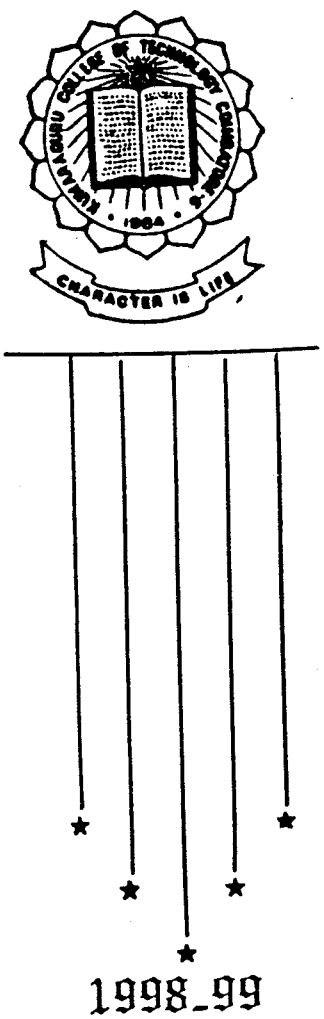
Test Jig for Yarn Quality Monitor



Project Report

Submitted by

KAUSHIK SUNDARAM
S. MURALIDHARAN

Guided by

Dr. K.A. PALANISWAMY
B.E , M Sc.(Engg), Ph.D, MISTE., C. Eng(I), FIE.

IN PARTIAL FULFILMENT OF THE REQUIREMENT
FOR THE AWARD OF THE DEGREE OF
BACHELOR OF ENGINEERING IN
ELECTRICAL AND ELECTRONICS ENGINEERING
OF THE BHARATHIAR UNIVERSITY COIMBATORE

Department of Electrical and Electronics Engineering
Kumaraguru College of Technology
Coimbatore_641 006

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Kumaraguru College of Technology

COIMBATORE - 641 006.

CERTIFICATE

This is to certify that the Project entitled

TEST JIG FOR A YARN QUALITY MONITOR

Has been submitted by

Mr. S.Muralidharan, Kaushik Sundaram

In partial fulfillment of the requirements for the award of degree of

Bachelor of Engineering in the Electrical & Electronics Engineering

branch of the Bharathiar University,

Coimbatore - 641 046 during the academic year 1998-99.

17/3/99

Guide

Professor and Head

PROFESSOR & HEAD

Dept. of Electrical & Electronics Engg.

Kumaraguru College of Tech., Coimbatore-

Certified that the candidate with University Register

No. 9527C0084

Was examined in project work Viva-Voce

Examination held on 17-3-99

Internal Examiner

External Examiner.

PREMIER

CERTIFICATE

This is to certify that the following students of **Kumaraguru College of Technology, Coimbatore.**

G. Ramanathan	**(ECE)**
V.K. Pradeep	**(ECE)**
S. Muralidharan	**(EEE)**
Kaushik Sundaram	**(EEE)**

had undertaken their Final year Project entitled,

" TEST JIG FOR A YARN QUALITY MONITOR "

in the academic year 1998 to 1999 in our Industry and have successfully completed it.

Their performance during that period was found to be good.

We wish them all success.

Guided by

Mr. Neel Mathews

Mr. K K VENKATARAMAN
Vice President (R & D)

Place : Coimbatore

Date : 08th March '99

premier polytronics ltd.
304, Trichy Road • Singanallur • Coimbatore - 641 005. India.
Tel : 0422 - 573548 • Fax : 0422 - 573651
Visit us : http://www.premier-1.com • Email : mail@premier-1.com
0 9 6 TNGST RC No. 1820282 CST RC No. 289178 Dated : 23.6.82
Regd. Office 185, A.T.D. Street, Race Course, Coimbatore - 641 018

ACKNOWLEDGEMENT

We take this opportunity to express our gratitude to ***Dr.K.A.Palaniswamy****,B.E.,M.Sc(Engg),Ph.D.,MISTE.,C.Eng(I),FIE., Professor and Head of Department of Electrical & Electronics Engineering, Kumaraguru College of Technology, Coimbatore for guiding and motivating us all along the way in the completion of the project.*

We wish to thank our beloved Principal ***Dr. K. K. Padmanabhan****, B.Sc(Engg),M.Tech.,Ph.D., for providing the facilities for this project work.*

We also express our gratitude to ***Mr. T. R. Padmanabhan*** *, Head of R & D and* ***Mr. K .K. Venkataraman****, Head of Yarn Section, R & D, Premier Polytronics Ltd., without whom this project would not have been possible.*

We would be failing in our duty, if we do not express our heart-felt gratitude to ***Mr. Neel Mathews****, Senior Engineer , R & D , Premier Polytronics Ltd., for guiding us right from the conceptualization to the materialization of this project.*

We would also like to express our gratitude to Mr. Vasanth and Mr. Murali , R& D, PPL., for their valuable suggestions.

SYNOPSIS

'QUALITY IS NO ACCIDENT ', It is the end result of sustained improvement and continuous evaluation of the existing systems.

It is with this in mind that this project of designing and developing a TEST SYSTEM for a yarn quality monitor had been undertaken. This yarn quality monitor ,'Q-CONE ',is a new product being developed by Premier Polytronics Ltd., Coimbatore.

In conventional yarn quality monitors ,the yarn deformities (faults) are just identified and removed . The Q-CONE differs from its predecessors in that it identifies, classifies and stores the information about the faults as well, before sending the required control signals to the related equipments for removing the faults. The stored data helps in future reference as well as in enhancing the overall quality.

The testing process involves the simulation of the actual yarn along with a wide range of fault signals and then ,testing the performance of Q-CONE over a wide range of inputs .The various processes involved are enabled by the add-on card and a set of software present in the TEST SYSTEM.

By comparing the results of Q-CONE with the simulated faults of the TEST SYSTEM, the performance evaluation of the Q-CONE is conducted.

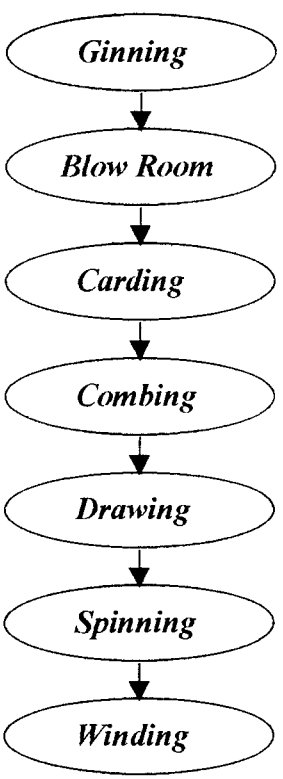
CONTENTS

INTRODUCTION

Mankind has indeed come a long way, right from the era of fig leaves to the present world of high fashion apparel. Today's clothing represents both the life style as well as the status in the society. As the saying goes 'Clothes maketh the man', so the quality of the fabric is given due importance. Hence the need for yarn quality control comes into the picture as the yarn forms the basic element of any fabric.

1.1 From raw cotton to Yarn:

The various stages involved in the process are given by the following flow diagram :



For further details refer appendix.

1.2 Importance of yarn quality:

We have already seen that the quality of the yarn determines the final finish and the texture of the fabric. For these reasons, efforts are being made to continuously check the quality of all the processes involved in the manufacture of yarn right from the stage of cotton selection to the final stage of winding. Yarn quality means that the thickness of the yarn should be within certain prescribed limits. In case the thickness increases beyond the prescribed it leads to uneven finishing and patches in the fabric or if the thickness is less than the prescribed limits,the strength of the fabric is reduced. Our project involves quality control in the last stage i.e., the winding stage.

The variations in the thickness of the yarn beyond the allowable mean thickness is defined as a *fault*. Faults can be classffied into different categories on the basis of the deviations from the prescribed thickness & length for which the deviation exists. We shall see the exact classification in the next section.

1.3 Fault classification:

Yarn faults **are** classified into 24 different categories based on the thickness and the length for which the deviation persists.

The faults which have the deviation from the mean thickness (% error) greater than 45% are classified as thick faults and the faults with % error less than -30% are classified as thin faults.

The exact classification is given in Fig 1.1.

1.4 Yarn clearing:

Certain categories of faults may not be allowed in the yarn. In such a case that particular length of the yarn is removed from the cone . This process is called clearing. The type of faults for which the clearing takes place can be set by the user by fixing the appropriate clearing curve.

1.5 Need for Monitoring:

Conventional clearers do not give any information about the faults which have been cut. The Q-Cone actually classifies the faults and stores the information before sending the control signal to the clearer. This enables the user (Textile Industry) to maintain quality standards which might be required for later reference, so that international requirements could be met.

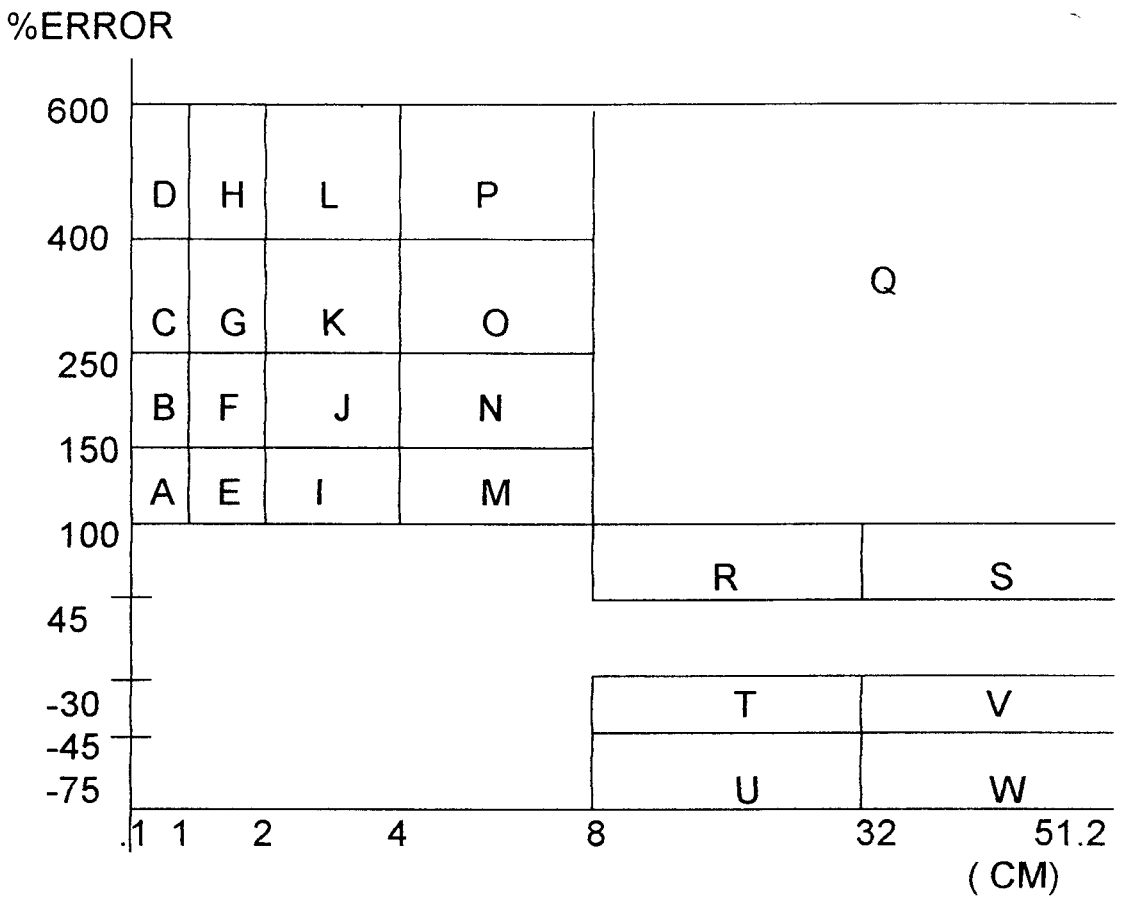


Fig 1.1 : Fault Classification Graph.

PROJECT OVERVIEW

2.1 AIM OF THE PROJECT :

The aim of the project is to design and develop a Test System that will test the performance of a Q-CONE and similar products over a wide range of yarn samples under varying conditions.

The essential components of the test system are

1. PCL-812 Add – on card.
2. PC with serial communication hardware.
3. Software.

The PCL –812 card is an Advantech Co. product. This is an industry standard data acquisition card. It has a provision for multiple digital inputs and outputs. In addition there are on-board DAC,ADC,Timers & Analog I/O Channels. This makes it a

multifunctional card for PC controlled data acquisition and process monitoring. (Further details regarding the various functions are explained in Chapter 3.)

The sequence of the working of the test system and the card is controlled by a set of software. The test system communicates with the Q-CONE via the serial port of the PC used in the test system, under the control of the software.

As the product 'Q-CONE' is in the development stage, we were required to make a few changes. We have simulated the operation of the Q-CONE with another PCL – 812 card and a set of our software programs. The Q-CONE was then replaced with the simulated version. This is a more general system which, with a few minor modifications can be extended to a whole family of similar products.

2.2 PROJECT METHODOLOGY :

The sequence of operations and Modules involved in the Test System are shown in fig 2.1.

The testing system verifies the ability of the Q-CONE to identify, classify and enumerate the various faults generated by the software.

The procedure of testing is as follows :

1. The actual yarn faults can be generated by the user(tester) according to his choice by using the software and the DAC.

   For example we can simulate a particular length of yarn with faults as AZZBZCZFZAZKBAZ
2. The analog signal corresponding to the fault sequence using the DAC in the PCL-812 Add-on card.

3. The Q-CONE senses this analog voltage and converts into digital information of % error using the ADC in the PCL – 812 Add-on card.

4. The counter 8253 in the PCL-812 Add-on card is used to find the length of the fault and this is performed by the fault identification module.
5. Using the error % and length information, the fault is classified using the fault classification module.
6. The number of faults identified in each category is counted.
7. The enumerated fault information is serially communicated to the testing equipment by RS – 232 serial ports in each system.
8. A comparison is made between the actual fault simulated by the user and the output of the Q-CONE and the result is displayed.

The flow diagram of the operations is given in Fig 2.2.

The Various Modules in the project are:

1) Fault generation

2) Fault identification

3) Fault Classification & Enumeration

4) Serial Communication & Result Validation

5) Need for data compression

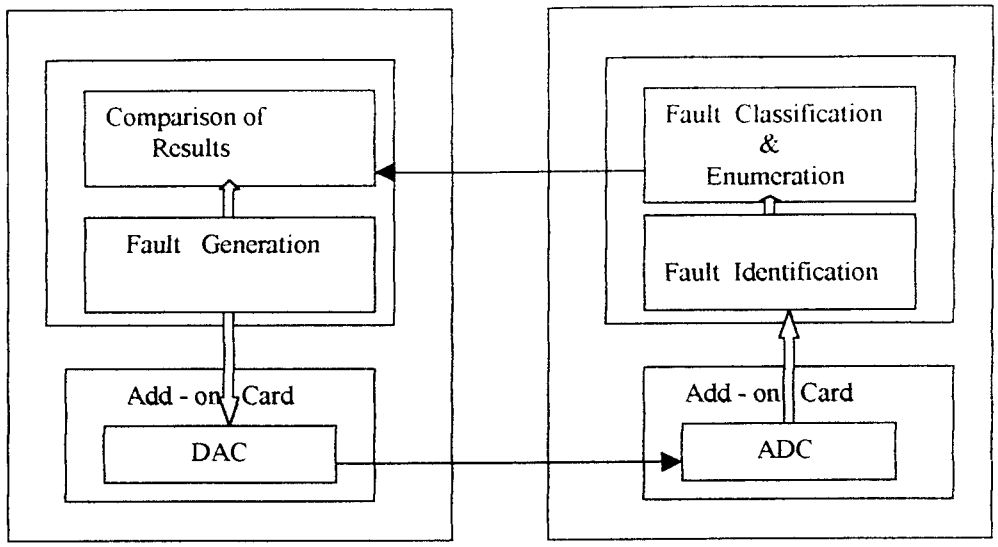


Fig 2.1 : Block diagram of Test system.

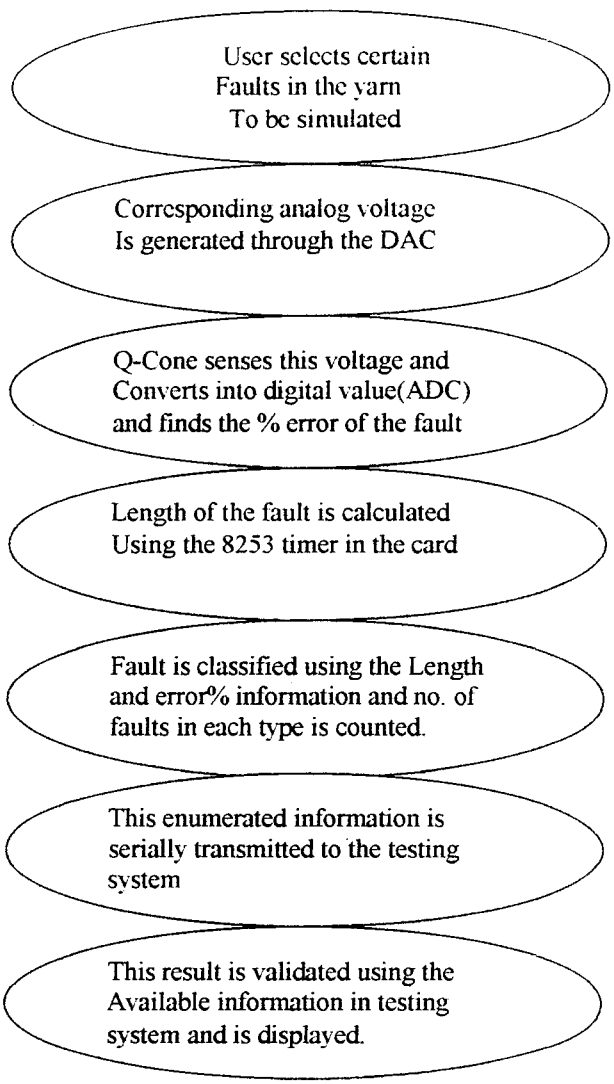


Fig 2.2 : Flow Diagram of Modules.

PCL – 812 ADD-ON CARD DETAILS

3.1 Introduction to PCL - 812:

The PCL-812 is a high performance, high speed, multi-function data acquisition card for IBM PC/XT/AT and compatible computers. The high-end specifications of this full-sized card, and complete software support from third party vendors make it ideal for a wide range of applications in industrial and laboratory environments. These applications include data acquisition, automatic testing and factory automation.

3.2 Key features:

* 16 single ended analog input channels.

* An industrial standard 12-bit successive approximation converter to convert analog inputs. The maximum A/D sampling rate is 30 Khz in DMA mode.

3.3.2 ANALOG OUTPUT (DAC)

* Channels : 2 channels.

* Resolution : 12 bits.

* Output Range : 0 to +5V with fixed -5 V reference , (+/-) 10 V with external DC or AC reference.

* Reference Voltage : Internal : -5V

External : DC or AC , (+/-)10 V max.

* Conversion Type : 12 bit monolithic multiplying.

* Analog devices : AD7541/AKN or equivalent.

* Linearity : (+/-) 1/2 bit.

* Output drive : (+/-) 5 mA max.

* Settling time : 30 microseconds.

* Switch selectable versatile analog input ranges.

  Bipolar : 1V, 2V, 5V, 10V (plus/minus)

* Three A/D trigger modes: Software trigger

  Programmable pacer trigger

  External trigger pulse trigger.

* The ability to transfer A/D converted data by program control, interrupt handler routine or DMA transfer.

* An INTEL 8253 programmable timer/counter provides pacer output (trigger pulse) at the rate of 0.5 MHZ to 35 minutes/pulse to the A/D. The timer time base is 2 MHZ. One 16-bit counter channel is reserved for user configurable applications.

* Two 12-bit monolithic multiplying D/A output channels. An output range from 0 to +5V can be created by using the onboard -5V reference. This precision reference is derived from A/D converter reference. External AC or DC references can also be used to generate other D/A output ranges.

* 16 TTL/DTL compatible digital input, and 16 digital output channels.

3.3 PRODUCT SPECIFICATIONS:

3.3.1 ANALOG INPUT (ADC)

* Channels : 16 Single-ended.

* Resolution : 12 bits.

* Input Range : Bipolar : 10 V , 5 V, 2 V, 1 V. All input ranges are switch selectable.

* Overvoltage : Continuous (+/-) 30 V max.

* Conversion type : Successive Approximation.

* Convertor : ADC574 or equivalent.

* Accuracy : 0.015 % of grading (+/-) 1 bit.

* Linearity : (+/-) 1 bit.

* Trigger mode : Software trigger , on-board programmable timer or external trigger.

* Data transfer : Program control, interrupt control or DMA.

* External trigger: TTL compatible, load 0.4 mA max. at 0.5 V (low) or 0.05 mA max. at 2.7V (high).

3.3.3 PROGRAMMABLE INTERVAL TIMER/COUNTER

* Device : INTEL 8253.

* Counters : 3 channels, 16 bit, 2 channels permanently connected to 2 MHZ clock as programmable pacer, 1 channel free for user application.

* Input gate: TTL/DTL/CMOS compatible.

* Time base : 2 MHZ.

* Pacer Output : 35 minutes/pulse to 0.015 MHZ.

More details on programmable interval timer/counter is furnished in section 3.7.

3.4 ON BOARD JUMPER SETTINGS:

The PCL - 812 has two DIP switches and 7 jumpers and the function of each switch is discussed in this section.

3.4.1 BASE ADDRESS SELECTION:

Switch name : **SW1.**

Most PC peripheral devices and interface cards are controlled through the input/output ports. These ports are addressed using the I/O port address space. The I/O port base address of the PCL_812 is selectable via an 8 position dip switch. The add-on card requires 16 consecutive address locations in I/O space. Valid addresses are from Hex 200 to Hex 3F0, however used some of these addresses are for other devices. The PCL-812 card base address switch setting is set to Hex 220 in the factory. If the card is to be adjusted another address range, the switch settings for various base addresses are illustrated as below:

I/O Address Range (Hex)	Switch Position 1	2	3	4	5	6
	A9 (Fixed)	A8	A7	A6	A5	A4	A3
200-20F	1	0	0	0	0	0	X
210-21F	1	0	0	0	0	1	X
220-22F*	1	0	0	0	1	0	X
220-23F	1	0	0	0	1	1	X
300-30F	1	1	0	0	0	0	X

Note : ON = 0, OFF = 1

A4 ..... A9 correspond to PC bus address lines.

* factory setting.

3.4.2 WAIT STATE SELECTION:

Some high speed PC' s may require that a wait state is added to the PCL-812 to achieve stable data transfer. The PCL-812 can be configured with 0,2,4, or 6 wait state delays for each transfer of data. Length of the wait state can be selected with the pins 7 and 8 on SW1 as shown below.

Switch Position		Time delay
7	8	
0	0	0
1	0	2
0	1	4
1	1	6

3.4.3 BIPOLAR INPUT RANGE SELECTION

Switch name : **SW2**

The specific analog input range within the bipolar group is selected by this 4 position DIP switch. The following table illustrates the switch setting and corresponding input ranges.

Switch Position				Bipolar Range
1	2	3	4	
ON	OFF	ON	OFF	(+/-) 10V
OFF	ON	ON	OFF	(+/-) 5V
ON	OFF	OFF	ON	(+/-) 2V
OFF	ON	OFF	ON	(+/-) 1V

3.4.4 DMA CHANNEL SELECTION

Jumper name : JP5, JP6

The PCL-812 provides DMA data transfer capability. The selection of DMA level 1 or level 3 is controlled by this jumper.

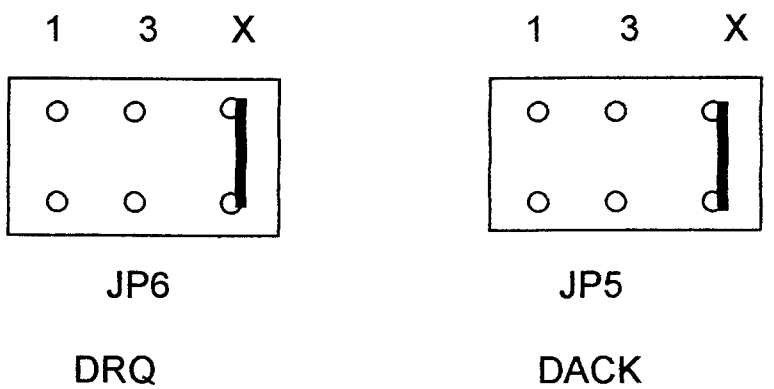


Fig 3.1 : Jumper settings for DMA channel selection.

3.4.5 TRIGGER SOURCE SELECTION

Jumper name : JP1

Fig 3.2 : Jumper settings for Trigger source selection.

3.4.6 USER'S COUNTER INPUT CLOCK SELECTION

Jumper name : JP2

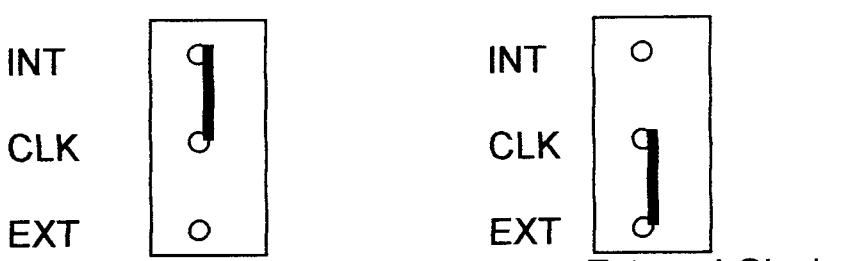
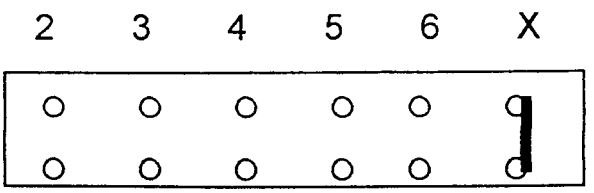


Fig 3.3: Jumper settings for counter input clock selection.

3.4.7 IRQ LEVEL SELECTION

Jumper name : JP4



JP4 (IRQ)

Fig 3.4: Jumper settings for IRQ level selection.

3.4.8 D/A REFERENCE SOURCE SELECTION

Jumper name : JP7, JP8

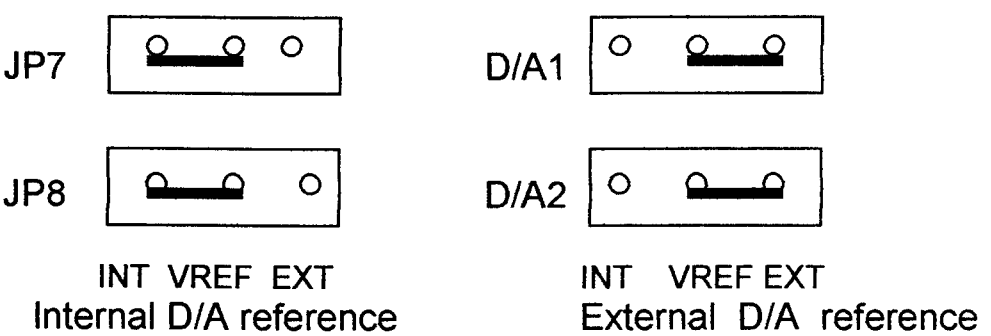


Fig 3.5: Jumper settings for D/A reference source selection.

3.5 Connector pin Assignment:

The PCL-812 is equipped with two 20-pin insulation displacement (mass termination) connectors, accessible from the rear plate three other 20-pin insulation displacement connectors on-board. All these connectors can be connected to the same type of flat cables. The following diagrams illustrate the pin alignment of each connector.

Legend :

A/D - Analog input
A.GND - Analog ground
D/A - Analog output
D/O - Digital output
D/I - Digital input
D.GND - Digital ground
CLK - Clock input for the 8253
GATE - Gate input for the 8253
OUT - Signal output of the 8253
VREF - Voltage reference

Connector 1 (CN 1) Analog Input

A/D 0	1	2	A.GND
A/D 1	3	4	A.GND
A/D 2	5	6	A.GND
A/D 3	7	8	A.GND
A/D 4	9	10	A.GND
A/D 5	11	12	A.GND
A/D 6	13	14	A.GND
A/D 7	15	16	A.GND
A/D 8	17	18	A.GND
A/D 9	19	20	A.GND

Connector 2 (CN 2) Analog Input/Output

A/D 10	1	2	A.GND
A/D 11	3	4	A.GND
A/D 12	5	6	A.GND
A/D 13	7	8	A.GND
A/D 14	9	10	A.GND
A/D 15	11	12	A.GND
D/A 1	13	14	A.GND
D/A 2	15	16	A.GND
V.REF1	17	18	A.GND
V.REF2	19	20	A.GND

Fig 3.6: Connector Pin Diagrams.

3.6 REGISTER STRUCTURE AND FORMAT

The PCL-812 requires 16 consecutive addresses in I/O space. The most important issue in programming the PCL-812 is understanding the meaning of the 16 registers addressable from the selected I/O port base address. A summary map of the function of each address and the data format of each register are given in the following sections.

3.6.1 I/0 PORT ADDRESS MAP

The following table shows the location of each register and driver relative to the base address and its usage.

Location	Read	Write
Base + 0	Counter 0	Counter 0
Base + 1	Counter 1	Counter 1
Base + 2	Counter 2	Counter 2
Base + 3	N/U	Counter control
Base + 4	A/D low byte	CH1 D/A Low byte
Base + 5	A/D high byte	CH1 D/A high byte
Base + 6	D/I low byte	CH2 D/A low byte
Base + 7	D/I high byte	CH2 D/A high byte

Base + 8	N/U	Clear interrupt request
Base + 9	N/U	N/U
Base + 10	N/U	MUX scan channel
Base + 11	N/U	PCL-812 Control
Base + 12	N/U	Software A/D trigger
Base + 13	N/U	D/O low byte
Base + 14	N/U	D/O high byte
Base + 15	N/U	

- N/U = Not Used

3.6.2 A/D Data Registers:

The A/D data registers use address BASE +4 and BASE +5.

Data Format :

1. A/D Low byte and channel number.

BASE +4	D7	D6	D5	D4	D3	D2	D1	D0
	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

2. A/D High byte.

BASE +5	D7	D6	D5	D4	D3	D2	D1	D0
	0	0	0	DRDY	AD11	AD10	AD9	AD8

Legend :

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

DRDY - Data ready signal.

3.6.3 D/A Output Registers:

The D/A output registers are write registers using address BASE +4, +5, +6 and +7.

Data Format :

BASE +4	D7	D6	D5	D4	D3	D2	D1	D0
D/A #1 low byte	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0
BASE +5	D7	D6	D5	D4	D3	D2	D1	D0
D/A #1 high byte	X	X	X	X	DA11	DA10	DA9	DA8
BASE +6	D7	D6	D5	D4	D3	D2	D1	D0
D/A #2 low byte	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0
BASE +7	D7	D6	D5	D4	D3	D2	D1	D0
D/A #2 high byte	X	X	X	X	DA11	DA10	DA9	DA8

Legend :

DA11 to DA0 - Digital to analog data. DA0 is the least significant bit (LSB) and DA11 is the most significant bit (MSB) of the data.

3.7 PROGRAMMABLE INTERVAL TIMER/COUNTER

3.7.1The intel 8253

The PCL - 812 uses the INTEL 8253 programmable interval timer/counter version 5. The 8253 is a very popular timer/counter device consisting of three independent 16 - bit down counters. Each counter has a clock input control gate and an output. It can be programmed to have a count from 2 to 65535.

The maximum clock input frequency is 5 MHZ for version 5 of 8253. The PCL-812 provides a 2 MHZ input frequency through an on - board crystal oscillator.

Counter1 and counter2 are cascaded and operated in fixed divider configuration. The counter 2 input is connected to the 2 MHZ frequency and the output of the counter2 is connected to the input of the counter1. The output of the counter 1 is internally configured to provide a trigger pulse to the A/D converter.

Counter 0 is not reserved by the PCL - 812 for any internal use, and the user may access counter 0 through connector 5. Please refer to section 3.5 for details of the connector 5 pin assignment.

3.7.2 Counter Read/Write and Control Registers:

The 8253 programmable interval counter uses 4 Registers at address BASE +0,1,2and 3. The function of each register is :

BASE +0	Counter 0 Read/Write
BASE +1	Counter 1 Read/Write
BASE +2	Counter 2 Read/Write
BASE +3	Counter Control Word

Since the 8253 counter uses a 16 - bit structure, each read/write data is split into the least significant byte and the most significant byte.

The data format of the control register is :

Base + 3	D7	D6	D5	D4	D3	D2	D1	D0
	SC1	SC0	RW1	RW0	M2	M1	M0	BCD

Legend :

SC1 & SC0 - Select counter.

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	illegal

RW2 & RW0 -- Select the read/Write operation

RW1	RW0	Operation
0	0	Counter Latch
0	1	Read/Write LSB
1	0	Read/Write MSB
1	1	Read/Write LSB first then MSB

M2,M1 and M0: Select the operation Mode.

M2	M1	M0	Mode
0	0	0	0 - Interrupt on terminal count
0	0	1	1 - Programmable one shot
X	1	0	2 - Rate generator
X	1	1	3 - Square wave rate generator
1	0	0	4 - Software trigger strobe
1	0	1	5 - Hardware trigger strobe

BCD - Select Binary or BCD counting

BCD : 1

Binary : 0

3.7.3COUNTER OPERATION

Read/Write Operation:

The type of read/write operation, operating mode and counter type all must be properly specified in the control byte and the control byte must be written before the initial count is written.

There are three types of counter operation:

1) Read/Write LSB
2) Read/Write MSB
3) Read/Write LSB followed by MSB

Counter Latch operation:

The control word corresponding to the latching operation is written. That is, RW1 and RW0 are made 00. Then, the latched value is read using any of the above 3 methods.

PROJECT MODULES

4.1 FAULT GENERATION :

In this module the user can select the required sequence of faults in the yarn using the software. Using the DAC present in the Add-on card a sequence of analog voltages corresponding to the faults on a particular scale of representation are generated.

Initially the sequence of fault is stored in an array. To each element of the array i.e.,for every individual fault its corresponding value of %error and length is assigned as per the software. This value of %error is converted to an equivalent 12 bit digital value using the formula

Digit = (error/675) * 4096. ---------- Formula 1

Time period = 2 * length * t5mm ---------- Formula 2

t5mm= 0.3/ speed

t5mm : Time duration between two 5mm pulses.

5 mm pulses: A pulse after each 5mm of yarn.

Here 675 is the range of the % error and 2^12 = 4096.

This 12 bit number is split into an 8 bit number(byte) and a 4 bit number (nibble) and loaded into the appropriate registers. i.e., byte is loaded in Base+4 register and nibble in Base + 5 register. The converted analog voltage is maintained for a time period corresponding length of the fault.

Software : **Simulate.C**

Hardware: DAC 1 of the PCL-812 add-on card.

Range of the output voltage: 0 -5 V.

4.1.1 ALGORITHM:

1) Get the fault sequence from the user and store it in an array.
2) Find the error and length for each fault in the array.
3) Determine the equivalent digital value of the error using Formula 1 and apply to the DAC1.

4) Determine the equivalent time period of the fault corresponding to its length by the formula 2.
5) Maintain the voltage from the DAC1 for that time period
6) Repeat the sequence for all the faults in the array.

4.2 FAULT IDENTIFICATION :

The generated analog signal is applied to the ADC of the Q-Cone. The %error and length of the individual fault are calculated by the following procedure. Whenever the incoming digital value exceeds or falls behind the range of digital values corresponding to the no fault criterion the counter is loaded with its maximum count. The counter is latched at that instance when the digital value falls in the range of values corresponding to the no fault criterion. The length of the fault is calculated based on the value of the counter and the error % is recovered from the digital value of the ADC.

Initially, the ADC channel is selected by setting the corresponding bit in the Base + 10 register . A start of conversion pulse (SOC) is issued by loading 0x01 to the Base + 10 register. The Base + 5 register is polled and checked for the End of conversion (EOC). The EOC is detected when the D4 bit (Data Ready) of the register is set. The converted value available in the two registers, Base+4 (LSB byte) and Base+5 (MSB nibble) are combined to get a 12 bit digital value. The equivalent error value is calculated from the digit by the formula

Error = ((digit - 2048)/2048)* 675. ----------

Formula 3

Length= time period / (2 * t5mm)

t5mm= 0.3/speed ----------

Formula4

Software : **Identify.C**

Hardware : ADC of the PCL-812 add-on card.

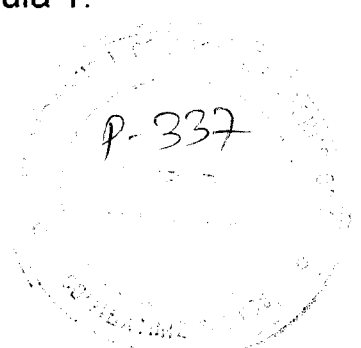
Input voltage range of ADC : -5V to +5V

Used voltage range : 0V to +5V.

ADC trigger mode : Software trigger mode.

4.2.1 ALGORITHM:

1) The ADC channel is selected .
2) The start of conversion pulse is issued.
3) The Base + 5 register is polled and checked for the EOC .
4) The binary ADC value is converted into an equivalent integer value.
5) If the value deviates from the 'no fault' range, the counter is loaded.
6) When the value falls within the 'no fault' range, the counter is latched.
7) The maximum of the integer value is used in the calculation of the % error using the formula 1.



8) Time period of the fault is calculated from the latched value of the counter .

9) The corresponding length of the fault is calculated from the time period by the formula 4.

4.3 FAULT CLASSIFICATION AND ENUMERATION:

The value of length and error information obtained from the previous module are used for classification of faults. Each particular set of length and error information corresponds to a fault of a unique class. Once the error information and length are known, a selection logic is used and the fault class is identified.

Once the class of the fault has been identified the array element corresponding to that particular fault class is incremented. At the end of the run, the total number of faults can be analysed. *Software* : **Classify.C**

4.3.1 ALGORITHM:

1) The % error and length are got from the previous module.
2) The fault category is identified based on the selection logic.
3) The array element corresponding to the fault category is incremented whenever that fault category is detected.
4) The array containing the enumerated fault information is obtained.

4.4 SERIAL COMMUNICATION AND RESULT VALIDATION

This module is used to establish a communication between the 'Q-CONE ' and the 'Testing System', to check the validity of the results of the Q-CONE.

The array containing the enumerated fault information is sent to the test system via the serial port of both the PC's.

* More information on serial communication is furnished in the appendix.

The received array values are then checked with the already available enumerated fault information array of the simulated yarn. Comparisons is made to check any undetected and/or wrong detection of faults. The wrong detections, if any, are then indicated in the Test System in a user friendly manner for evaluating the performance of the Q-Cone.

For the calculation of fault length in the Q-Cone, the information on the speed of the yarn is required to be available at the Q-Cone. Hence, initially before the simulation of yarn, the speed information is transmitted from the test system to the Q-Cone through serial communication.

Software : **Comm.C , Asyn_comm.C.**

4.4.1 ALGORITHM:

1) Transmit the speed information from test system to Q-Cone.
2) Transmit the enumerated fault from Q-Cone to test system.
3) Compare the enumerated values of the Q-Cone with that of the test system.
4) Display the results of the comparison made.

Note:

Serial communication is established by the transmission of characters between the system. Hence, the integer values that are to be transmitted from one terminal to the other is converted to its equivalent character format by a conversion algorithm.

For example, the integer value 24 is transmitted as character '2' followed by character '4'.

```
/*****************************************************************/
/*                                                               */
/*                        Main Module                            */
/*    This module calls the other modules viz., Fault simulation,*/
/*       Serial communication, Result Validation etc.            */
/*****************************************************************/


#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <alloc.h>
#include <graphics.h>

/*  files containing  the subroutines for the various modules*/
/*  ---------------------------------------------------------- */

#include "comm.c"
#include "grapic.c"
#include "speedcom.c"
#include "compare.c"
#include "simulate.c"
#include "introduc.c"


int settings= (0x80|0x08|0x00|0x03); /* setting of bioscom parameters*/
int com= 1;                          /* comm. port  selection */

/* Initialization of the fault count(no.of faults)in the array */

int arr[24]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};


/***** Main Module  *****/

main()
{

int *getarr;

/***** Allocation of memory for the  integer array ******/

getarr=malloc(50 * sizeof(int));


intro();        /***** User interface screen *****/

simulate();          /***** Fault simulation module ****/
```

```
getarr=receive();  /* Receiving enumerated fault info. from Qcone */

printf("\n\n\n\t Press <Enter> to view graphical output!");
getch();

userint(getarr);   /* Graphical Display of received fault info. */


}   /*****  end of main module     *****/
```

```
                /*****  Comm.C  *****/

/*********************************************************************/
/*                                                                   */
/*             Serial  Communication   Module                        */
/*       Receives the enumerated Fault info. from the Q-Cone         */
/*********************************************************************/



int *receive(void)
{


extern int settings;          /*****  These are external ****/
extern int com;               /***** variables defined in main module*/



char get;
char fa[4];
int fcount[24];
int i=0,j=0;
int *rec;          /*** stores received enumerated fault info.**/


bioscom(0,settings,com); /**** Initialise settings in the comm.port*/


                           /*** receive input from comm. port
do                        until data is ready.
{                               ( till the end of receiving
bioscom(1,' ',com);             the  char 'x'  )
get=bioscom(2,0,com);                                    **/
}while(get!='x');


                           /****  receive enumerated fault
do                              info. from the Q-Cone and
{                               store in an array
bioscom(1,' ',com);                                      */
fa[i]=bioscom(2,0,com);
get=fa[i];

if(i==3)
{
i=0;
/******  converting received character info. into integer values*/
rec[j]=((fa[0]-48)*1000)+((fa[1]-48)*100)+
((fa[2]-48)*10)+(fa[3]-48);


j++;
}

else i++;
```

```
}while(get!='?');


return(rec);

}
```

```
                /****  Grapic.C  ****/
/*************************************************************/
/*                                                           */
/*          Graphical Display of enumerated fault info.      */
/*                received from the Q-cone                    */
/*                                                           */
/*************************************************************/


#define N 24   /****  24 different fault categories  ***/

void userint(int *arr)
{
int gd,gm;
int i,j,k,p,pp;
int c,r;
int xorig,yorig;
int *bet;
int key=0;
int xmax=640,ymax=480;
char flt[N][3];
char *stri;

clrscr();



fflush(stdin);



for(i=0;i<N;i++)                    /***  To convert the    ***/
{                                   /***  enumerated fault info.*/
flt[i][0]=(arr[i] /10) + 48;          /*into a character string**/
flt[i][1]=(arr[i] %10) + 48;
flt[i][2]='\0';

}

clrscr();


detectgraph(&gd,&gm);       /** Initialising the default */
initgraph(&gd,&gm,"");      /** graphic driver  ***/


setcolor(RED);
setbkcolor(WHITE);
xorig=50;
yorig=465;

setcolor(RED);
line(xorig,yorig-116,562,yorig-116);
```

```
setcolor(7);
□
for(i=1;i<6;i++)
□
line(xorig+80+(40*i),yorig-80,xorig+80+(40*i),yorig-116);
for(i=1;i<6;i++)
line(xorig+80,yorig-80-(6*i),xorig+512,yorig-80-(6*i));
setcolor(12);
rectangle(xorig+80, yorig-116,xorig+320,yorig-80);
line(xorig+319,yorig-116,xorig+319,yorig-80);


setcolor(7);
for(i=1;i<6;i++)
line(xorig+320+(32*i),yorig-80,xorig+320+(32*i),yorig-116);

setcolor(RED);
rectangle(xorig+320, yorig-116,xorig+512,yorig-80);


setcolor(1);
rectangle(xorig,yorig-450,xorig+80,yorig-116);
line(xorig+40,yorig-450,xorig+40,yorig-116);
line(xorig+20,yorig-450,xorig+20,yorig-116);

line(xorig+10,yorig-450,xorig+10,yorig-116);

line(xorig,yorig-316,xorig+80,yorig-316);

line(xorig,yorig-216,xorig+80,yorig-216);

line(xorig,yorig-148,xorig+80,yorig-148);
setcolor(7);
for(i=1;i<=4;i++)
{
line(xorig+(2*i),yorig-450,xorig+(2*i),yorig-116);
}
for(i=1;i<=4;i++)
{
line(xorig+10+(2*i),yorig-450,xorig+10+(2*i),yorig-116);
}

for(i=1;i<=4;i++)
{
line(xorig+20+(4*i),yorig-450,xorig+20+(4*i),yorig-116);
}
for(i=1;i<=4;i++)
{
line(xorig+40+(8*i),yorig-450,xorig+40+(8*i),yorig-116);
}


for(i=1;i<=3;i++)
{
line(xorig,yorig-116-(8*i),xorig+80,yorig-116-(8*i));
}
```

```
for(i=1;i<17;i++)
{
line(xorig,yorig-148-(4*i),xorig+80,yorig-148-(4*i));
}
for(i=1;i<10;i++)
{
line(xorig,yorig-216-(10*i),xorig+80,yorig-216-(10*i));
}
for(i=1;i<17;i++)
{
line(xorig,yorig-316-(8*i),xorig+80,yorig-316-(8*i));
}


setcolor(7);
for(i=1;i<42;i++)
{
line(xorig+80,yorig-116-(8*i),xorig+512,yorig-116-(8*i));
}
for(i=1;i<12;i++)
{
line(xorig+80+(36*i),yorig-450,xorig+80+(36*i),yorig-116);
}

setcolor(12);
line(xorig+80,yorig-450,xorig+512,yorig-450);
line(xorig+512,yorig-450,xorig+512,yorig-116);
line(xorig+80,yorig-117,xorig+512,yorig-117);
line(xorig+79,yorig-450,xorig+79,yorig-117);


setcolor(7);
for(i=1;i<5;i++)
line(xorig+80,yorig-20-(2*i),xorig+512,yorig-20-(2*i));

for(i=1;i<5;i++)
line(xorig+80,yorig-(4*i),xorig+512,yorig-(4*i));

for(i=1;i<6;i++)
line(xorig+80+(40*i),yorig-30,xorig+80+(40*i),yorig);

for(i=1;i<6;i++)
line(xorig+320+(32*i),yorig-30,xorig+320+(32*i),yorig);
setcolor(5);

rectangle(xorig+80,yorig-30,xorig+320,yorig);
rectangle(xorig+320,yorig-30,xorig+512,yorig);
line(xorig+80,yorig-20,xorig+512,yorig-20);
line(xorig+320,yorig-30,xorig+320,yorig);

setcolor(9);
line(50,5,50,465);
line(50,465,580,465);


outtextxy(xorig-50,yorig-425,"%ERROR");
```

```
outtextxy(xorig-50,yorig-423,"_____");

outtextxy(xorig+415,yorig+3,"LENGTH (CM)");
outtextxy(xorig+415,yorig+5,"_____");


outtextxy(xorig-50,yorig-116,"100%");
outtextxy(xorig-50,yorig-148,"150%");
outtextxy(xorig-50,yorig-216,"250%");
outtextxy(xorig-50,yorig-316,"400%");
outtextxy(xorig-50,yorig-450,"600%");
outtextxy(xorig-50,yorig-80,"45%");

outtextxy(xorig-50,yorig-30,"-30%");
outtextxy(xorig-50,yorig-20,"-45%");
outtextxy(xorig-50,yorig,"-75%");

outtextxy(xorig-10,yorig+3,".1");
outtextxy(xorig+10,yorig+3,"1");
outtextxy(xorig+20,yorig+3,"2");
outtextxy(xorig+40,yorig+3,"4");
outtextxy(xorig+78,yorig+3,"8");
outtextxy(xorig+318,yorig+3,"32");
outtextxy(xorig+510,yorig+3,"51.2");

setcolor(12);
settextstyle(SANS_SERIF_FONT,HORIZ_DIR,4);
setcolor(RED);
outtextxy(xorig+250,yorig-308," Q ");          /***  To display the ***/
setcolor(BLUE);
outtextxy(xorig+250,yorig-298,flt[16]);        /*** enumerated fault ***/

setcolor(RED);
outtextxy(xorig+200,yorig-72," Z ");           /*** information at the ***/
                                   /*** corresponding area ***/
settextstyle(SANS_SERIF_FONT,HORIZ_DIR,2);
setcolor(RED);
***/                                           /*** of the fault category
outtextxy(xorig+200,yorig-108," R ");
setcolor(BLUE);
outtextxy(xorig+200,yorig-98,flt[17]);
setcolor(RED);
outtextxy(xorig+410,yorig-108," S ");
setcolor(BLUE);
outtextxy(xorig+410,yorig-98,flt[18]);
setcolor(RED);
outtextxy(xorig+200,yorig-18," U ");
setcolor(BLUE);
outtextxy(xorig+230,yorig-18,flt[20]);
setcolor(RED);
outtextxy(xorig+410,yorig-18," W ");
setcolor(BLUE);
outtextxy(xorig+435,yorig-18,flt[22]);

settextstyle(SMALL_FONT,HORIZ_DIR,5);
setcolor(RED);
```

```
outtextxy(xorig+200,yorig-29," T ");
setcolor(BLUE);
outtextxy(xorig+225,yorig-29,flt[19]);
setcolor(RED);
outtextxy(xorig+410,yorig-29," V ");
setcolor(BLUE);
outtextxy(xorig+430,yorig-29,flt[21]);

setcolor(RED);
outtextxy(xorig+3,yorig-140,"A");
setcolor(BLUE);
outtextxy(xorig-7,yorig-130,flt[0]);

setcolor(RED);
outtextxy(xorig+3,yorig-190,"B");
setcolor(BLUE);
outtextxy(xorig-7,yorig-180,flt[1]);

setcolor(RED);
outtextxy(xorig+3,yorig-280,"C");
setcolor(BLUE);
outtextxy(xorig-7,yorig-270,flt[2]);

setcolor(RED);
outtextxy(xorig+3,yorig-390,"D");
setcolor(BLUE);
outtextxy(xorig-7,yorig-380,flt[3]);

setcolor(BLUE);
outtextxy(xorig+14,yorig-140,"E");
setcolor(RED);
outtextxy(xorig+10,yorig-130,flt[4]);

setcolor(BLUE);
outtextxy(xorig+14,yorig-190,"F");
setcolor(RED);
outtextxy(xorig+10,yorig-180,flt[5]);

setcolor(BLUE);
outtextxy(xorig+14,yorig-280,"G");
setcolor(RED);
outtextxy(xorig+10,yorig-270,flt[6]);

setcolor(BLUE);
outtextxy(xorig+14,yorig-390,"H");
setcolor(RED);
outtextxy(xorig+10,yorig-380,flt[7]);
setcolor(RED);
outtextxy(xorig+30,yorig-140,"I");
setcolor(BLUE);
outtextxy(xorig+25,yorig-130,flt[8]);

setcolor(RED);
outtextxy(xorig+30,yorig-190,"J");
setcolor(BLUE);
outtextxy(xorig+25,yorig-180,flt[9]);
setcolor(RED);
```

```
outtextxy(xorig+30,yorig-280,"K");
setcolor(BLUE);
outtextxy(xorig+25,yorig-270,flt[10]);

setcolor(RED);
outtextxy(xorig+30,yorig-390,"L");
setcolor(BLUE);
outtextxy(xorig+25,yorig-380,flt[11]);

setcolor(BLUE);
outtextxy(xorig+53,yorig-140,"M");
setcolor(RED);
outtextxy(xorig+48,yorig-130,flt[12]);
setcolor(BLUE);
outtextxy(xorig+53,yorig-190,"N");
setcolor(RED);
outtextxy(xorig+48,yorig-180,flt[13]);
setcolor(BLUE);
outtextxy(xorig+53,yorig-280,"O");
setcolor(RED);
outtextxy(xorig+48,yorig-270,flt[14]);
setcolor(BLUE);
outtextxy(xorig+53,yorig-390,"P");
setcolor(RED);
outtextxy(xorig+48,yorig-380,flt[15]);

setcolor(RED);
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(190,40,"Fault Distribution Data");


getch();


}
```

```
                /* Speed_Com.C ***/

/*********************************************************/
/*                                                       */
/*          Sends speed info. to Q-Cone                  */
/*********************************************************/


void send_speed(int speed)
{
extern int settings;
extern int com;
int i,j;
char sp[4];
int dum;

sp[3]=48+speed%10;speed/=10;/* converting integer value */
sp[2]=48+speed%10;speed/=10;/* into characters          */
sp[1]=48+speed%10;speed/=10;
sp[0]=48+speed;



bioscom(0,settings,com);     /* Initialising comm. port */


bioscom(1,'x',com);      /* sending start of data */

for(i=0;i<4;i++)
{
bioscom(1,sp[i],com);    /* sending spped info. */
}


bioscom(1,'?',com);      /* sending end of data */


}
```

```
                    /*** Compare .C ***/
/*****************************************************************/
/*                    Comparison module                          */
/*****************************************************************/



void compare(int *getarr)
{

int i;
extern int arr[24];                    /*** arr[] : sent faults ***/

printf(" \n\n Received faults :  ");

for(i=0;i<23;i++)
printf(" %d",getarr[i]);              /*** getarr[]: received faults */

for(i=0;i<23;i++)
{
if(arr[i] != getarr[i])
 {
 printf("\n Error in %c",(i+65));
 if(arr[i]>getarr[i])printf("(%d faults undetected)",arr[i]-getarr[i]);
 else printf(" ( %d faults excess )",getarr[i]-arr[i]);
 }
 }



}
```

```
            /****  Simulate.C  ****/

/*************************************************************/
/*              Yarn Fault Generation (simulation)           */
/*************************************************************/


void simulate(void)
{


extern int arr[24];

char categ[100];
int *getarr;
int speed;
int ret[2];
float t5mm;
float tfault;
float decimal;
float tot_len=0;
int volt,k=0;
float length;
int var;
int tt5mm;
float error;
float rad;
float del;
int div,i;
int no=0;
int trun;
int coun=0;
int byte,temp,nibble,j,rem;
int byte1,byte2,nibble1,nibble2;
float equi;

textcolor(7);
textbackground(RED);

printf("\nEnter the speed in mpm  :");
scanf("%d",&speed);

printf("\nEnter the category   :");
scanf("%s",categ);


send_speed(speed);/* Speed info. is sent to Q-Cone through*/
                /**  serial communications              */



 getarr=malloc(50 * sizeof(int));
```

```
outportb(0x224,0x04); /*  Initially 'no fault' is     */
□
outportb(0x225,0x02); /** is sent ( voltage corresponding */
delay(310);            /* to Z category is */
delay(200);

for(i=0;i<strlen(categ);i++)/*for loop run to access each*/
{ if (categ[i]>='a' && categ[i]<='z') /*of the fault type */
categ[i]=toupper(categ[i]); /*** in the array - categ   ***/
switch(categ[i])
{
case 'A':
{
      error=210.0; /* error and length values in A category*/
      length=0.5; /*is assigned              */
      arr[0]++; /* first element of array (arr) is  */
                /*  is incremented ( A type fault ) */
}
break;

case 'B':
{
      error=275.0;
      length=0.5;
      arr[1]++;
      }
      break;
case 'C':
{
      error=400.0;
      length=0.5 ;
      arr[2]++;
      }break;


case 'D':
      {
      error=575.0;
      length=0.5;
      arr[3]++;
      }break;
case 'E':
      {
      error=210.0;
      length=1.5;
      arr[4]++;
      }break;

case 'F':
      {
      error=275.0;
      length=1.5;
      arr[5]++;
      }break;

case 'G':
```

```
      {
      error=400.0;
      length=1.5;
      arr[6]++;
      }break;
case 'H':
      {
      error=575.0;
      length=1.5;
      arr[7]++;
      }break;

case 'I':
      {
      error=210.0;
      length=3.0;
      arr[8]++;
      }break;

case 'J':
      {
      error=285.0;
      length=3.0;
      arr[9]++;
      }break;

case 'K':
      {
      error=400.0;
      length=3.0;
      arr[10]++;
      }break;

case 'L':
      {
      error=575.0;
      length=3.0;
      arr[11]++;
      }break;
case 'M':
      {
      error=210.0;
      length=6.0;
      arr[12]++;
      }break;
case 'N':
      {
      error=275.0;
      length=5.6;
      arr[13]++;
      }break;
case 'O':
      {
      error=400.0;
      length=5.6;
      arr[14]++;
      }break;
```

```
  case 'P':
        {
        error=575.0;
        length=6.0;
        arr[15]++;
        }break;

  case 'Q':
        {
        error=450.0;
        length=30.0;
        arr[16]++;
        }break;

  case 'R':
        {
        error=156.0;
        length=20.0;
        arr[17]++;
        }break;
 case 'S':
        {
        error=156.0;
        length=40.0;
        arr[18]++;
        }break;
 case 'T':
 {
        error=37.0;
        length=20.0;
        arr[19]++;
 }break;
 case 'U':
        {
        error=15.0;
        length=20.0;
        arr[20]++;
        }break;
 case 'V':
        {
        error=37.0;
        length=40.0;
        arr[21]++;
        }break;
case 'W':
{
      error=15.0;
      length=40.0;
      arr[22]++;
}break;


case 'Z': { error=85.0;
         length= 2.0;
          break;
          }
```

```
}

tot_len+=length;          /*  Total length of yarn run  */

decimal=(error/675.0)*4095.0;/*decimal = equivalent voltage*/

volt=decimal;      /*  to convert 'decimal'(integer) to */
if(decimal-volt>=0.5)volt++;   /* proper float value*/

t5mm=0.3/speed;    /*t5mm : time between every 5mm of yarn */
tfault=2*length*t5mm; /* tfault : time period of the fault*/
del=tfault;       /* hence, del= time corresponding to the */
                    /*** length of that fault */



temp=volt;byte=nibble=0;

for(j=0;j<8;j++)            /** bits seperation logic **/
{                          /** error info. in 12 bits **/
rem=temp%2;                /** is split into 8 bit   **/
byte+=rem*pow(2,j);        /** LSB ( byte ) and 4 bit **/
temp=temp/2;               /** MSB (nibble)           **/
}

for(j=0;j<4;j++)
{
rem=temp%2;
nibble+=rem*pow(2,j);
temp=temp/2;
}


outport(0x224,byte); /*The byte and nibble representing */
outport(0x225,nibble); /* error of the fault is output in*/
delay(del);            /* appropriate base registers    */


outportb(0x224,0x04);    /** 'No Fault' is sent between **/
outportb(0x225,0x02);    /** each fault ( Z category ) **/
delay(60);


}/* END OF FOR LOOP*/


outportb(0x224,0x00); /**  No yarn condition      **/
outportb(0x225,0x00);    /** To indicate end of yarn **/
```

```
printf("\n\n Total length of the yarn: %f",tot_len);


printf(" \n\n Fault category  :  ");

for(i=0;i<23;i++)
printf(" %c",(i+65));

printf(" \n\n Faults sent:  ");/* Enumerated info. of */
                               /*simulated faults    */
for(i=0;i<23;i++)                  /*** printed   */
printf(" %d",arr[i]);


}
```

```
                    /*** Intro.C  ***/

/*************************************************************/
/*        Initial screen showing fault classifications       */
/*************************************************************/


#include <conio.h>
#include <stdio.h>
#include <graphics.h>



void intro(void)
{
int gd,gm;
int i,j,k,p,pp;
int c,r;
int xorig,yorig;
int key=0;
int xmax=640,ymax=480;

clrscr();

detectgraph(&gd,&gm);  /*** initialising the default  ***/
initgraph(&gd,&gm,""); /*** graphics  mode            ***/


setcolor(RED);
setbkcolor(WHITE);
xorig=50;
yorig=465;


setcolor(RED);
line(xorig,yorig-116,562,yorig-116);


setcolor(7);
for(i=1;i<6;i++)
line(xorig+80+(40*i),yorig-80,xorig+80+(40*i),yorig-116);
for(i=1;i<6;i++)
line(xorig+80,yorig-80-(6*i),xorig+512,yorig-80-(6*i));
setcolor(12);
rectangle(xorig+80, yorig-116,xorig+320,yorig-80);
line(xorig+319,yorig-116,xorig+319,yorig-80);


setcolor(7);
for(i=1;i<6;i++)
line(xorig+320+(32*i),yorig-80,xorig+320+(32*i),yorig-116);

setcolor(RED);
rectangle(xorig+320, yorig-116,xorig+512,yorig-80);

setcolor(1);
```

```
rectangle(xorig,yorig-450,xorig+80,yorig-116);
line(xorig+40,yorig-450,xorig+40,yorig-116);
line(xorig+20,yorig-450,xorig+20,yorig-116);

line(xorig+10,yorig-450,xorig+10,yorig-116);

line(xorig,yorig-316,xorig+80,yorig-316);

line(xorig,yorig-216,xorig+80,yorig-216);

line(xorig,yorig-148,xorig+80,yorig-148);
setcolor(7);
for(i=1;i<=4;i++)
{
line(xorig+(2*i),yorig-450,xorig+(2*i),yorig-116);
}
for(i=1;i<=4;i++)
{
line(xorig+10+(2*i),yorig-450,xorig+10+(2*i),yorig-116);
}

for(i=1;i<=4;i++)
{
line(xorig+20+(4*i),yorig-450,xorig+20+(4*i),yorig-116);
}
for(i=1;i<=4;i++)
{
line(xorig+40+(8*i),yorig-450,xorig+40+(8*i),yorig-116);
}



for(i=1;i<=3;i++)
{
line(xorig,yorig-116-(8*i),xorig+80,yorig-116-(8*i));
}
for(i=1;i<17;i++)
{
line(xorig,yorig-148-(4*i),xorig+80,yorig-148-(4*i));
}
for(i=1;i<10;i++)
{
line(xorig,yorig-216-(10*i),xorig+80,yorig-216-(10*i));
}
for(i=1;i<17;i++)
{
line(xorig,yorig-316-(8*i),xorig+80,yorig-316-(8*i));
}



setcolor(7);
for(i=1;i<42;i++)
{
```

```
line(xorig+80,yorig-116-(8*i),xorig+512,yorig-116-(8*i));
}
for(i=1;i<12;i++)
{
line(xorig+80+(36*i),yorig-450,xorig+80+(36*i),yorig-116);
}

setcolor(12);
line(xorig+80,yorig-450,xorig+512,yorig-450);
line(xorig+512,yorig-450,xorig+512,yorig-116);
line(xorig+80,yorig-117,xorig+512,yorig-117);
line(xorig+79,yorig-450,xorig+79,yorig-117);


setcolor(7);
for(i=1;i<5;i++)
line(xorig+80,yorig-20-(2*i),xorig+512,yorig-20-(2*i));

for(i=1;i<5;i++)
line(xorig+80,yorig-(4*i),xorig+512,yorig-(4*i));

for(i=1;i<6;i++)
line(xorig+80+(40*i),yorig-30,xorig+80+(40*i),yorig);

for(i=1;i<6;i++)
line(xorig+320+(32*i),yorig-30,xorig+320+(32*i),yorig);
setcolor(5);

rectangle(xorig+80,yorig-30,xorig+320,yorig);
rectangle(xorig+320,yorig-30,xorig+512,yorig);
line(xorig+80,yorig-20,xorig+512,yorig-20);
line(xorig+320,yorig-30,xorig+320,yorig);

setcolor(9);
line(50,5,50,465);
line(50,465,580,465);


outtextxy(xorig-50,yorig-425,"%ERROR");
outtextxy(xorig-50,yorig-423,"______");

outtextxy(xorig+415,yorig+3,"LENGTH (CM)");
outtextxy(xorig+415,yorig+5,"___________");


outtextxy(xorig-50,yorig-116,"100%");
outtextxy(xorig-50,yorig-148,"150%");
outtextxy(xorig-50,yorig-216,"250%");
outtextxy(xorig-50,yorig-316,"400%");
outtextxy(xorig-50,yorig-450,"600%");
outtextxy(xorig-50,yorig-80,"45%");
```

```
outtextxy(xorig-50,yorig-30,"-30%");
outtextxy(xorig-50,yorig-20,"-45%");
outtextxy(xorig-50,yorig,"-75%");

outtextxy(xorig-10,yorig+3,".1");
outtextxy(xorig+10,yorig+3,"1");
outtextxy(xorig+20,yorig+3,"2");
outtextxy(xorig+40,yorig+3,"4");
outtextxy(xorig+78,yorig+3,"8");
outtextxy(xorig+318,yorig+3,"32");
outtextxy(xorig+510,yorig+3,"51.2");

setcolor(12);
settextstyle(SANS_SERIF_FONT,HORIZ_DIR,4);
outtextxy(xorig+250,yorig-308," Q ");
outtextxy(xorig+200,yorig-72," Z ");

settextstyle(SANS_SERIF_FONT,HORIZ_DIR,2);
outtextxy(xorig+200,yorig-108," R ");
outtextxy(xorig+410,yorig-108," S ");
outtextxy(xorig+200,yorig-20," U ");
outtextxy(xorig+410,yorig-20," W ");
settextstyle(SMALL_FONT,HORIZ_DIR,5);
outtextxy(xorig+200,yorig-31," T ");
outtextxy(xorig+410,yorig-31," V ");

setcolor(RED);
outtextxy(xorig+3,yorig-140,"A");
outtextxy(xorig+3,yorig-190,"B");
outtextxy(xorig+3,yorig-280,"C");
outtextxy(xorig+3,yorig-390,"D");

outtextxy(xorig+14,yorig-140,"E");
outtextxy(xorig+14,yorig-190,"F");
outtextxy(xorig+14,yorig-280,"G");
outtextxy(xorig+14,yorig-390,"H");

outtextxy(xorig+30,yorig-140,"I");
outtextxy(xorig+30,yorig-190,"J");
outtextxy(xorig+30,yorig-280,"K");
outtextxy(xorig+30,yorig-390,"L");

outtextxy(xorig+53,yorig-140,"M");
outtextxy(xorig+53,yorig-190,"N");
outtextxy(xorig+53,yorig-280,"O");
outtextxy(xorig+53,yorig-390,"P");

setcolor(RED);
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
outtextxy(200,40,"Fault Distribution Data");


getch();
```

```
clrscr();
setfillstyle(1,9);
bar(0,0,640,480);
}
```

```
                /*** Identify.c ***/

/*************************************************************************/
/*                        Fault detection module                         */
/*************************************************************************/


#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <bios.h>
#include <time.h>
#include "class1.c"
#include "asyn_com.c"
#include "speed.c"
#include "screen.c"
int settings = (0x80|0x08|0x00|0x03);
int com=1;

main()
{

int faa[25]={ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
char ge;
int ch=1;
int high;
int low;
unsigned int value;
int digit;
int flag=0;
int msb;
int lsb;
unsigned int count0;
int i=0;
int j=0;

float del;
float avalue;
float volt;
float fault[100][2];
float xxx[20];
float error;
float max;
float length;
float speed;
int xx=0;
int k=0;
int store;
int get;
int prev;
int great,plus=1,minus=1;
clrscr();


screen();                       /***  Initial screen showing ***/
                          /***  fault classification   ***/
```

```
speed=get_speed();              /*** Receives speed from test jig ***/
                          /*** through serial communication ***/


do
{
        xx++;

outportb(0x30a,ch);  /*** channel selection  **/
outportb(0x30c,0x01); /** SOC  **/

do
{

high=inportb(0x305);   /** check for EOC ***/
}while(high>=0x10);

low=inportb(0x304);
digit=low+(high<<8);

k=0;

great=digit;
plus=minus=0;

do
{
                  /**** next value  ******/

outportb(0x30a,ch);
outportb(0x30c,0x01);

do
{
high=inportb(0x305);

}while(high>=0x10);

low=inportb(0x304);
prev=digit;
digit=(high<<8)+low;


if((digit>great) && plus==1) great=digit;

if(digit<great && minus==1) great=digit;


store=abs(prev-digit);

  if( store > 20 && (digit <2300|| digit>2310  ))

      {
      if(k==0)                        /****  counter loading ****/
       {
      outportb(0x303,0x36);
```

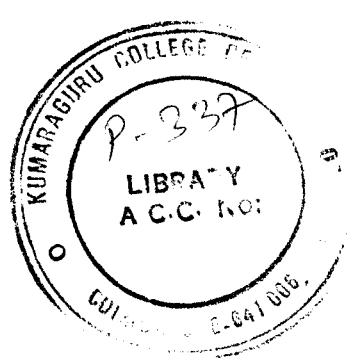
```
    outportb(0x300,0xff);
    outportb(0x300,0xff);
    k=1;

    if(prev-digit >0) minus=1;
    if(prev-digit<0) plus=1;


    }
    }

    else
    {
    if(store >20 )          /*****  counter latching *****/
    {
    outportb(0x303,0x06);
    outportb(0x303,0x36);
    low=inportb(0x300);
    high=inportb(0x300);
    count0=(high<<8) + low;
    del=(65535-count0)*0.5/1000.0;
    plus=minus=0;



    break;
     }
    }



 }while(digit>2052);

 error= ((great-2047.0)/2047.0)*675;

    length=(((del-5.0)*52.0/25.0))*speed/1040.0;

 fault[j][0]=length;
 fault[j][1]=error;

 j++;



 if(digit<2052)
   {
   gotoxy( 34,4);
   settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
   outtextxy(200,400,"Yarn break!");
   }


}while(digit>2052);
```

```
printf("\n ");
for(i=0;i<j;i++)
{

ge=check(fault[i][0],fault[i][1]);   /*** gets classified fault ***/



switch(ge)
{
case 'A' : faa[0]++;break;
case 'B' : faa[1]++;break;
case 'C' : faa[2]++;break;
case 'D' : faa[3]++;break;    /***** Enumerating the appropriate ****/
case 'E' : faa[4]++;break;    /***** fault type                   ****/
case 'F' : faa[5]++;break;
case 'G' : faa[6]++;break;
case 'H' : faa[7]++;break;
case 'I' : faa[8]++;break;
case 'J' : faa[9]++;break;
case 'K' : faa[10]++;break;
case 'L' : faa[11]++;break;
case 'M' : faa[12]++;break;
case 'N' : faa[13]++;break;
case 'O' : faa[14]++;break;
case 'P' : faa[15]++;break;
case 'Q' : faa[16]++;break;
case 'R' : faa[17]++;break;
case 'S' : faa[18]++;break;
case 'T' : faa[19]++;break;
case 'U' : faa[20]++;break;
case 'V' : faa[21]++;break;
case 'W' : faa[22]++;break;
case 'X' : faa[23]=1;break;
}

}



send(faa);  /*** Sends the enumerated fault info. to test system ***/
          /*** through serial communication ***/
screen();   /*** output screen ***/


 }
```

```
            /*****  classify.c  *****/

/*****************************************************************/
/*                                                               */
/*          Fault classification module                          */
/*****************************************************************/



#include <stdio.h>
#include <conio.h>
#include <graphics.h>

char check(float length,float error)
{

int i,j;
int c=0;

int xary[5],yary[6],comb[20];
int ret[20];
int gd,gm;
float x,y;
int p;


x=length;y=error;



if(x<8)                                          /*  Z region */
{
if(y>=2 && y<175) return('Z');
}
else
{
if(y>=45 && y<120) return('Z');
}


if(x<8)
{
if(x>=2)
{
if(x>=4)
 {

  if(y>=325)
  {
    if(y<475)
    { return('O');
    }
    else return('P');
  }
  else
  {
```

```
   if(y>=225)
   { return('N'); }
   if(y<225 && y>=175)
     {  return('M');}
 }
}


if(x<4 && x>=2)
{
 if(y>=325)
 {

   if(y>=475)
   {return('L');}
   if(y<475)
   {return('K');}
 }
 else
 {

   if(y>=225)
  { return('J'); }
   if(y<225 && y>=175)
   { return('I'); }
 }
 }
}

else
{

 if(x<2 && x>=1)
 {

  if(y>=325)
  {

    if(y>=475)
    {return('H');}
    if(y<475)
    {return('G');}
  }
  else
  {

    if(y>=225)
    { return('F'); }
    if(y<225 && y>=175)
    {return('E'); }
  }
 }


  if(x<1)
  {
```

```
 if(y>=325)
 {

   if(y>=475)
   { return('D');}
   if(y<475)
   {return('C');}
 }
 else
 {

   if(y>=225)
   { return('B');}
   if(y<225 && y>=175)
   { return('A');}
 }
}

 }

}

 else
 {
 if(y>=75)
 {

  if(y>=175)
   {
    return('Q');
   }
  else
  {

  if(y>=120)
  {
    if(x<32)
    {
    return('R');
    }
    else
    {
    return('S');
    }
  }
  }
 }

 else
 {


 if(y<=45)
 {
  if(y>=30)
  {
```



```
  if(x<32)
  {
   return('T');
  }

  else
  {
  return('V');
  }
 }

 else
 {
  if(x<32)
  {
   return('U');
   }
  else
  {
  return('W');
  }
 }
 }

}
}
```

```
/************  Asyn_comm.c *************/


#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <bios.h>
#include <stdlib.h>
#include <math.h>


void send(int *faa)
{
extern int settings;
extern int com;


int i=0,j=0;
char cc[4];


bioscom(0,settings,com);  /***  initialising serial port ***/


bioscom(1,'x',com);       /*** sending 'x' initially ***/


  for(i=0;i<24;i++)
  {
    for(j=3;j>=0;j--)                  /*** converting integer values ***/
     {                                 /*** to characters before sending **/
      cc[j]=putchar(faa[i]%10 +48);
      printf("\b ");
      faa[i]/=10;
      }

      for(j=0;j<4;j++)
      {

      bioscom(1,cc[j],com);   /*** sending character by character ***/
      }
  }

  for(i=0;i<3;i++)                /*** sending '?' character to ***/
  {                        /*** denote end of data ***/
bioscom(1,'?',com);
}


  }
```

```
                    /* Speed.c */

/*********************************************************/
/*          Receives speed info. from test system        */
/*********************************************************/


int get_speed(void)
{
extern int settings;
extern int com;
int i=0,j=0;
int speed;
int g;
char spe[4];


bioscom(0,settings,com);    /* Initialising comm. port */

do
{
g=bioscom(2,0,com);      /* Checking for start of data */
}while(g!='x');

do
{
spe[i]=bioscom(2,0,com); /* receiving speed info. */
g=spe[i];

if(i==3)
{
i=0;
speed=((spe[0]-48)*1000)
 + ((spe[1]-48)*100)          /* convert char. value */
 + ((spe[2]-48)*10)
 + spe[3]-48;                /* to integer value  */
}
else i++;
}while(g!='?');


return(speed);     /* returns speed to main module */
}
```

```
                                /**** Screen.c ****/

/*************************************************************************/
/*                    Screen displaying 'Q-Cone'                         */
/*************************************************************************/


void screen(void)
{
int gd,gm;

clrscr();

detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

setbkcolor(9);
setcolor(RED);

settextstyle(DEFAULT_FONT,HORIZ_DIR,10);
outtextxy(100,190,"Q-CONE");


getch();
}
```

CONCLUSION

A test system for evaluating the performance of the yarn quality monitor (Q-Cone) has been designed and implemented. Certain changes which were made as per the company's requirements were also incorporated into the test system. These changes make the system more general and extend its scope and range.

The system was able to test the performance of the Q-Cone for a wide variety of values corresponding to the various fault classes and at different yarn speed settings. The sequence of simulated faults and the outputs of the Q-Cone are displayed together in a user-friendly manner so that the end results can be easily analysed.

Though this system by itself is sufficient to test the performance of the Q-Cone certain improvements can be made. Provisions can be made for the calculation of certain statistical

measurements of the error like Cumulative Variance calculations. This mainly involves one more software program. This would give a more accurate information about the quality of the yarn.

The same test system can be interfaced with several Q-Cones by having the required hardware, address selection logic and the related synchronizing equipment. This will be very much useful in a textile industry, where the entire process can be controlled by a single operator or computer with greater efficiency, with multiple Q-Cone facilities.

REFERENCES

1. Joe Campbell, "The RS-232 solution", BPB Publications, New Delhi, 1992 Edition.

2. Peter Norton, "Inside IBM PC", Prentice Hall, New York,1991 Edition .

3. Mark Nelson, "Serial Communication – A C++ Developer's Guide", BPB Publications, New Delhi, 1988 Edition.

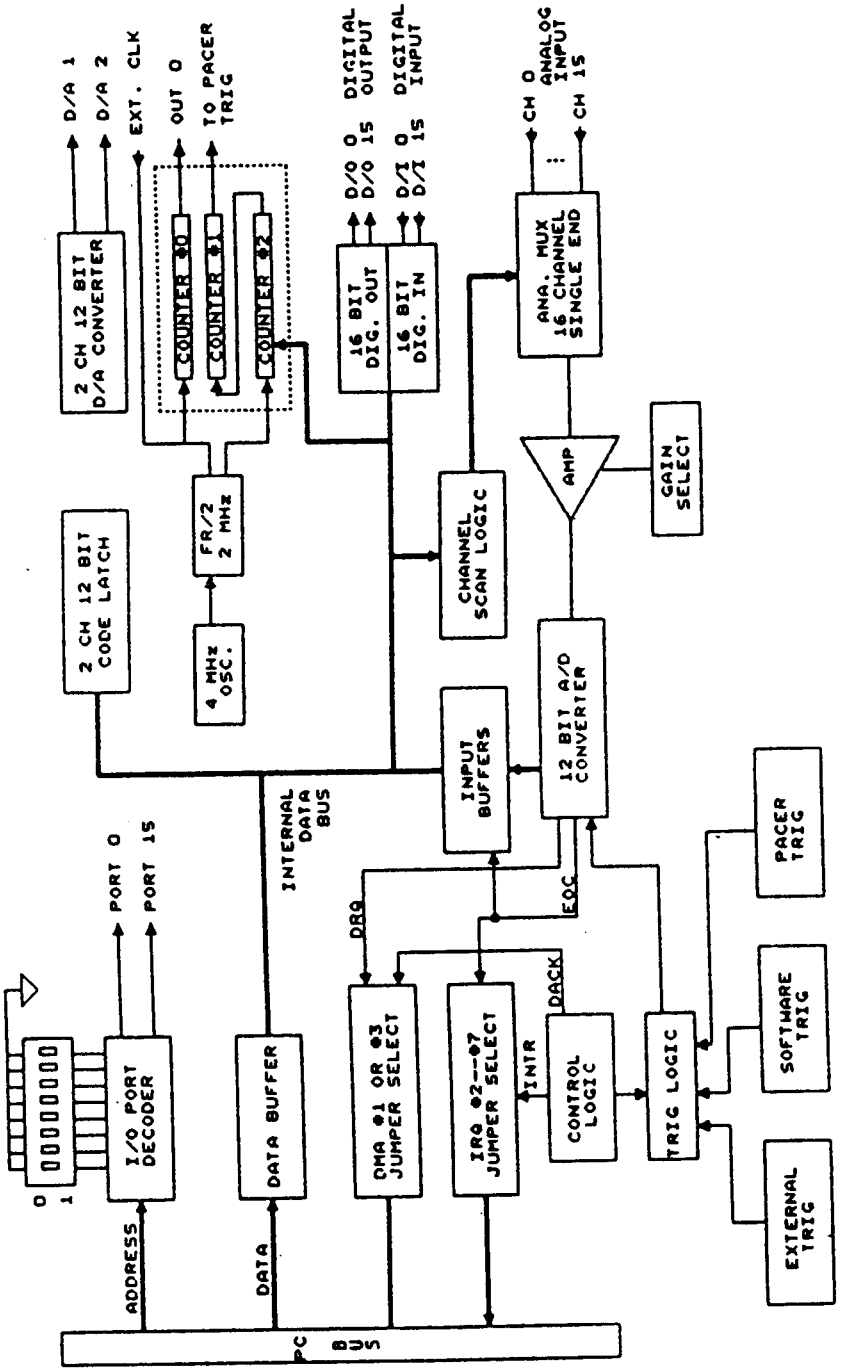
4. P.Govindarajulu, "IBM PC and Clones", TMH Publications, New Delhi, 1991 Edition.

5. Sanjay K. Bose, "Hardware and Software of PC", TMH Publications, New Delhi, 1989 Edition.

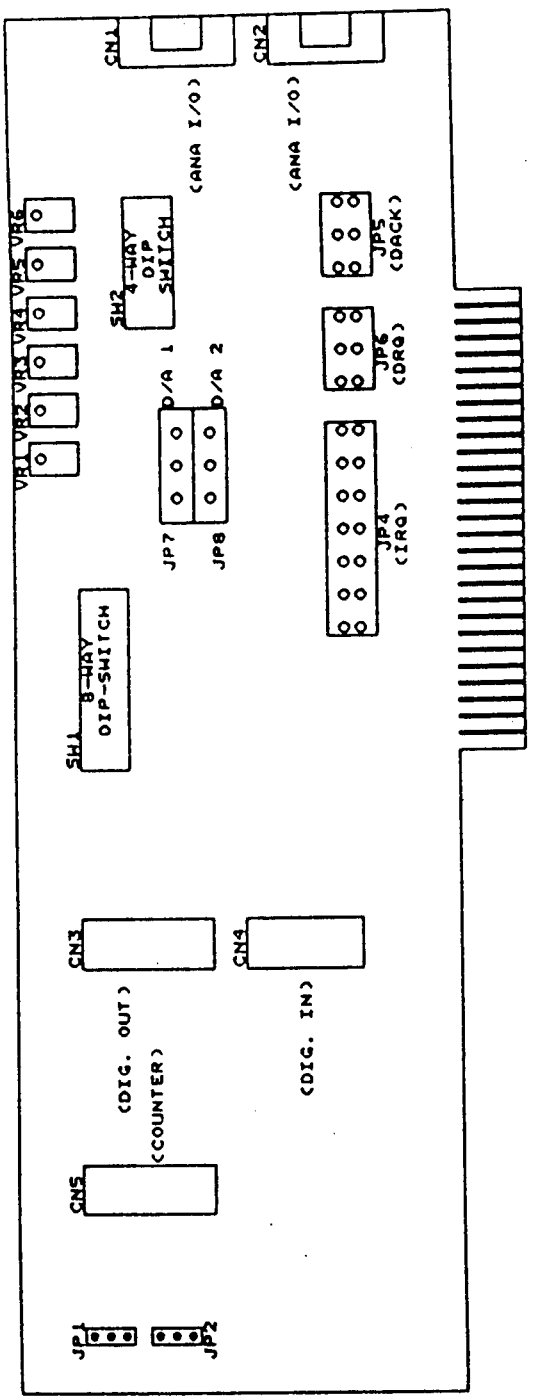
6. Byron Gottfried, "Programming with C", TMH Publications, New Delhi, 1995 Edition.

7. R.G.Dromey, "How to solve it by computer", PHI Eastern Economy Edition, New Delhi, 1982 Edition.

APPENDIX A PCL-812 BLOCK DIAGRAM



APPENDIX B PCL-812 CONNECTOR, SWITCH AND VR LOCATIONS



APPENDIX C:

FROM COTTON TO YARN:

Glossary:

1. ***Ginning*** : It is the process by which the lint is separated from the cotton seeds.

2. ***Blow room*** : Refers to the process in which the opening and the cleaning of the cotton bales is performed. This process is also done to remove the moisture present in the cotton . Mixing of cotton from various bales is done to get the required consistency.

3. ***Carding*** : The process wherein the separation of the fibres takes place is called carding. The individual fibres are grouped to form slivers.

4. ***Combing*** : This is a stage where the short fibres and neps are removed in order to improve the effective length of the yarn .

5. ***Drawing*** : In this stage the various slivers are combined and are drafted to a single strand.

6. ***Spinning*** : The process of conversion of roving to yarn , which is later wound on a cop is known as spinning.

7. ***Winding*** : It is the process where the yarn from several cops is wound on to a cone .

APPENDIX D

SERIAL COMMUNICATION

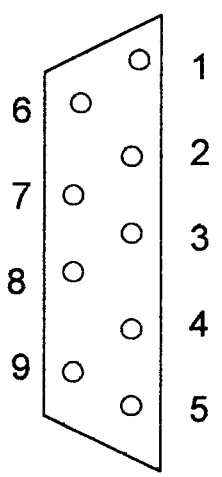
Communication of data between networks can be accomplished using either serial or parallel communication modes. In serial communication the data is transferred bit by bit along the communication channel, whereas in parallel mode the data is transferred in parallel streams of data. Serial communication has its own significant advantages over parallel communication like noise immunity ,less number of conductors, and is cost effective for long distance communication channels.

THE RS -232 INTERFACE

The RS –232 interface protocol forms the basis for serial communications. It provides the connectivity between a Data Terminal Equipment (DTE) and another DTE via a null modem connection, employing a serial binary data interchange. The serial port of a pc makes use of a RS – 232 interface for establishing data communication in a serial fashion. The serial data transfer can be implemented by employing a 9 – pin or a 25 – pin female D-type

RS – 232 connector. The pin diagrams of the 9 – pin & 25 – pin connector are shown below. The PC port connectors are male D – type connectors. The serial communications can be controlled by a programmable LSI device known as Universal Asynchronous Receiver Transmitter (UART) like the 8250. This device contains registers which can be accessed by the CPU as I/O ports. These registers can also be used to read the status of the various device buffers and operations being performed.

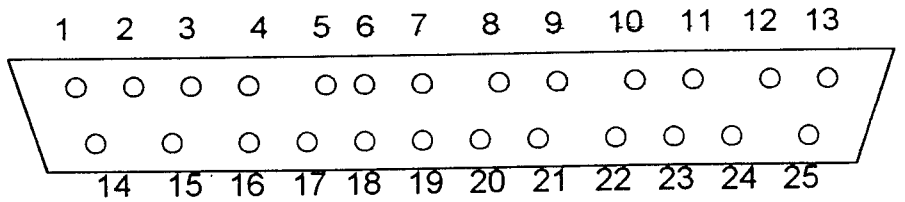
9 pin D – type Female connector – pin configuration:



Front View

1) Carrier detect (CD)
2) Received data (RD)
3) Transmitted data (TD)
4) Data terminal ready (DTR)
5) Signal ground.
6.) Data set ready (DSR)
7) Request to send (RTS)
8) Clear to send (CTS)
9) Ring Indicator (RI)

25 pin D- type Male connector:



2 – Transmit Data (TD) 3 – Receive Data (RD)
4 – Request To Send (RTS) 5 – Clear To Send (CTS)
6 – Data Set Ready (DSR) 20 – Data Terminal Ready (DTR)

In 'C',the serial communication is provided by the function ***bioscom( ).*** It can be used to initially set the communication parameters, transmit and receive data and to read the status of communication channel.

Syntax of the function:

int bioscom(int c, char byte, int port)

About the function:

The first parameter 'c' defines the mode of operation. 'c' can take the values 0,1,2 and 3 which represents respectively, setting the communication parameters, sending a character in 'byte' to the 'port', receiving a character from the 'port' and returning the current status of the communication port.

The second parameter 'byte' , which is a byte of data, defines the communication parameters of the 'port' viz., the baud rate, parity, no. of bits and no. of stop bits.

The third parameter 'port' is the port-id of the serial port. There are two communication ports in an IBM PC with the following address:

Comm. Port1 = 0 (25 pin connection)

Comm. Port2 = 1 (9 pin connection)

Connections:

For a 9 pin connector , the pin2 of the first connector is connected to the pin3 of the second and similarly the pin2 of the second to the pin3 of the first. Similar such connectors are established for pins 4 & 6 and 7 & 8.

This arrangement provides a two-way communication between the two devices/terminals.