
            top=top/512;
            bottom=bottom*(.998046875);
  //        printf("t=%d/b=%d/r=%d/L=%d",top,bottom,right,left);
            /* draw a rectangle */
```

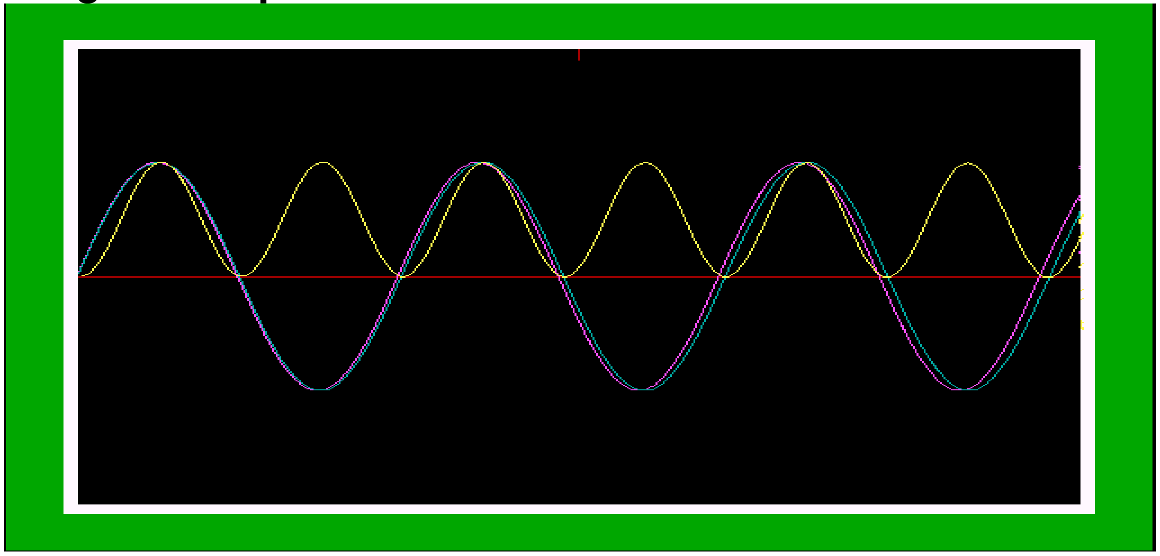
```
        for(ip=0;ip<40;ip++)
        {
        if(ip<32)
        setcolor(2);
        else
        setcolor(15);
        rectangle(left+ip,top+ip,right-ip,bottom-ip);
        }



        //=============================================================

        m=left+ip;
        n=right-ip;
        c=top+ip;
        d=bottom-ip;
        k=n-m;
        w=d-c;
        u=k/2;
        v=w/2;
        setcolor(4);
        line(m,v+c,n,v+c);
        line(u+m,c,u+m,d);
        return(m);
        /* clean up */
        }


         /// graph end1


///==========================================================
void main()
{
              unsigned int i;
              float       DataBuf1[1000];
              float ud;
                    int x1,y1;
                    float m1;


             // clrscr() ;
              FILE *fp1;
              FILE *fp2;
              fp1=fopen("c:\\plot1.txt","wt");
              fp2=fopen("C:\\plot2.txt","wt");
              clrscr();
              dat= data;
              param[0]  = 0;          /* Board number                    */
              param[1]  = 0x300;        /* Base I/O address                 */
              param[5]  = 10;          /* Pacer rate = 10M / (50 * 100) = 2000 Hz  */
```

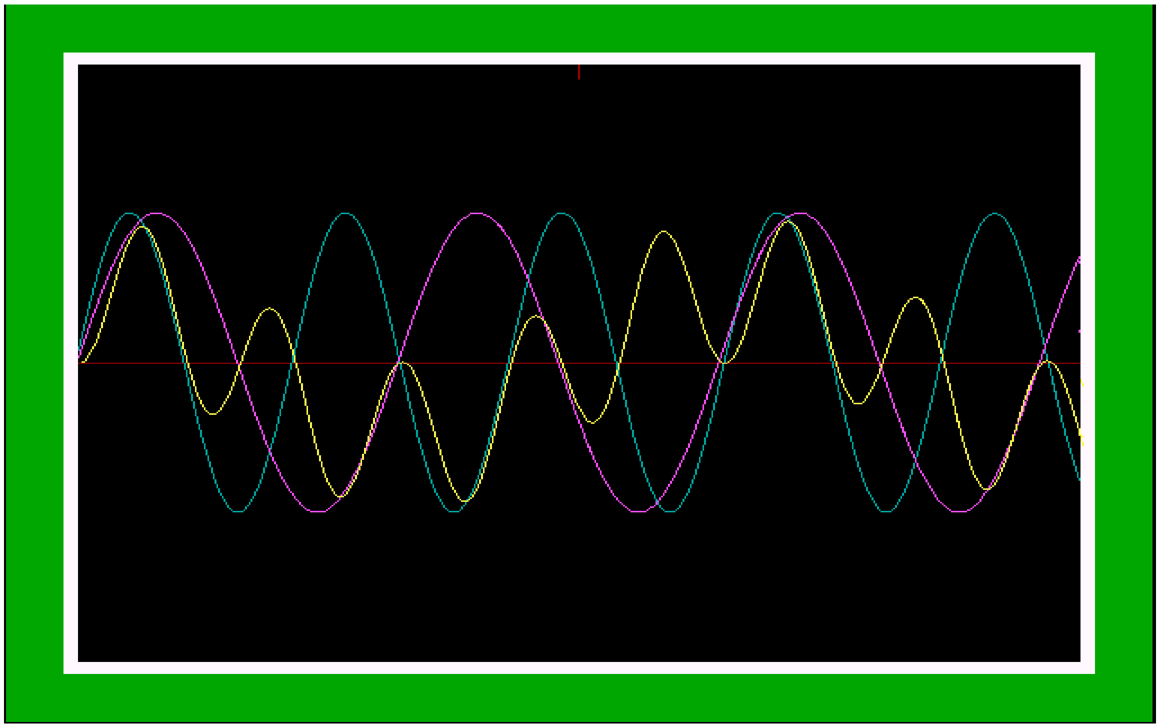
```
                    param[6]  = 10;
                    param[7]  = 0;           /* Trigger mode, 0 : pacer trigger        */
                    param[10] = FP_OFF(dat);  /* Offset of A/D data buffer A            */
                    param[11] = FP_SEG(dat);  /* Segment of A/D data buffer A           */
                    param[12] = 0;           /* Data buffer B address, if not used,    */
                    param[13] = 0;           /* must set to 0.                         */
                    param[14] = 900;           /* A/D conversion number                  */
                    param[15] = 2;           /* A/D conversion start channel           */
                    param[16] = 3;           /* A/D conversion stop channel            */
                    param[17] = 0;           /* Overall gain code, 0 : +/- 5V          */

                    /* param[18] = FP_OFF(gain_array);
                         param[19] = FP_SEG(gain_array);  */

                    /* param[45] : Error code
                         param[46] : Return value 0
                         param[47] : Return value 1 */

                    pcl818HG(3, param);       /* Func 3 : Hardware initialization       */
                    if (param[45] != 0) {
                              printf(" DRIVER INITIALIZATION FAILED !");
                              exit(1);
                              }


                    pcl818HG(4, param);       /* Func 4 : A/D initialization            */
                    if (param[45] != 0) {
                              printf(" A/D INITIALIZATION FAILED !");
                              exit(1);
                              }


/* Functin 5 will use FIFO, it must enable hardware FIFO.            */
                    do
                    {
                    pcl818HG(5, param);       /* Func 5 : "N" times of A/D trigger      */
                    if (param[45] != 0) {
                    printf(" A/D SOFTWARE DATA TRANSFER FAILED !");
                    exit(1);
                    }


               ///==============================================================
                    //graph start2

               m1=rectangl_draw();
               //==============================================================



               x1=getmaxx();
               x1=x1/2;
               y1=getmaxy();
               y1=y1/2;
```

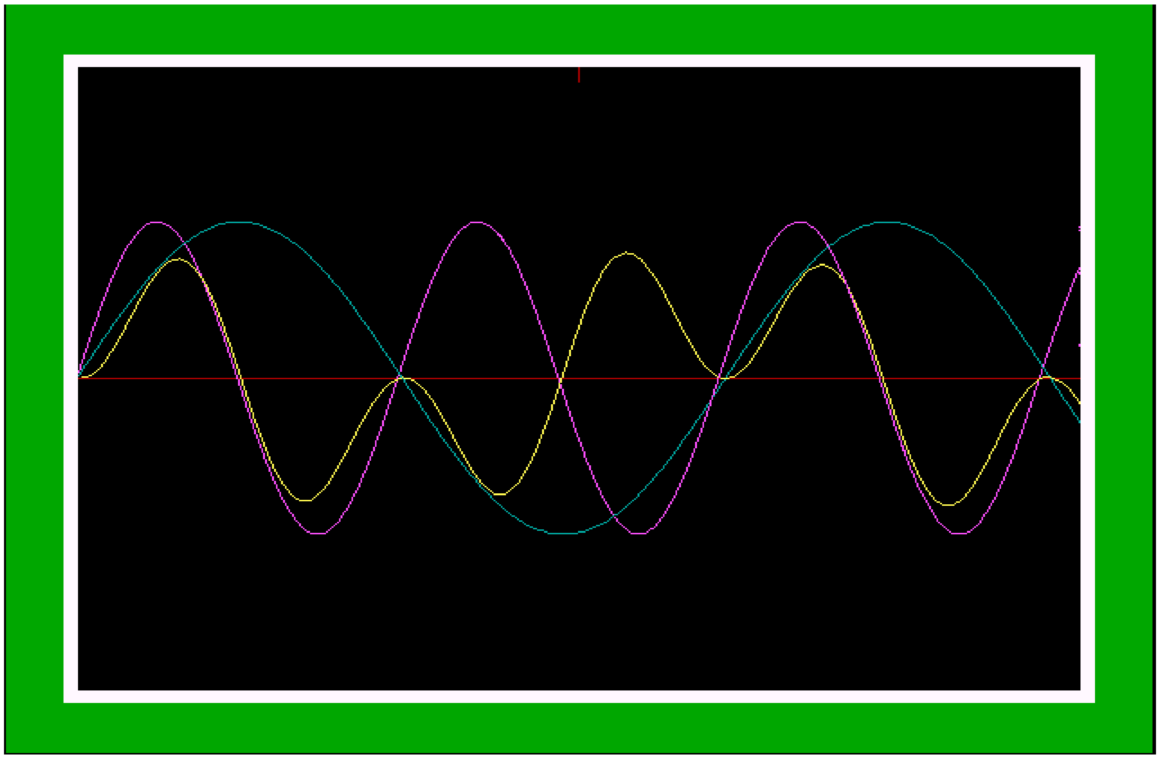
```
            setcolor(9);

                        //          setcolor(13);
            //for(p=1;p<555;p++)
            //{
            //q[0]=0;
    //      q[p]=100*cos(p*3.14/180);
    //      line(p+m1,y1-q[p-1],m1+p+1,y1-q[p]);
    //      }

    //      getch();
    //      closegraph();

                       //================

                       // ud=rectangl_draw();

                        for (i = 0; i < param[14]; i++)    /* Display data
                                                                          */
             {
             if(i==0||i%2==0)

             {

                        DataBuf1[i] = data[i] & 0xFFF;
                        DataBuf1[i] =(((5 - (-5)) * DataBuf1[i] / 4096) + (-5));
                        fprintf(fp1,"\n%f",DataBuf1[i]);
              }
                       else
                       {

                        DataBuf1[i] = data[i] & 0xFFF;
                        DataBuf1[i] =(((5 - (-5)) * DataBuf1[i] / 4096) + (-5));

                        /*
                                    (5 - (-5)) : A/D input range (-5V  to  5V)
                                    4096       : Full scale 12 bit A/D data
                                    DataBuf    : A/D input data
                                    (-5)       : A/D input range "-5" V
                        */
                       // printf("\ndata[%3d] = %f V ", i, DataBuf1[i]);
                        fprintf(fp2,"\n%f",DataBuf1[i]);
                        }
              }




              //=========================
             for(i=0;i<555;i++)
                        {
                        if(i==0||i%2==0)
                        {
```

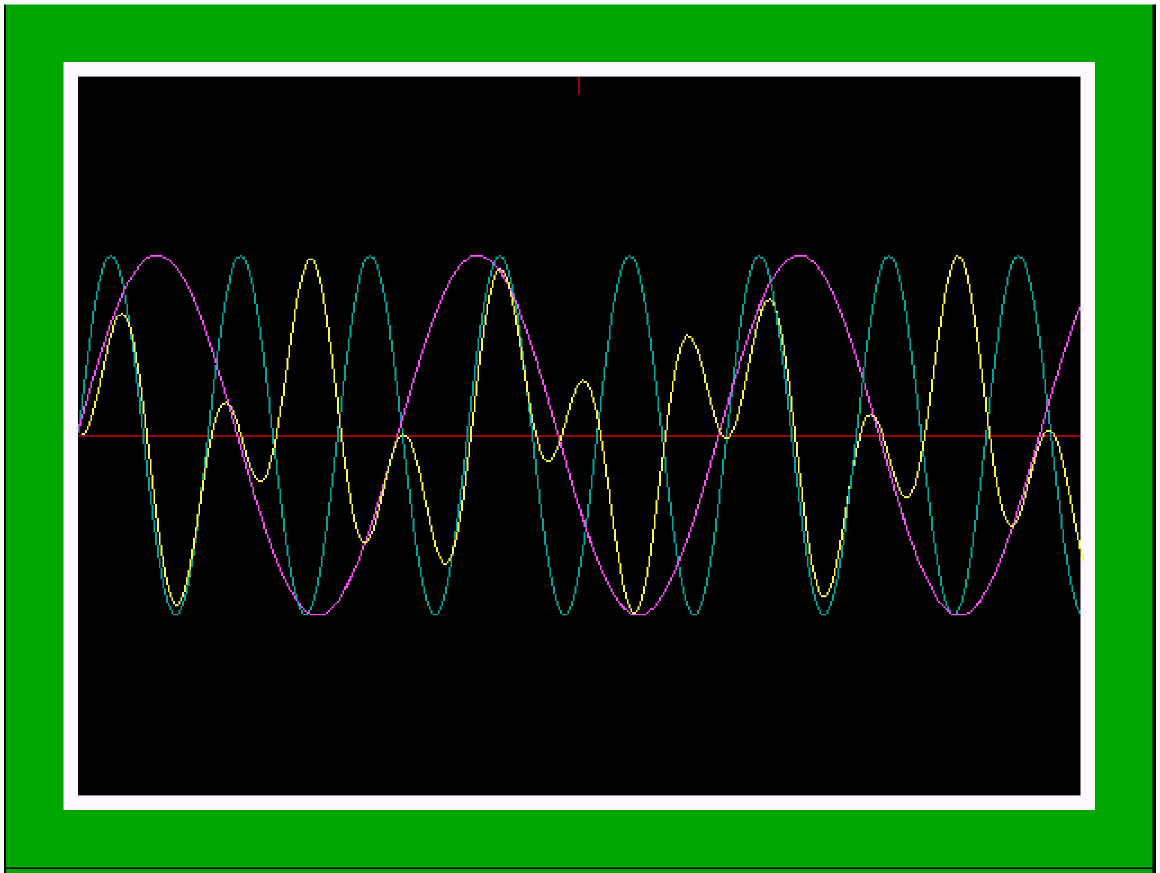
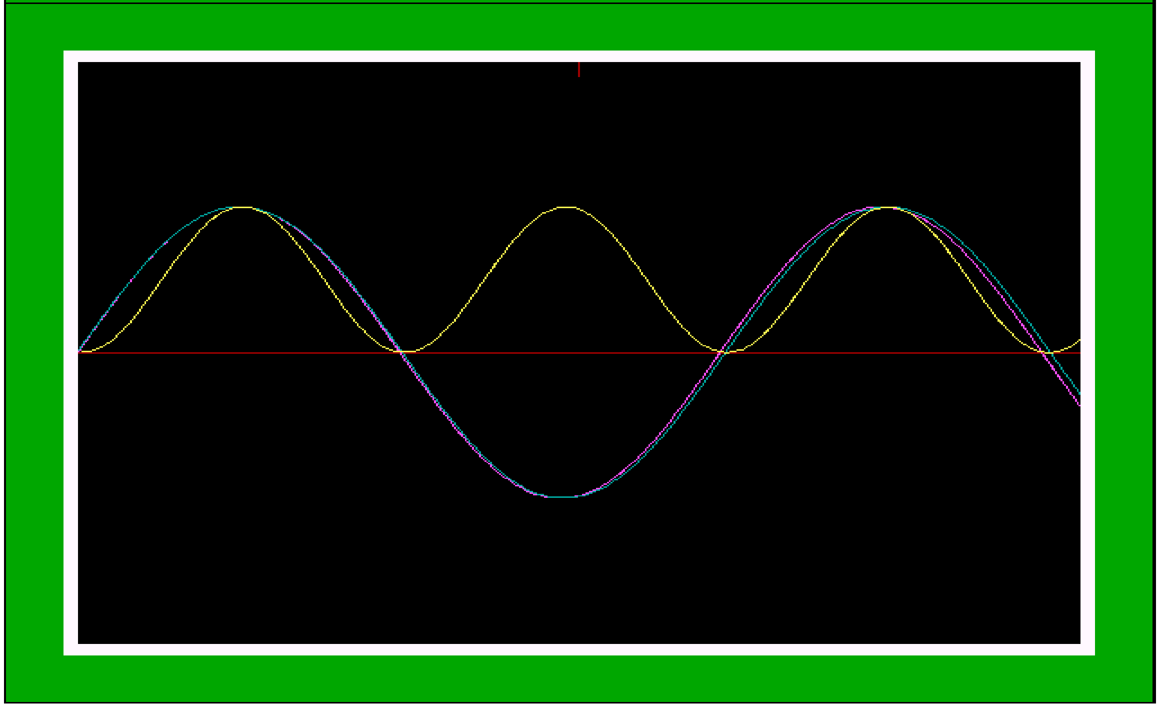
```
            DataBuf1[0]=0;
                   setcolor(13);
            line(i+m1,y1-25*DataBuf1[i-1],m1+i+2,y1-25*DataBuf1[i+1]);
                   }
                   else
                   {
                   setcolor(14);
            line(i+m1+1,y1-25*DataBuf1[i-1],m1+i+3,y1-25*DataBuf1[i+1]);

                   }
                   }



             //=============================
             getch();
             }
             while(kbhit()==0);

             fclose(fp1);
      }
```

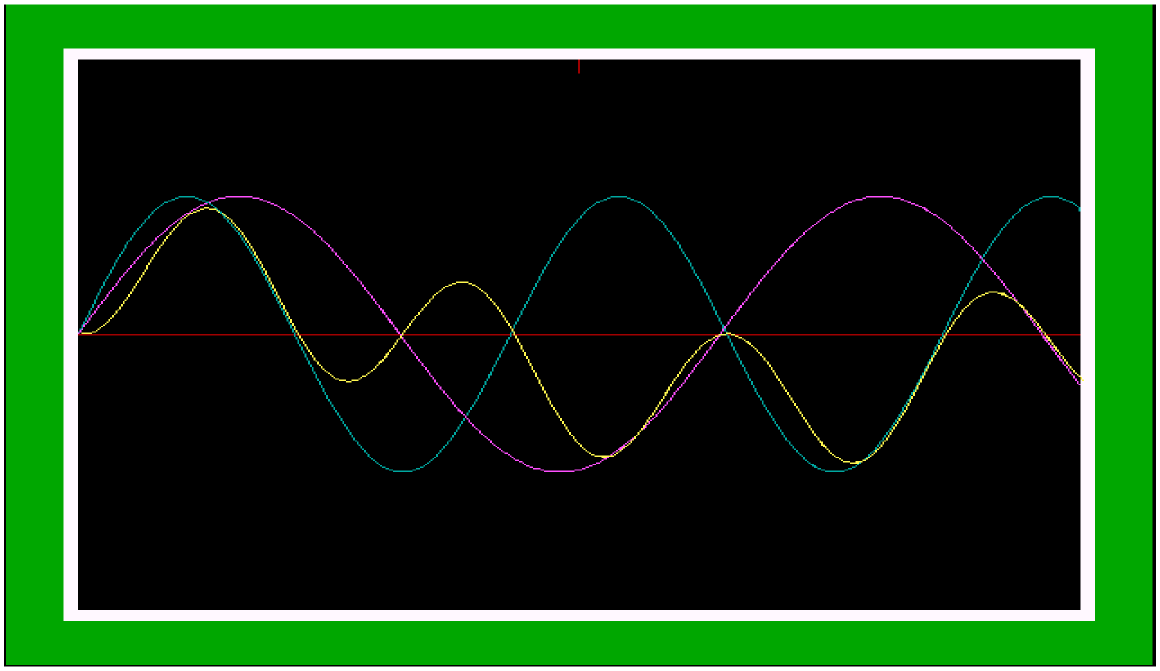
Program output:









Summary of the operation:::::

4)Formation of energy curve::::
Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include<graphics.h>

extern pcl818HG(int, unsigned int *);
unsigned int param[60];       /* If two boards installed, need to declare
                                                                                                    the second
parameter array               */
unsigned int data[2000];
```

```
                    /* Conversion data buffer              */
unsigned int far * dat;

///=================================================================
          /// graph start 1
float rectangl_draw(void)
{
          /* request auto detection */
          int gdriver = DETECT, gmode, errorcode;
          int left, top, right, bottom;
          int ip;
                    int x1,x2,y1,y2,m,n,c,d,k,w;
          float u,v;
          /* initialize graphics and local variables */
          initgraph(&gdriver, &gmode, "C:\\AAC\\TC\\BGI");

          /* read result of initialization */
          errorcode = graphresult();
          if (errorcode != grOk)  /* an error occurred */
          {
                    printf("Graphics error: %s\n", grapherrormsg(errorcode));
                    printf("Press any key to halt:");
                    getch();
                    exit(1); /* terminate with an error code */
          }
          //=================================================================
======

 //       int ip;
          left = getmaxx();
          top = getmaxy();
          right =getmaxx();
          bottom=getmaxy();
          left=left/512;
          right=right*(.998046875);
          top=top/512;
          bottom=bottom*(.998046875);
 //       printf("t=%d/b=%d/r=%d/L=%d",top,bottom,right,left);
          /* draw a rectangle */

          for(ip=0;ip<40;ip++)
          {
          if(ip<32)
          setcolor(2);
          else
          setcolor(15);
          rectangle(left+ip,top+ip,right-ip,bottom-ip);
          }



          //==============================================================
          m=left+ip;
          n=right-ip;
          c=top+ip;
          d=bottom-ip;
          k=n-m;
          w=d-c;
          u=k/2;
          v=w/2;
          setcolor(4);
          line(m,v+c,n,v+c);
          line(u+m,c,u+m,d);
          return(m);
          /* clean up */
          }


           /// graph end1


///==========================================================
void main()
{
                    unsigned int i;
                    float      DataBuf1[1000];
                    float ud,pwr[1000],en_gy[1000];
                            int x1,y1,ia,ib,ck1,ck2,sp1,sp2,sn1,sn2,z,t1,t2,b1,b2;
                            float m1,energy,f1,f2;



                    // clrscr() ;
                    FILE *fp1;
                    FILE *fp2;
                    fp1=fopen("c:\\plot1.txt","wt");
                    fp2=fopen("C:\\plot2.txt","wt");
                    clrscr();
                    for(i=0;i<1000;i++)
                    {
                    pwr[i]=0.0;
                    DataBuf1[i]=0.0;
                    en_gy[i]=0.0;
                    }
           //       f1=0.0;
          //      f2=0.0;
                    energy=0;

                    //============================================
                    dat= data;
                    param[0]  = 0;          /* Board number                    */
                    param[1]  = 0x300;        /* Base I/O address                  */
                    param[5]  = 10;          /* Pacer rate = 10M / (50 * 100) = 2000 Hz  */
                    param[6]  = 10;
                    param[7]  = 0;          /* Trigger mode, 0 : pacer trigger       */
```

```
        param[10] = FP_OFF(dat);  /* Offset of A/D data buffer A             */
        param[11] = FP_SEG(dat);  /* Segment of A/D data buffer A            */
        param[12] = 0;            /* Data buffer B address, if not used,    */
        param[13] = 0;            /* must set to 0.                         */
        param[14] = 900;           /* A/D conversion number                  */
        param[15] = 2;            /* A/D conversion start channel           */
        param[16] = 3;            /* A/D conversion stop channel            */
        param[17] = 0;            /* Overall gain code, 0 : +/- 5V          */

        /* param[18] = FP_OFF(gain_array);
                param[19] = FP_SEG(gain_array);  */

        /* param[45] : Error code
                param[46] : Return value 0
                param[47] : Return value 1 */

        pcl818HG(3, param);       /* Func 3 : Hardware initialization       */
        if (param[45] != 0) {
                        printf(" DRIVER INITIALIZATION FAILED !");
                        exit(1);
                        }


        pcl818HG(4, param);       /* Func 4 : A/D initialization            */
        if (param[45] != 0) {
                        printf(" A/D INITIALIZATION FAILED !");
                        exit(1);
                        }
                        m1=rectangl_draw();

        /* Functin 5 will use FIFO, it must enable hardware FIFO.            */
        do
        {


        pcl818HG(5, param);       /* Func 5 : "N" times of A/D trigger      */
        if (param[45] != 0)
         {
        printf(" A/D SOFTWARE DATA TRANSFER FAILED !");
        exit(1);
        }


 ///=============================================================
        //graph start2


//=========================================================



x1=getmaxx();
x1=x1/2;
y1=getmaxy();
```

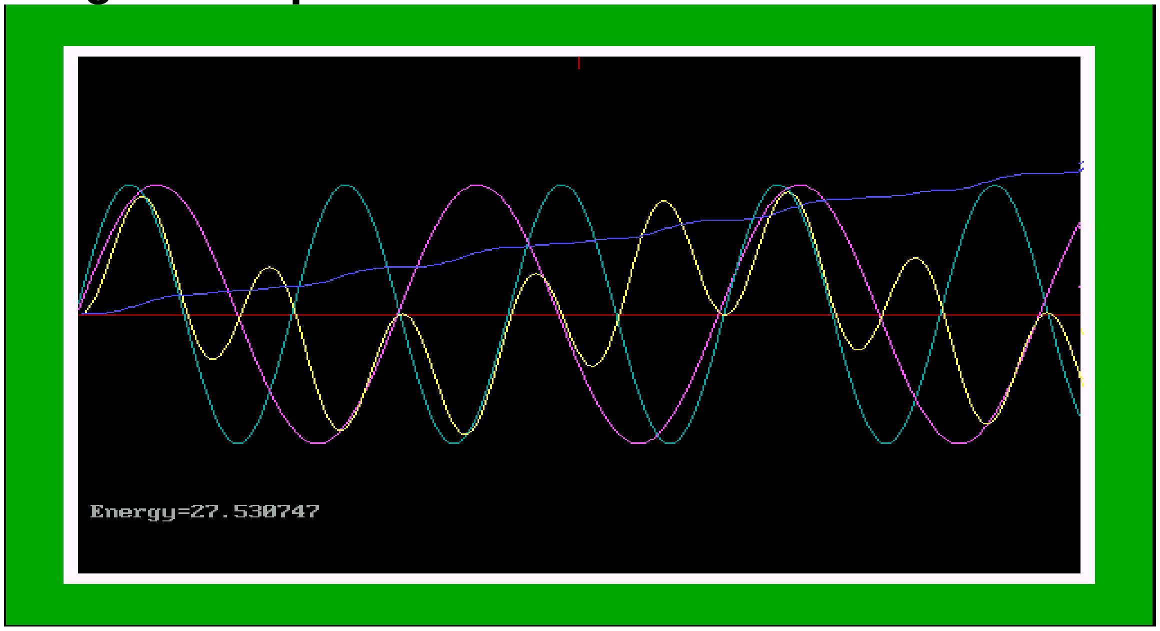
```
            y1=y1/2;
            setcolor(9);


                        //================

                        // ud=rectangl_draw();

                         for (i = 0; i < param[14]; i++)    /* Display data
                                                                                                    */
             {
             if(i==0||i%2==0)

             {

                         DataBuf1[i] = data[i] & 0xFFF;
                         DataBuf1[i] =(((5 - (-5)) * DataBuf1[i] / 4096) + (-5));
                         fprintf(fp1,"\n%f",DataBuf1[i]);
              }
                        else
                        {

                         DataBuf1[i] = data[i] & 0xFFF;
                         DataBuf1[i] =(((5 - (-5)) * DataBuf1[i] / 4096) + (-5));

                         /*
                                    (5 - (-5)) : A/D input range (-5V  to  5V)
                                    4096       : Full scale 12 bit A/D data
                                    DataBuf    : A/D input data
                                    (-5)       : A/D input range "-5" V
                         */
                        // printf("\ndata[%3d] = %f V ", i, DataBuf1[i]);
                         fprintf(fp2,"\n%f",DataBuf1[i]);
                         }
              }




//========================%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%




            ia=0;
            ib=0;
            z=0;
            ck1=0;
            ck2=0;
            sp1=0;
            sp2=0;
            sn1=2;
            sn2=3;
            t1=0;
            t2=0;
            b1=0;
            b2=0;
            for(i=2;i<700;i++)
            {
            if(i%2==0)
            {
            if(((DataBuf1[i]>0&&DataBuf1[i]<.1)&&(DataBuf1[i]-DataBuf1[i-2]>0))||ck1==1)
            {
            if(sp1<555)
            {
            setcolor(13);
  line(m1+ia,y1-25*DataBuf1[i],m1+ia+2,y1-25*DataBuf1[i+2]);
  }
  else
  {
  ck1=1;
  }
                        ia=ia+2;
                        ck1=1;
                        sp1=sp1+2;
                        t1=sn1;
                        }
                        else
                        {
                        sn1=sn1+2;
                        ck1=0;
                        t1=0;
                        }
                        }

                        else
                        {
                                    if(((DataBuf1[i]>0&&DataBuf1[i]<.1)&&(DataBuf1[i]-DataBuf1[i-
2]>0))||ck2==1)
                        {
                        if(sp2<555)
                        {
                        setcolor(3);
                                    line(m1+ib,y1-25*DataBuf1[i],m1+ib+2,y1-25*DataBuf1[i+2]);
                                    }
                                    else
                                    {
                                    ck2=1;
                                    }
                                    ck2=1;
                                    ib=ib+2;
                                    sp2=sp2+2;
                                    t2=sn2;
```

```
                    }
                    else
                    {
                    sn2=sn2+2;
                    ck2=0;
                    t2=0;
                    }
                    }



          /*

//============================%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if((ck1==1&&ck2==1)&&t1>t2)
{
if(z==0)
{
pwr[z]=DataBuf1[i]*DataBuf1[t2+z]/100;
en_gy[z]=fabs(pwr[z]*.01);
z=z+1;

}
else if(z!=0&&z%2==0)
{
pwr[z]=DataBuf1[i]*DataBuf1[t2+z]/100;
en_gy[z]=fabs(pwr[z]*.01);
en_gy[z]=en_gy[z]+en_gy[z-2];
energy=energy+en_gy[z];
setcolor(14);
line(m1+z,y1-pwr[z-2],m1+z+2,y1-pwr[z]);
setcolor(9);
line(m1+z,y1-en_gy[z-2],m1+z+2,y1-en_gy[z]);
z=z+1;
}
else
{
pwr[z]=0.0;
z=z+1;
}
}
else if((ck1==1&&ck2==1)&&t2>t1)
{
if(z==0)
{
pwr[z]=(DataBuf1[t1+z]*DataBuf1[i])/100;
en_gy[z]=fabs(pwr[z]*.01);
z=z+1;
}
}
else if((ck1==1&&ck2==1)&&t2>t1)
{
if(z==0)
```

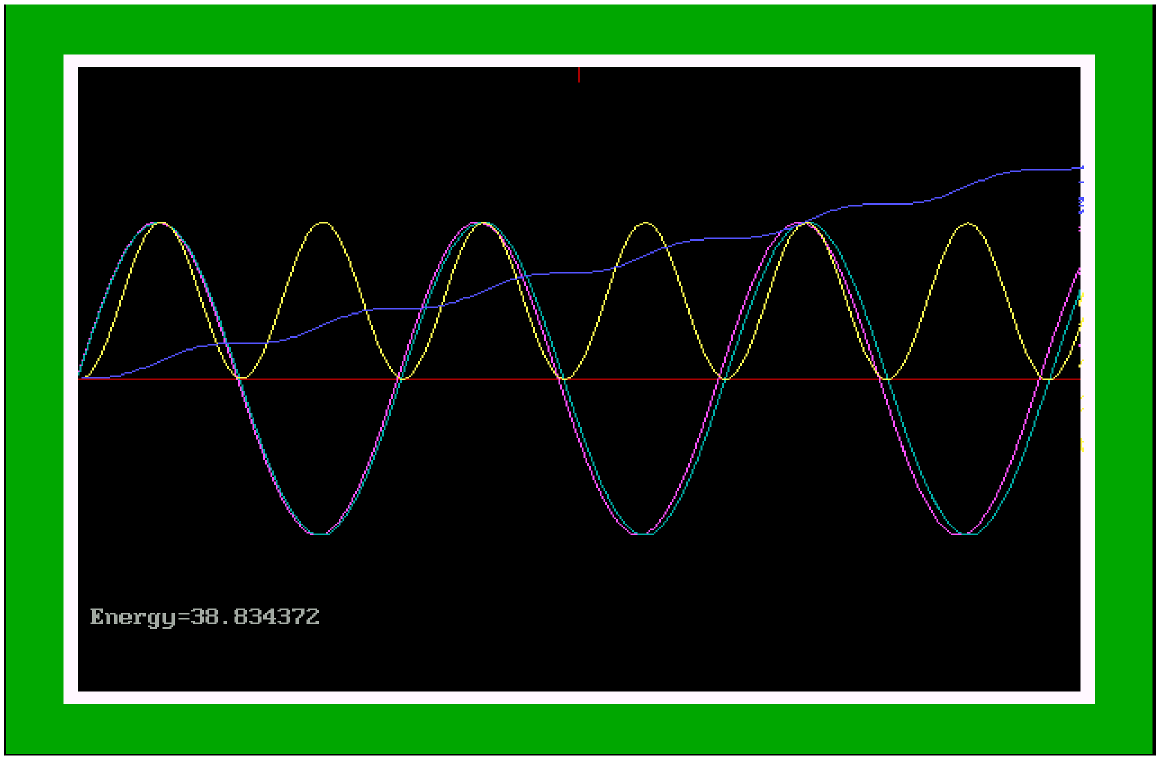
```
{
pwr[z]=(DataBuf1[t1+z]*DataBuf1[i])/100;
en_gy[z]=fabs(pwr[z]*.01);
z=z+1;
}
else if(z!=0&&z%2==0)
{
pwr[z]=(DataBuf1[t1+z]*DataBuf1[i])/100;
en_gy[z]=fabs(pwr[z]*.01);
en_gy[z]=en_gy[z]+en_gy[z-2];
energy=energy+en_gy[z];
setcolor(14);
line(m1+z,y1-pwr[z-2],m1+z+2,y1-pwr[z]);
setcolor(9);
line(m1+z,y1-en_gy[z-2],m1+z+2,y1-en_gy[z]);
z=z+1;
}
else
{
pwr[z]=0.0;
z=z+1;
}
}
else
{
printf("");
}
//================================
gotoxy(7,25);
printf("ENERGY=%f",energy/10000);
//=========================================================
*/

if((sp1==6&&i%2==0)&&fabs(fabs(DataBuf1[i]-DataBuf1[i-2])-f1)>.1)
{
f1=fabs(DataBuf1[i]-DataBuf1[i-2]);
for(i=2;i<558;i++)
{
setcolor(0);
line(m1+b1,50,m1+b1+1,438);
b1=b1+1;
}
setcolor(4);
line(m1,y1,2*x1-m1,y1);
break;
}
if((sp2==6&&i%2!=0)&&fabs(fabs(DataBuf1[i]-DataBuf1[i-2])-f2)>.1)
{
f2=fabs(DataBuf1[i]-DataBuf1[i-2]);
for(i=2;i<558;i++)
{
setcolor(0);
line(m1+b2,50,m1+b2+1,438);
```

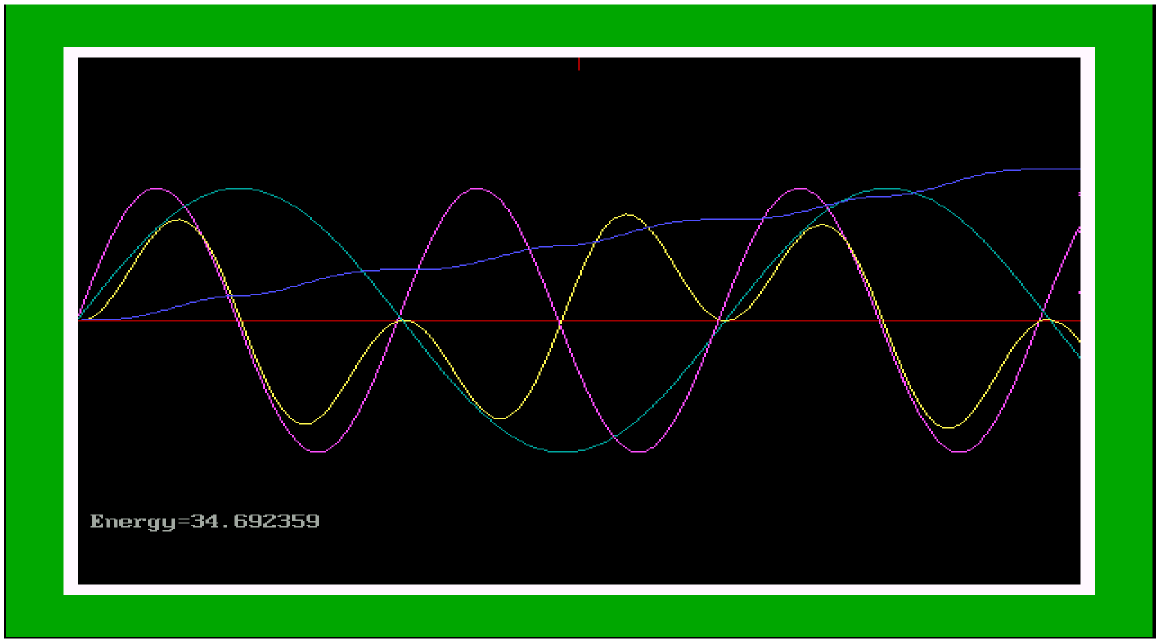
```
                b2=b2+1;
                }
                setcolor(4);
                line(m1,y1,2*x1-m1,y1);
                break;
                }
                }
                //=============================================================
                }while(!kbhit());

              //  fclose(fp1);
}
```

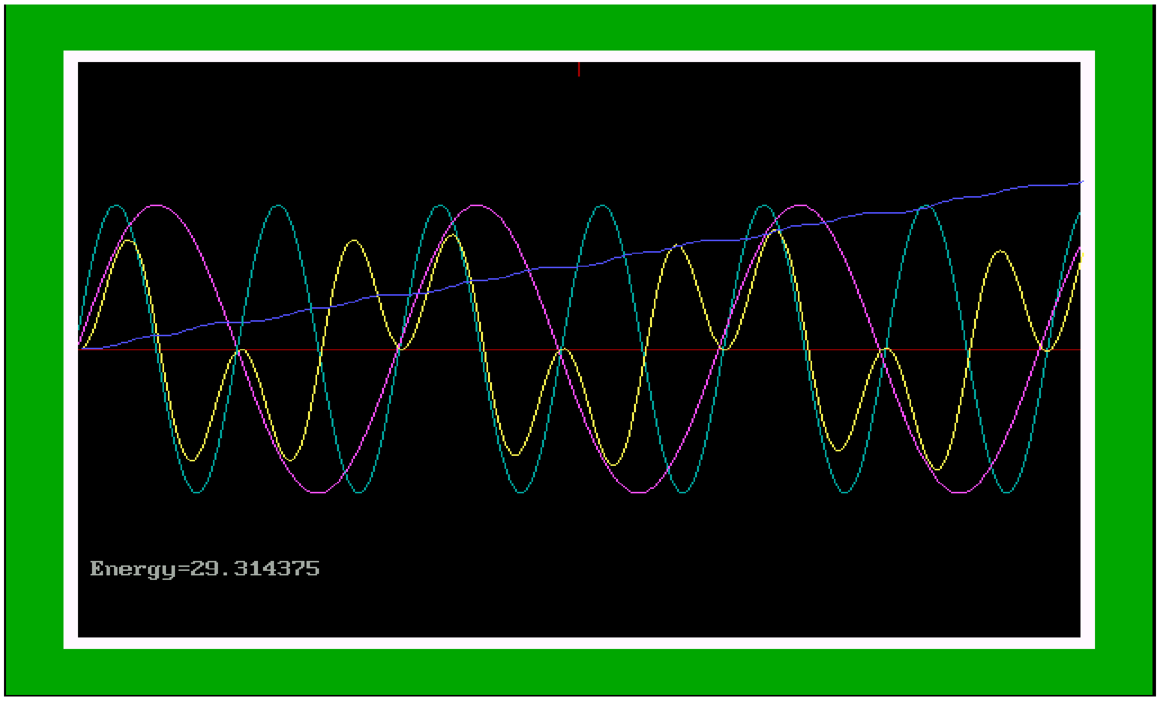
Program output:::::

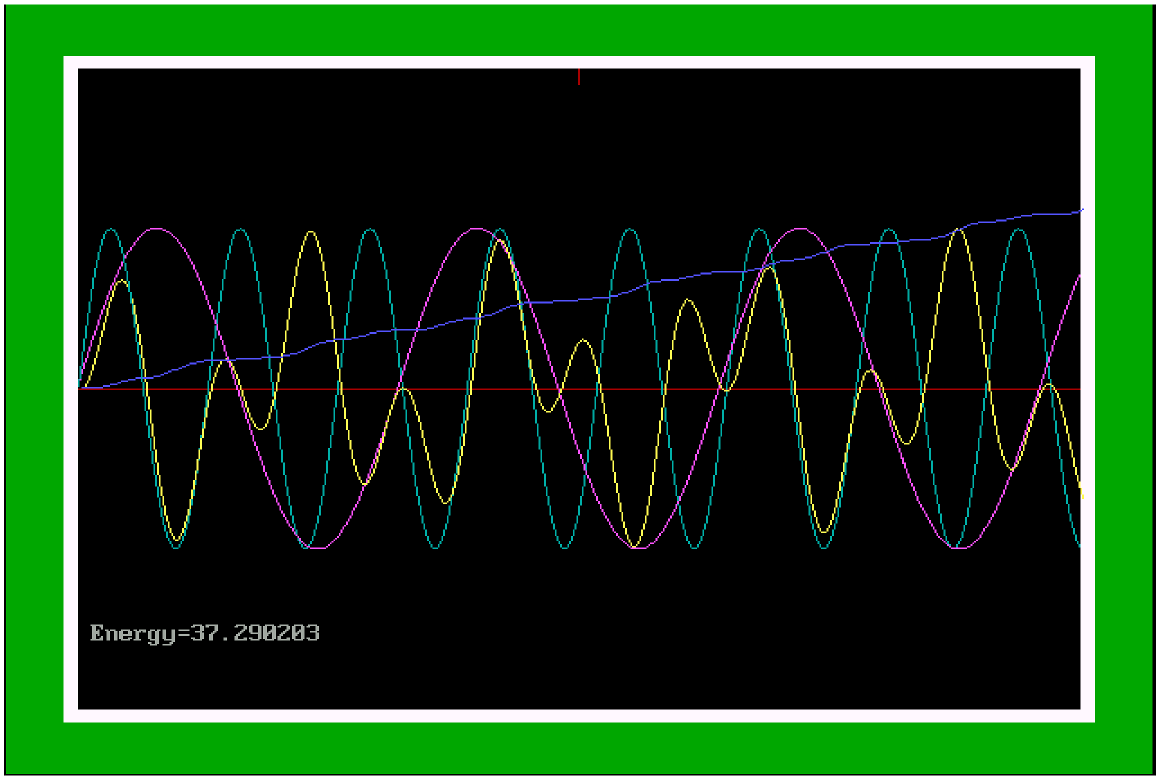


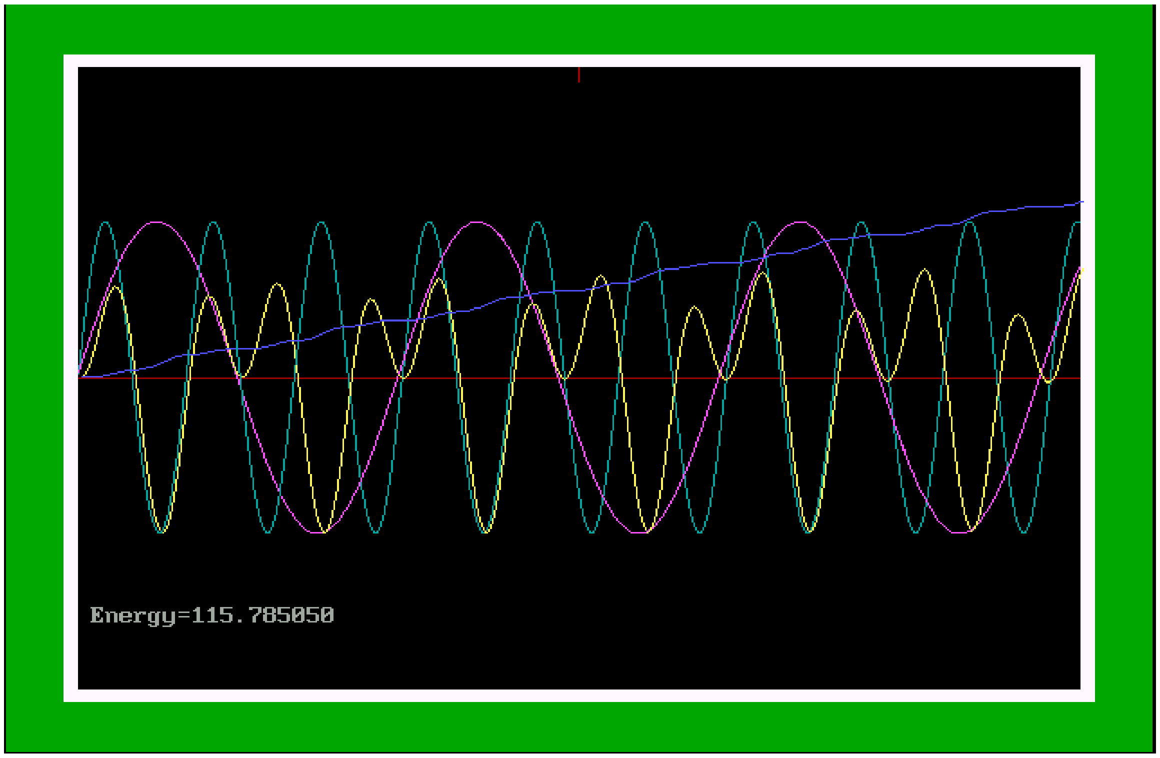
```
Energy=27.530747
```


Energy=38.834372


Energy=34.692359


Energy=29.314375


Energy=37.290203



Energy=115.785050

Development of computerized standard energy meter:

Fundamental Theory:

Energy:

Energy is the total power delivered or consumed over a period of time.
That is ,
Energy = power x time
Electrical energy(E) deve3loped over an interval of time t may be expressed as:

$$E=\int_{0}^{t} vi\,dt$$

Where ‘v’ is the instanteneous voltage and ‘i’ is the instanteneous current.
In digital domain, it is defined as follows:

$$E= \sum vi\Delta t$$

We used the above formula for energy calculation.
Analog to digital conversion::::::
Analog signal is converted into digital form in our data acquisition card. For this purpose we follow two basic processes namely:::
1)Sampling
2)Quantization

Sampling::::::

The sampling process is usually described in the time domain.It is an operation that is basic to digital signal processing, an analog signal is converted into a corresponding sequence of samples that are usually spaced uniformly in time. For practice utility it is necessary that we choose the sampling rate properly, so that the sequence of samples uniquely defines original analog signal.

Quantization::::

The process of converting a sampling signal into a signal by expressing each sample value as a finite number of digits is known as quantization. A continuous signal has continuous range amplitudes and therefore its samples have a continuous amplitude range. Inother words, within the final amplitude range of the signal, we find a finite number of amplitude levels. Any human sense can detect only finite intensity difference. This means that the original continuous signal may be approximated by a signal constructed of discrete amplitudes selected ona minimum error basis from an available set. If we make the approximated signal practically indistinguishable from the original continuous signal. Quantization is done in ADC in our data acquisition card.

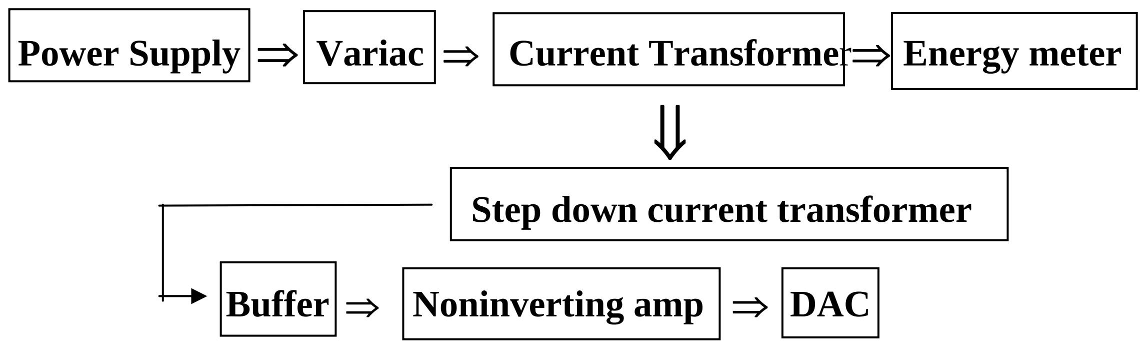
Block diagram for analog to digital conversion:::

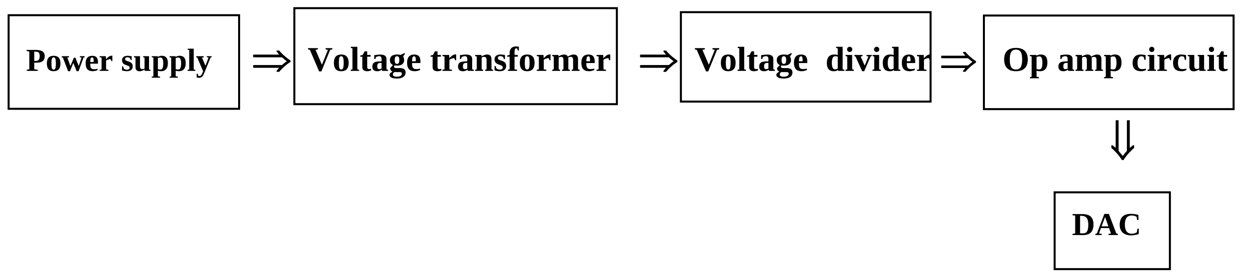
$$\begin{tikzcd} \boxed{\text{Analog signal}} \arrow[r, Rightarrow] & \boxed{\text{Sampling}} \arrow[r, Rightarrow] & \boxed{\text{Quantization}} \arrow[r, Rightarrow] & \boxed{\text{Digital signal}} \end{tikzcd}$$

A/D converter::::::::
An analog to digital converter takes an analog input voltage and after a certain amount of time produces a digital output code which represents the analog input.
There are many different types of ADCs. Among them most widely used are

1)Digital ramp ADC
2)Integration type ADC
3)Successive approximation
4)Flush ADC
Among these ADCs successive approximation ADC is used in Data Acquisition Card.

BLOCK DIAGRAM OF COMPUTERIZED STANDARD ENERGY METER





In the above block diagram-there are two units---------------------
1) One side is for voltage and this one is applied into a channel of DAC
2) Another side is equivalent to current and this one is applied as voltage into another channel of DAC

Current transformer (step down)::::::

While developing the computerized energy meter we used the current transformer for reducing the current value. The connection of this current transformer is little bit surprising. Here in the secondary terminal we got reduced current completely by induction process.

Specification::::

Type MSQ-40
500/5 A
10VA

Application

The series of current transformers can be applied to test, control, display and record the running of the electrical equipment, and to protect the equipment against the damage, in the AC circuit with the rated voltage value below 660V and the frequency of 50-60Hz, The product can be also applied to form a complete set of mine transformer.

Variac::::

We used a variac for getting high current. The primary side was supplied by power supply(220v) and the secondary terminal of this variac was shorted and we got very high current. For quick calibration this variac really helped us a lot. Due to this high current the aluminium disc of energy meter rotated very fast.

Variac:

We offer Variac transformers (variable transformers) with ratings from 120V to 600VAC single and three phase and fractional to 700 Amps. Variac transformers are AC voltage controls that provide a variable AC voltage. Their distortion free output is ideal for sensitive electronic applications. Variac transformers are available that boost the output voltage in excess of twice the input voltage. Input frequencies to 2000 Hz can be easily accommodated.

Current transformer::::(step up)

We also used another type of current transformer for increasing the value of higher current. The current what we got from the variac that is supplied to this current transformer and current is ten times increased. This high current was supplied to energy meter.

Application:

These commercial grade, low cost window type current transformers are used with ammeters in control panels and engine generators. These CT's can be supplied with leads or terminals and mounting brackets. It can be used for most metering and relaying applications in low voltage switchboards, switchgear and motor control centers. A wide range of window sizes and ratios up to 10,000 amperes is available.

Energy meter:

For calibration we used a normal single phase analog energy meter which is extensively used in our house.

Op-amp circuit::::

For eliminating loading effect problem we used op-amp circuit. For amplification purpose, we also used it. At that time non-inverting amplifier circuit was implemented in the breadboard.