

UNIVERSITI MALAYSIA PAHANG

BORANG PENGESAHAN STATUS TESIS♦

JUDUL: COMPUTER BASED INSTRUMENTATION SYSTEM FOR
TEMPERATURE MEASUREMENT USING RTD IN MATLAB
SESI PENGAJIAN: 2007/2008

Saya FAIZ BIN MOHD ZABRI (860209-43-5145)
(HURUF BESAR)

mengaku membenarkan tesis (Sarjana Muda/~~Sarjana~~ /~~Doktor Falsafah~~)* ini disimpan di
Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Tesis adalah hakmilik Universiti Malaysia Pahang (UMP).
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. **Sila tandakan (√)

☐

SULIT

(Mengandungi maklumat yang berdarjah keselamatan
atau kepentingan Malaysia seperti yang termaktub
di dalam AKTA RAHSIA RASMI 1972)

☐

TERHAD

(Mengandungi maklumat TERHAD yang telah ditentukan
oleh organisasi/badan di mana penyelidikan dijalankan)

☒

TIDAK TERHAD

Disahkan oleh:

(TANDATANGAN PENULIS)

(TANDATANGAN PENYELIA)

Alamat Tetap:

1764 BAGAN JERMAL,
12300 BUTTERWORTH,
PULAU PINANG

MOHD ASHRAF BIN AHMAD
(Nama Penyelia)

Tarikh: _____

Tarikh: : _____

- CATATAN:
- * Potong yang tidak berkenaan.
 - ** Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh tesis ini perlu dikelaskan sebagai atau TERHAD.
 - ♦ Tesis dimaksudkan sebagai tesis bagi Ijazah doktor Falsafah dan Sarjana secara Penyelidikan, atau disertasi bagi pengajian secara kerja kursus dan penyelidikan, atau Laporan Projek Sarjana Muda (PSM).

COMPUTER BASED INSTRUMENTATION SYSTEM FOR TEMPERATURE
MEASUREMENT USING RTD IN MATLAB APPLICATION

FAIZ BIN MOHD ZABRI

This thesis is submitted as partial fulfillment of the requirements for the award of the
Bachelor of Electrical Engineering (Hons.) (Electronics)

Faculty of Electrical & Electronics Engineering
Universiti Malaysia Pahang

NOVEMBER, 2008

DECLARATION

“All the trademark and copyrights use herein are property of their respective owner. References of information from other sources are quoted accordingly; otherwise the information presented in this report is solely work of the author.”

Signature : _____

Author : FAIZ BIN MOHD ZABRI

Date : _____

DEDICATION

Specially dedicate to
My beloved family and those people who have guided and inspired me
throughout my journey of education.

ACKNOWLEDGEMENT

In the name of Allah S.W.T, the most Gracious, the ever Merciful, Praise is to Allah, Lord of the universe and Peace and Prayers be upon His final Prophet and Messenger Muhammad S.A.W.

First, I would like to express my acknowledgment and gratitude to my supervisor, Miss Najidah Binti Hambali and also my co-supervisor, Mr Mohd Ashraf Bin Ahmad for encouragement, advice, information, motivation, guidance and co-operation that been given throughout the progress and to complete this project.

My sincere appreciation also extends to all my colleagues and others who have provided assistance at various occasions. Their views and tips are useful indeed. Unfortunately, it is not possible to list all of them in this limited space.

Finally, special thanks extended to my beloved family who had given me moral support and prayed for my success.

Thank you,

Faiz Bin Mohd Zabri

ABSTRACT

This project presents the computer based instrumentation system designed for temperature measurement using Resistance Temperature Detector, RTD. This system operated by using Matlab application. This project consists of 3 parts that are for instrument part, the hardware and the software. For instrument part, the Resistance Temperature Detector, RTD is called temperature sensitive resistor. It is positive temperature coefficient device which means that the resistance increases with temperature. The Digital Thermometer 7563 was used to read the input value of temperature and also used as temperature reference. The function of Yokogawa Temperature Transmitter PT 100 is to change the value of temperature to current and transmitted to ammeter. The hardware used to interface the temperature instrumentation with the software (Matlab). The Advantech PCI-1710HG was used as the hardware to interface from 2793 Decade Resistance Box to the Matlab application for analysis the data and find the result. The system will be operated by using GUI Matlab. Graphical User Interface, GUI is the type of user interface which allows people to interact with electronic devices like computers. This system is compatible software and can work with Advantech PCI-1710HG. The user can use this system in two ways that in automatic function or in manual function. This system was developed to find actual UUT output, output error, average, standard deviation and uncertainty. It also can plot graph and save the picture.

ABSTRAK

Projek ini memperkenalkan sistem peralatan berasaskan komputer direka untuk mengukur suhu dengan menggunakan RTD. Sistem ini beroperasi dengan menggunakan aplikasi Matlab. Projek ini terdiri daripada 3 bahagian iaitu bahagian peralatan, bahagian perkakas dan bahagian perisian. Untuk bahagian peralatan, “Resistance Temperature Detector”, RTD dikenali sebagai perintang peka suhu. Ia adalah alat peka suhu yg jelas di mana rintangan yang meningkat bersama suhu. Alat “Digital Thermometer 7563” digunakan untuk membaca nilai data kemasukan suhu dan juga digunakan sebagai suhu rujukan. Kegunaan “Yokogawa Temperature Transmitter PT 100” adalah untuk menukar nilai suhu kepada nilai arus dan dipancarkan ke meter arus. Perkakas digunakan untuk mengantaramuka di antara peralatan suhu dengan perisian Matlab. Advantech PCI-1710HG digunakan sebagai perkakas mengantaramukakan daripada “2793 Decade Resistance Box” ke aplikasi Matlab untuk menganalisis data dan mendapatkan keputusan. Sistem ini boleh beroperasi dengan menggunakan GUI Matlab. Grafikal Pengantaramuka Pengguna, GUI adalah sejenis pengantaramuka pengguna di mana membenarkan manusia berinteraksi dengan peralatan elektronik seperti Komputer. Sistem ini merupakan perisian yang bersesuaian dan boleh bekerja dengan Advantech PCI-1710HG. Pengguna boleh menggunakan sistem ini dalam dua cara iaitu dengan fungsi automatic atau fungsi manual. Sistem ini direka untuk mencari keluaran “actual UUT”, keluaran ralat, purata, sisihan piawai dan ketidakpastian. Ia juga boleh memplot graf dan menyimpan gambar graf.

TABLE OF CONTENTS

| CHAPTER | TITLE | PAGE |
|----------|-------------------------|----------|
| | Declaration | ii |
| | Dedication | iii |
| | Acknowledgement | iv |
| | Abstract | v |
| | Abstrak | vi |
| | Table of contents | vii |
| | List of tables | x |
| | List of figures | xi |
| | List of abbreviations | xiii |
| | List of appendices | xiv |
| 1 | INTRODUCTION | 1 |
| | 1.1 Overview | 1 |
| | 1.2 Problem statement | 2 |
| | 1.2.1 Current situation | 2 |
| | 1.2.2 Problem solution | 2 |
| | 1.3 Objective | 3 |
| | 1.4 Scope | 4 |

| | | |
|----------|--|----------|
| 2 | LITERATURE REVIEW | 5 |
| 2.1 | Resistance Temperature Detector, RTD | 5 |
| 2.2 | Continuous Resistance Temperature Detector Calibration Using Johnson Noise Thermometry. | 8 |
| 3 | METHODOLOGY | 9 |
| 3.1 | Instrument part | 9 |
| 3.1.1 | ISOTECH Jupiter 650B | 10 |
| 3.1.2 | Digital Thermometer 7563 | 11 |
| 3.1.3 | Yokogawa Temperature Transmitter, PT100 | 11 |
| 3.1.4 | HART 375 Field Communicator | 12 |
| 3.1.5 | Resistance Temperature Detector, RTD | 12 |
| 3.2 | Hardware part | 13 |
| 3.2.1 | Advantech PCI-1710HG | 13 |
| 3.2.2 | Common specifications | 13 |
| 3.2.3 | Pin Assignment | 14 |
| 3.3 | Software part | 15 |
| 3.3.1 | Real Time Windows Target Setup | 15 |
| 3.3.2 | Installation and configuration | 17 |
| 3.3.3 | Procedure of Creating Real Time Application | 21 |
| 3.3.4 | Creating Graphical User Interfaces | 34 |

| | | |
|----------|---|-----------|
| 4 | RESULT AND DISCUSSION | 39 |
| 4.1 | Result | 39 |
| 4.2 | Calculations | 40 |
| 4.2.1 | Desired UUT output | 40 |
| 4.2.2 | Output error (%) | 41 |
| 4.2.3 | Average | 42 |
| 4.2.4 | Standard deviation | 43 |
| 4.3 | Uncertainty evaluation | 44 |
| 4.3.1 | Uncertainty due to repeatability Of the experiment | 44 |
| 4.3.2 | Uncertainty contribution due to MSU error | 44 |
| 4.3.3 | Uncertainty due to UUT Resolution/MSU resolution | 45 |
| 4.3.4 | Combined standard uncertainty | 46 |
| 4.4 | Result | 47 |
| 4.4.1 | Result from plotting graph | 47 |
| 4.4.2 | GUI Using Matlab 7.0 | 49 |
| 4.4.3 | The Operation of system | 52 |
| 5 | CONCLUSION AND RECOMMENDATIONS | 57 |
| 5.1 | Conclusion | 57 |
| 5.2 | Recommendations | 58 |
| | REFERENCES | 59 |

LIST OF TABLES

| TABLE NO. | TITLE | PAGE |
|------------------|---|-------------|
| 4.1 | Five-point calibration of temperature transmitter | 39 |

LIST OF FIGURES

| FIGURE NO. | TITLE | PAGE |
|-------------------|--|-------------|
| 3.1 | Instrumentation of temperature measurement | 9 |
| 3.2 | ISOTECH Jupiter 650B | 10 |
| 3.3 | Digital Thermometer 7563 | 11 |
| 3.4 | Yokogawa Temperature Transmitter, PT100 | 11 |
| 3.5 | HART 375 Field Communicator | 12 |
| 3.6 | Resistance Temperature Detector, RTD | 12 |
| 3.7 | Advantech PCI-1710HG | 13 |
| 3.8 | Pin Assignment for PCI-1710 HG | 14 |
| 3.9 | Required Products of Real Time Windows Target | 16 |
| 3.10 | Simulink Model rtvdp.mdl | 20 |
| 3.11 | Create a new model | 22 |
| 3.12 | Empty Simulink model | 22 |
| 3.13 | Block Parameters of Signal Generator | 23 |
| 3.14 | Block Parameters of Analog Output | 25 |
| 3.15 | Scope Parameters Dialog Box | 26 |
| 3.16 | Scope Properties: axis 1 | 27 |
| 3.17 | Completed Simulink Block Diagram | 28 |
| 3.18 | Configuration Parameters – Solver | 29 |
| 3.19 | Configuration Parameters – Hardware Implementation | 30 |
| 3.20 | System Target File Browser | 31 |
| 3.21 | Configuration Parameters – Real-Time Workshop | 31 |
| 3.22 | Connect to target from the Simulation menu | 33 |
| 3.23 | GUIDE Quick Start | 36 |
| 3.24 | Layout Editor | 37 |
| 3.25 | M-file Editor | 38 |

| | | |
|-------------|---|-----------|
| 4.1 | Graph of MSU value ($^{\circ}\text{C}$) vs. Actual UUT output (mA) | 47 |
| 4.2 | Graph of MSU value ($^{\circ}\text{C}$) vs. Output error (%) | 48 |
| 4.3 | The starting software | 49 |
| 4.4 | The Automatic section | 50 |
| 4.5 | The Manual section | 51 |
| 4.6 | To show the range MSU | 52 |
| 4.7 | Insert value of 1 st Actual UUT output and find output error | 53 |
| 4.8 | Plotting Graph output | 53 |
| 4.9 | Plotting Graph output error | 54 |
| 4.10 | To calculate average and standard deviation | 54 |
| 4.11 | Uncertainty panel | 55 |
| 4.12 | Calculate uncertainty | 56 |

LIST OF ABBREVIATIONS

| Component | The Description |
|------------------|----------------------------------|
| RTD | Resistance Temperature Detector |
| GUI | Graphical User Interface |
| UUT | Unit under Test |
| MSU | Master Standard Unit |
| DAQ | Data acquisition System |
| JNT | Johnson noise thermometry |
| PRTs | Platinum Resistance Thermometers |
| MATLAB | Matrix Laboratory |
| CPU | Central Processing Unit |
| vs. | Versus |

LIST OF APPENDIXES

| APPENDIX | TITLE | PAGE |
|-----------------|----------------------------|-------------|
| A | T-distribution Curve Table | 60 |
| B | PCI-1710 HG Datasheet | 61 |
| C | Coding Program | 64 |

CHAPTER 1

INTRODUCTION

1.1 Overview

As we know, the students for 4 BEC were doing the experiment for subject Industrial Instrumentation (BEE 4523) at the lab manually. They need to start their experiment from connecting the instruments, find the data of experiment, calculate data and plot the graph for Point Calibration and Error Plot. So, they need more time to do this analysis and many calculations like to calculate the desired Unit under Test (UUT) output, actual Unit under Test output and the output error. Besides that, they need perform the uncertainty of measurement evaluation for one equipment calibration. They need calculate the uncertainty due to repeatability of the experiment, uncertainty contribution due to MSU error, the uncertainty due to UUT resolution/MSU resolution and combined standard uncertainty.

The computer based instrumentation system will be designed for temperature measurement using Resistance Temperature Detector, RTD. This system will be operated by using Matlab application. This system used GUI Matlab that can show the progress. It can use to solve these problems efficiently. This system developed for educational purpose. It means the students can use this system for their analysis of subject BEE 4523.

As overall, this system can look as the communication between the user and the instrument for finding the more accurate result and easy to plot the graph. This system works fast without need to do more works. The user needs to set the Master Standard Unit (MSU) value at the RTD and when values of temperatures reach at the MSU value at Digital Thermometer. The user presses the set button. The all data will transfer through the system to find the result and graph. This system also can calculate the output error, average, standard deviation and uncertainty. So, we can use this program to do the analysis.

1.2 Problem statement

1.2.1 Current situation

The students doing the experiment for subject Industrial Instrumentation in the lab. However, the problems are:

(i) Doing experiment manually

Firstly, they have more steps to setup the instruments. Then, they need to collect the data from instrument for this analysis.

(ii) More time

They need more times to do analysis for this experiment especially for temperature measurement. They rushed to find the result to calculate for output error, average, standard deviation and uncertainty. They also need to plot the graph.

1.2.2 Problem solution

This system can solve this situation and has advantages compare to another program.

- i. This system was developed for educational purpose. The student from 4 BEC can use this system to do the analysis in lab. This system suitable to use for all computer that have installed Matlab software.
- ii. This system can make work faster and easier when doing the analysis. It can calculate all calculations and plotting the graph.

1.3 Objective

The objectives of this project are:

- i. Understanding about basic concept of temperature measurement instrumentation. The function of each instrument must know and to find the reading of temperature using RTD.
- ii. Interface the temperature instrumentation with software using DAQ's (Data acquisition System) card. The PCI-1710HG is suggestion hardware that use for this project.
- iii. Developing the system using MATLAB application. The GUI Matlab was use to show the progress of result and the graph efficiently. It also can calculate the output error, average, standard deviation and uncertainty.

1.4 Scope

The scopes of the project are:

- i. Study the function of each instrument and know to read the data measurement from the instruments. The connection of each instrument are important because to avoid from error when doing the experiment.
- ii. Use the PCI-1710HG (suggestion hardware) as the medium between the instrument and the software. This DAQ card is suitable for the MATLAB software.
- iii. Choosing the MATLAB application for this project because there many features that can apply for this system especially to plot the graph directly and compatible software with DAQ board that can use for this system.

CHAPTER 2

LITERATURE REVIEW

2.1 Resistance Temperature Detector, RTD

The research is about what is Resistance Temperature Detector, RTD. The RTDs also called resistance thermometers are temperature sensors that exploit the predictable change in electrical resistance of some materials with changing temperature. As they are almost invariably made of platinum, they are often called platinum resistance thermometers (PRTs). They are slowly replacing the use of thermocouples in many industrial applications below 600 °C [1]. RTD sensors used to measure temperature by correlating the resistance of the RTD element with temperature. Most RTD elements consist of a length of fine coiled wire wrapped around a ceramic or glass core. The element is usually quite fragile, so it is often placed inside a sheathed probe to protect it. The RTD element is made from a pure material whose resistance at various temperatures has been documented. The material has a predictable change in resistance as the temperature changes; it is this predictable change that is used to determine temperature [2].

The Resistance Temperature Resistance is constructed in a number of forms and offer greater stability, accuracy and repeatability in some cases than thermocouples. While thermocouples use the Seebeck effect to generate a voltage, resistance thermometers use electrical resistance and require a small power source to operate. The resistance ideally varies linearly with temperature [2]. At low temperatures PVC, silicon

rubber or PTFE insulators are common to 250°C. Above this, glass fibre or ceramic are used. The measuring point and usually most of the leads require a housing or protection sleeve. This is often a metal alloy which is inert to a particular process. Often more consideration goes in to selecting and designing protection sheaths than sensors as this is the layer that must withstand chemical or physical attack and offer convenient process attachment points [1].

There are 3 type of resistance thermometer wiring configurations. They are two-wire configuration, see figure 1.1, three-wire configuration, see figure 1.2 and four-wire configuration, see figure 1.3. The simplest resistance thermometer configuration uses two wires. It is only used when high accuracy is not required as the resistance of the connecting wires is always included with that of the sensor leading to errors in the signal. Using this configuration you will be able to use 100 meters of cable. This applies equally to balanced bridge and fixed bridge system [1]. In order to minimize the effects of the lead resistances a three wire configuration can be used. Using this method the two leads to the sensor are on adjoining arms, there is a lead resistance in each arm of the bridge and therefore the lead resistance is cancelled out. High quality connection cables should be used for this type of configuration because an assumption is made that the two lead resistances are the same. This configuration allows for up to 600 meters of cable [1]. The four wire resistance thermometer configuration even further increases the accuracy and reliability of the resistance being measured. In the diagram above a standard two terminal RTD is used with another pair of wires to form an additional loop that cancels out the lead resistance. The above Wheatstone bridge method uses a little more copper wire and is not a perfect solution. Below is a better alternative configuration four-wire Kelvin connection that should be used in all RTD. It provides full cancellation of spurious effects and cable resistance of up to 15 Ω can be handled. Actually in four wire measurement the resistance error due to lead wire resistance is zero [1].

The advantages using RTD are RTDs is one of the most accurate temperature sensors. Not only does it provide good accuracy, it also provides excellent stability and

repeatability. RTDs are also relatively immune to electrical noise and therefore well suited for temperature measurement in industrial environments, especially around motors, generators and other high voltage equipment [2]. RTDs also stable output for long period of time, ease of recalibration and accurate readings over relatively narrow temperature spans. Their disadvantages, compared to the thermocouples, are: smaller overall temperature range, higher initial cost and less rugged in high vibration environments. They are active devices requiring an electrical current to produce a voltage drop across the sensor that can be then measured by a calibrated read out device [3].

Difference between RTDs and Thermocouple. The RTD sensing element consists of a wire coil or deposited film of pure metal. The element's resistance increases with temperature in a known and repeatable manner. RTD's exhibit excellent accuracy over a wide temperature range and represent the fastest growing segment among industrial temperature sensors [4]. A thermocouple consists of two wires of dissimilar metals welded together into a junction. At the other end of the signal wires, usually as part of the input instrument, is another junction called the reference junction. Heating the sensing junction generates a thermoelectric potential (emf) proportional to the temperature difference between the two junctions. This millivolt-level emf, when compensated for the known temperature of the reference junction, indicates the temperature at the sensing tip. Published millivolt tables assume the reference junction is at 0°C. Thermocouples are simple and familiar. Designing them into systems however is complicated by the need for special extension wires and reference junction compensation [4]. The sensor comparison chart sees at Appendix A.

2.2 Continuous Resistance Temperature Detector Calibration Using Johnson Noise Thermometry.

Johnson noise thermometry (JNT) is approaching a state of technological development to where it may be practically applied to continuous recalibration of resistance temperature detectors (RTDs) in industrial process environments. Johnson noise arises from the motion of the electrons and protons in a resistor as they thermally vibrate. Fundamentally, temperature is merely a convenient representation of the mean translational kinetic energy of an atomic ensemble. Since Johnson noise is a fundamental representation of temperature (rather than a response to temperature such as electrical resistance or thermoelectric potential), Johnson noise is immune from chemical and mechanical changes in the material properties of the sensor. As such, on-line measurement of the Johnson noise of the resistive element may be used to continuously recalibrate the RTD resistance-to-temperature relationship effectively eliminating the requirement for periodic recalibration. Measuring the RTD resistance continuously and quasi-continuously making corrections to the RTD resistance-to-temperature relationship is central to the new JNT implementation. The new JNT implementation incorporates amplifier design concepts from previous JNT developments while employing modern digital signal processing technology to remove spurious signals from the measured noise spectrum. [5]

CHAPTER 3

METHODOLOGY

3.1 Instrument Part

Figure 3.1 shows the block diagram for instrument of temperature measurement. The instruments that apply for this system are:

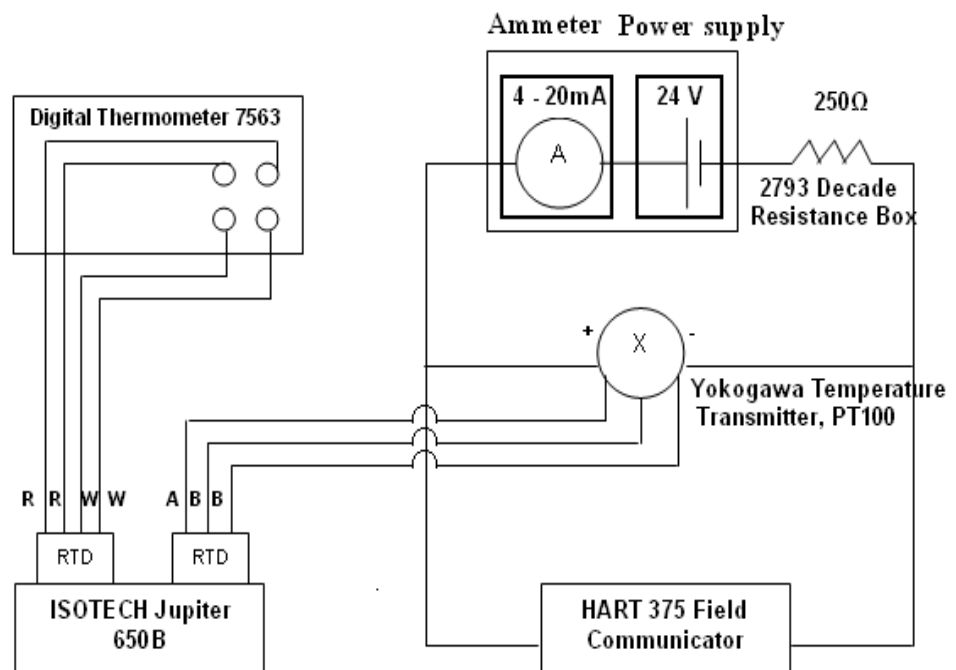


Figure 3.1: Instrumentation of temperature measurement

3.1.1 ISOTECH Jupiter 650B

The ISOTECH Jupiter 650B is designed for fast heating and cooling for suitable use. It offers industry-leading performance in an easy to use portable package. The standard insert can hold up to six thermometers. The model includes a universal sensor input allowing Resistance Temperature Detector, Thermocouples (K, N, R, S, L, B, PL2, T, J & E) along with Linear Process Inputs including 4-20mA current transmitters to be displayed on the built-in indicator.

The indicator is commonly used to display an external standard thermometer giving greater accuracy by eliminating any temperature gradient and loading errors. The indicator also can be programmed with up to five calibration points to provide high accuracy digital probe matching.

The functions of this model are to set the value of temperature and read the value of Resistance Temperature Detector, RTD as shown in Figure 3.2



Figure 3.2: ISOTECH Jupiter 650B

3.1.4 Digital Thermometer 7563

The 7563 Digital Thermometer has 16 ranges of temperature sensors and DC, V, and Ohm measuring functions as shown in Figure 3.3. It has features superior noise immunity, stability and high-speed sampling. In addition, versatile functions are suitable for system use and cover a wide variety of applications from test to R&D. For this instrument part, the digital thermometer is use to read the input value of temperature. It also use as the temperature reference.



Figure 3.3: Digital Thermometer 7563

3.1.3 Yokogawa Temperature Transmitter, PT100

The temperature transmitter is use to transmit the data from RTD to the ammeter. It also changed the value of temperature to current in range 4-20 mA. Figure 3.4 shown the Yokogawa Temperature Transmitter PT100



Figure 3.4: Yokogawa Temperature Transmitter, PT100

3.1.4 HART 375 Field Communicator

The HART 375 Field Communicator is the new standard in handheld communicator. The Hart 375 Field Communicator runs on Windows CE, a robust, real-time, operating system. The display makes it easy to read in both bright sunlight and in normal lighting. It also includes a multi-level backlight, allowing the display to be viewed in those areas with dim light. The touch sensitive display and large physical navigation buttons provide for efficient use in the field. Figure 3.5 shows the HART 375 Field Communicator.



Advanced Test Equipment Corp © 2005

Figure 3.5: HART 375 Field Communicator

3.1.5 Resistance Temperature Detector, RTD

Resistance Temperature Detector is called resistance thermometers are temperature sensors that exploit the predictable change in electrical resistance of some materials with changing temperature. As they are almost invariably made of platinum, they are often called platinum resistance thermometers (PRTs). Figure 3.6 shows the Resistance Temperature Detector, RTD.



Figure 3.6: Resistance Temperature Detector, RTD

3.2 Hardware Part

3.2.1 Advantech PCI-1710HG

The Advantech PCL-1710HG is the perfect choice to use for this project because it is low cost. The budget will be saving to buy this multifunction DAS (Data Acquisition System) card. It present in the best price and performance in the market. This custom gives higher performance and reliability with lower power consumption. The size of this hardware is half size DAS Card. The software is compatible for this PCL from MATLAB. Figure 3.7 shows the Advantech PCI-1710HG.

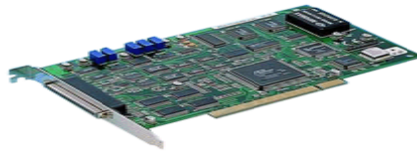


Figure 3.7: Advantech PCI-1710HG

3.2.2 Common Specifications:

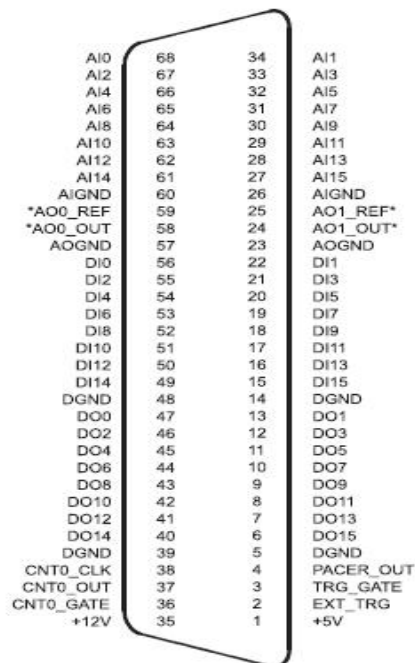
- a) Analog Input
 - Channels: 16, single-ended or 8 differential.
 - Resolution: 12 bits
 - Max. Sampling Rate: 100kS/s
 - Input range selection: (V, software programmable)
 - Auto channel/gain scanning
 - Input impedance: 1 G ohms
 - Input overvoltage: +/-30 VDC maximum

b) Analog Output

- Channels: 2
- Resolution: 12-bits
- Output range: (V, software programmable)

3.2.3 Pin Assignments

Figure 3.8 shows the Pin Assignment for PCI-1710 HG. It used to connect from the I/O board to PC.



*: Pins 23~25 and pins 57~59 are not defined for PCI-1710L/1710HGL.

Figure 3.8: Pin Assignment for PCI-1710 HG

3.3 Software Part

The MATLAB software is use for this project because it allows one to perform numerical calculations and visualize the result without need for complicated and time consuming programming. This software provides an easy way to go directly from collecting data to deriving informative result. It also accurately solves the problem, to produce graphics easily and create the code efficiently.

MATLAB software is compatible with the Advantech PCI-1710HG that will work together in this project. It also supports the entire data acquisition and analysis process, including interfacing with data acquisition devices and instruments, analyzing and visualizing the data and producing presentation quality output.

3.3.1 Real Time Windows Target Setup

Real Time Windows Target enables to run Simulink and Stateflow models in real time on desktop or laptop PC for rapid prototyping or hardware-in-the-loop simulation of control system and signal processing algorithms. A real-time execution can be created and controlled entirely through Simulink. Using Real-Time Workshop, C code can be generated, compiled and started real-time execution on Window PC while interfacing to real hardware using PC I/O Board (PCL-818). I/O device drivers are included to support an extensive selection of I/O Board, enabling to interface to other devices for experimentation, development and testing real-time systems. Simulink block diagram can be edited and Real-Time Workshop can be used to create a new real-time binary file. This integrated environment would implement any designs quickly without lengthy hand coding and debugging. Figure 3.9 shows the required product of Real Time Windows Target.

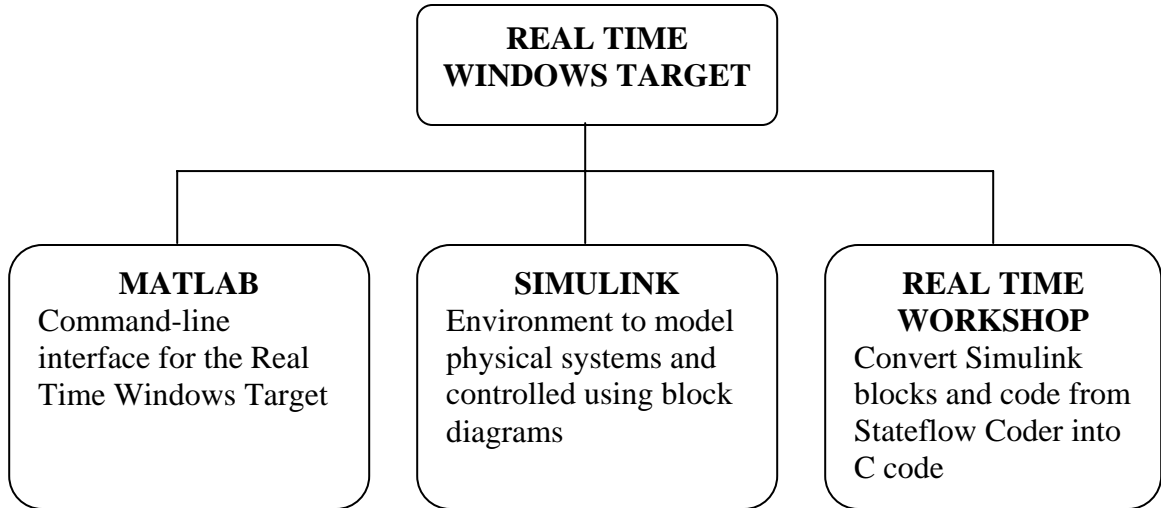


Figure 3.9: Required Products of Real Time Windows Target

Real Time Windows Target includes a set of I/O blocks that provide connections between the physical I/O Board and real time model. Hardware-in-the-loop simulations can be ran and quickly observed how Simulink model responds to real-world behavior. I/O signals can be connected using the block library for operation with numerous I/O boards.

The following types of blocks are included:

- Digital Input blocks : Connect digital input signals to Simulink block diagram to provide logical inputs.
- Digital Output blocks : Connected logical signals from Simulink block diagram to control external hardware.
- Analog Input blocks : Enable to use A/D converters that digitize analog signal for use as input to Simulink block diagram.
- Analog Output blocks : Enable Simulink block diagram to use D/A converters to output analog signal from Simulink model using I/O board(s).
- Counter Input blocks : Enable to count pulses or measure frequency using hardware counters on I/O board(s).
- Encoder Input blocks : Enable to include feedback from optical encoders.

3.3.2 Installation and Configuration

The Real-Time Windows Target is a self-targeting system where the host and the targeting computer are the same computer. It can be installed on a PC-compatible computer running Windows NT 4.0, Windows 2000 or Windows XP.

3.3.2.1 C Compiler

The Real-Time Windows Target requires one of following C compilers which not included in with the Real Time Windows Target:

- Microsoft Visual C/C ++ compiler - - Version 5.0, 6.0 or 7.0
- Watcom C/C ++ compiler - - Version 10.6 and 11.0. During installation of Watcom C/C ++ compiler, a DOS target is specified in addition to a windows target to have necessary libraries available for linking.

After installation, the MEX utility is run to select compiler as the default compiler for building real-time applications.

Real Time Workshop uses the default C compiler to generate executable code and the MEX utility uses this compiler to create MEX-files.

This procedure is executed in order to select either a Microsoft Visual C/C ++ compiler or a Watcom C/C ++ compiler before build an application. Note, the LCC compiler is not supported:

1. `mex -setup` is typed in the MATLAB window

MATLAB will display the following message:

```
Please choose your compiler for building external
interface
(MEX) files. Would you like mex to locate
installed compilers? ([y] / n ) :
```


Then a letter “y” is typed.

MATLAB will display the following message:

```
Select a compiler:
[1]: WATCOM Compiler in c: \watcaom
[2]: Microsoft compiler in c: \visual
[0]: None
Compiler:
```

Next, a number is typed. For example, number 2 is typed to select the Microsoft compiler.

MATLAB will display the following message:

```
Please verify your choices:
Compiler: Microsoft 5.0
Location: c: \visual
Are these correct? ([y] / n )
```

Finally, a letter “y” is typed.

MATLAB will reset the default compiler and display the message:

```
The default option file:
"c:\WINNT\Profiles\username\Application
Data\MathWorks\MATLAB\mexopts.bat" is being updated.
```

3.3.2.2 Installation the Kernel

During installation, all software for the Real-Time Windows Target is copied onto hard drive. The kernel is not automatically installed. Installing the kernel sets up the kernel to start running in the background each time when the computer is started. The kernel can be installed just after the Real-Time Windows Target has been installed. The installation of the kernel is necessary before a Real-Time Windows Target can be executed:

1. `rtwintgt -install` is typed in MATLAB window.

MATLAB will display the following message:

```
You are going to install the Real-Time Windows
Target kernel.
```

```
Do you want to proceed? [y] :
```

2. The kernel installation is continued by typing a letter “y”.

MATLAB will install the kernel and display the following message:

```
The Real-Time Windows Target kernel has been
successfully installed.
```

The computer has to be restart if a “restart” message being displayed.

3. The kernel should be checked whether it was correctly installed. Then, `rtwho` is typed.

MATLAB would display a message similar to

```
Real-Time Windows Target version 2.5.0 (C) The
MathWorks, Inc.
```

```
1994-2003
```

```
MATLAB performance = 100.0%
```

```
Kernel timeslice period = 1ms
```

After the kernel being installed, it remains idle, which allows Window to control the execution of any standard Windows application. Standard Windows applications include internet browsers, word processors, MATLAB and so on. It is only during real-time execution of model that the kernel intervenes to ensure that the model is given priority to use the CPU to execute each model updating at the prescribed sample intervals. Once the model update at a particular sample interval completed, the kernel releases the CPU to run any other Windows application that might need servicing.

3.3.2.3 Testing the Installation

The installation can be tested by running the model `rtvdp.mdl`. This model does not have any I/O blocks, so that this model can be run regardless of the I/O boards in computer. Running this model would test the installation by executing Real-Time Workshop, Real-Time Windows Target and Real-Time Windows Target kernel. After the Real-Time Windows Target kernel being installed, the entire installation can be tested by building and running a real-time application. The Real-Time Windows Target includes the model `rtvdp.mdl`, which already has the correct Real-Time Workshop options selected for users:

1. `rtvdp` is typed in MATLAB window.

The Simulink model `rtvdp.mdl` window will be opened as shown in Figure 3.10

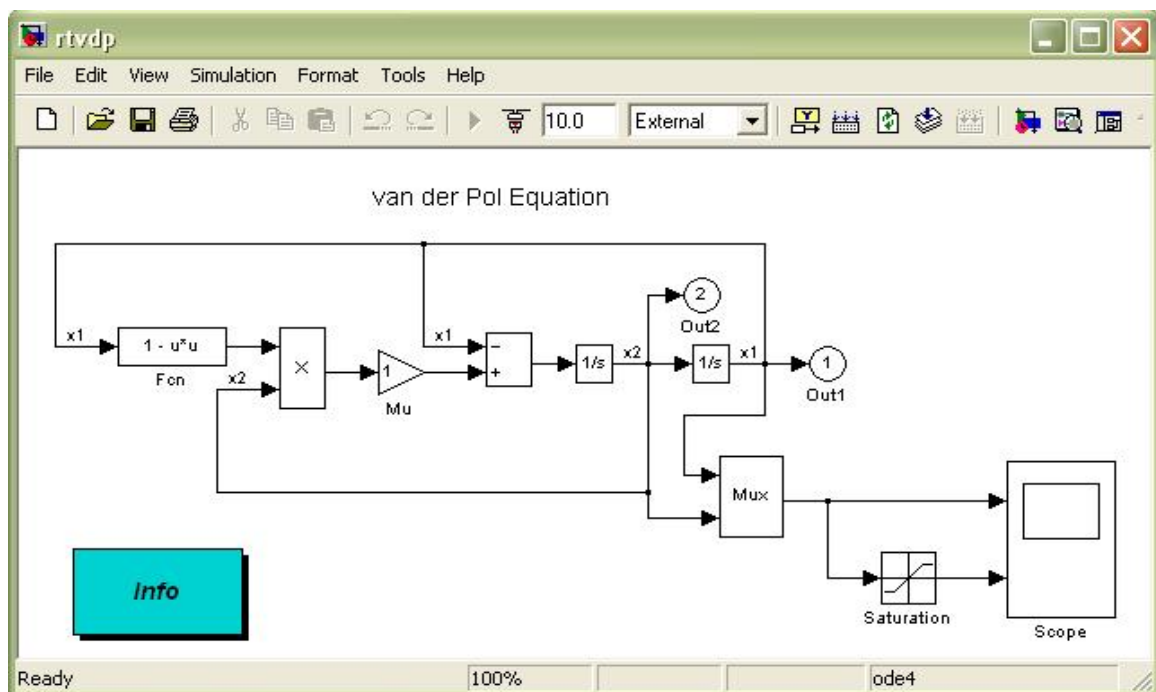


Figure 3.10: Simulink Model `rtvdp.mdl`

2. From the **Tools** menu, it should be pointed to **Real-Time Workshop**, and then clicked **Build Model**. The MATLAB window will display the following messages:

```

### Starting Real-Time Workshop build for model:
rtvdp
### Invoking Target Language Compiler on rtvdp.rtw
. . .
### Compiling rtvdp.c
. . .
### Created Real-Time Windows Target module
rtvdp.rwd.
### Successful completion of Real-Time Workshop
build procedure for model: rtvdp

```

3. From the **simulation** menu, **External** should be clicked and followed by clicking **Connect to target**.

The MATLAB window displayed the following message:

```
Model rtvdp loaded
```

4. **Start Real-Time Code** is clicked from **Simulation** menu.
The Scope window will display the output signals. After the Real-Time Windows Target has been successfully installed and the real-time application has been run, Scope window should indicate such a figure.
5. From **Simulation** menu, after the **Stop Real-Time Code** is clicked. The real-time application will stop running and then the Scope window will stop displaying the output signals.

3.3.3 Procedures of Creating Real Time Applications

3.3.3.1 Creating a Simulink Model

This procedure explains how to create a simple Simulink model. This model is used as an example to learn other procedures in the Real-Time Windows Target. A

Simulink model has to be created before it can run a simulation or create a real-time application:

1. Simulink is typed in the MATLAB Command Window.
The Simulink Library Browser window is opened as shown in Figure 3.11.
2. From the toolbar, the **Create a new model** button is clicked.

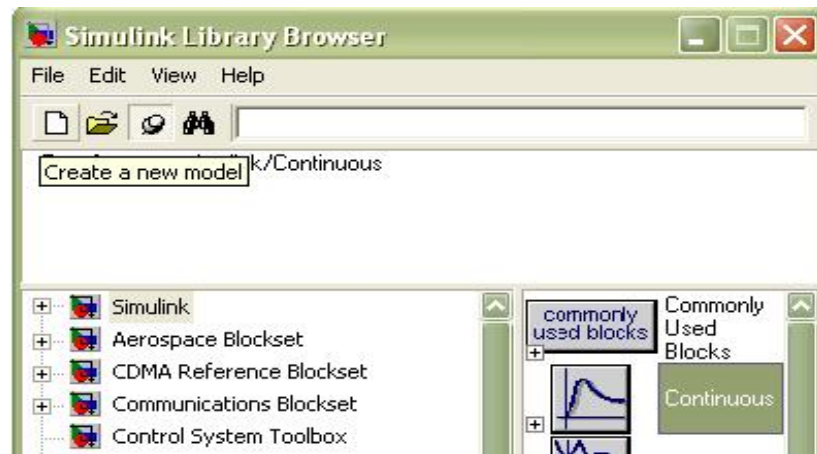


Figure 3.11: Create a new model

An empty Simulink window is opened. With the toolbar and status bar disabled, the window looks like following figure 3.12 (Figure).

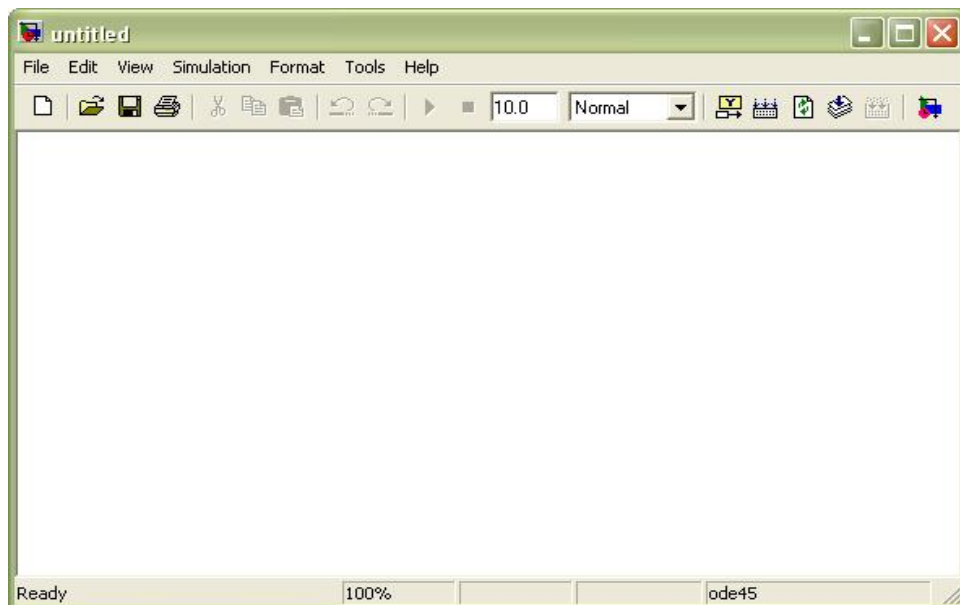


Figure 3.12: Empty Simulink model

3. In the Simulink Library Browser window, **Simulink** is double-clicked and then **Sources** is also double-clicked. Next, **Signal Generator** is clicked and dragged to Simulink window.

Sinks is clicked. **Scope** is clicked and dragged to the Simulink window. Real-Time Windows Target is clicked. **Analog Output** is clicked and dragged to the Simulink window.

4. The **Signal Generator** output is connected to the scope input by clicking-and-dragging a line between the blocks. Likewise, the **Analog Output** input is connected to the connection between **Scope** and **Signal Generator**.

5. The Signal Generator block is double clicked. The **Block Parameters** dialog box opened. From the **Wave form** list, square is selected.

In the **Amplitude** text box, 0 . 25 is entered.

In the **Frequency** text box, 2 . 5 are entered.

From the **Units** list, Hertz is selected.

The **Block Parameters** dialog box is shown in Figure 3.13.

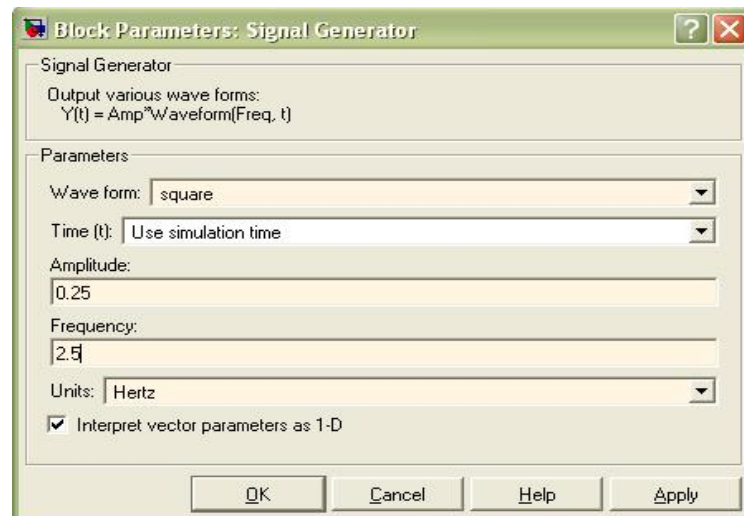


Figure 3.13: Block Parameters of Signal Generator

6. **OK** is clicked.
7. The analog output block is double clicked.
The **Block Parameters** dialog box will open.

8. The **Install new board** button is clicked. From the list, it should be pointed to manufacturer and then clicked a board name. For example, it should be pointed to Advantech and then click PCL818.
9. One of the following is selected:
 - For an **ISA** bus board, a base address is entered. This value must match the base address switches or jumpers set on the physical board. For example, to enter a base address of 0x300 in the address box, 300 is typed. The base address also could be selected by selecting check boxes A9 through A3.
 - For a **PCI** bus board, the PCI slot is entered or the Auto-detect check box is selected.
10. The **Test** button is clicked.
The Real-Time Windows Target tried to connect to the selected board and the following message would display if successful.
11. On the message box, **OK** is clicked.
12. The same value as entered in the **Fixed step size** box from the **Configuration Parameters** dialog box is entered in the Sample time box. For example, 0.001 is entered.
13. A channel vector that selected the analog input channels that are using on this board is entered in the **Output channels** box. The vector can be any valid MATLAB vector form. For example, to select analog output channel on PCL818 board 1 is entered.
14. The input range for the entire analog input channel that has been entered in the Input channels box is chosen from the **Output range** list. For example, with the PCL818 board, 0 to 5V is chosen.
15. From the Block Input signal list, the following options is chosen:
 - Volts – Expected a value equal to the analog output range.
 - Normalized unipolar – Expected a value between 0 and +1 that is converted to the full range of the output voltage regardless of the output voltage range. For example, an analog output range of 0 to +5 volts and -5 to +5 volts would both converted from values between 0 and +1.

- Normalized bipolar – Expected a value between -1 and +1 that is converted to the full range of output voltage regardless of the output voltage range.
 - Raw – Expected a value of 0 to 2^n-1 . For example, a 12-bit A/D converter would expected a value between 0 and $2^{12}-1$ (0 to 4095). The advantage of this method is the expected value is always an integer with no round off error.
16. The initial value is entered for each analog output channel that has been entered in the **Output Channels** box. For example, if 1 is entered in the **Output Channels** box and the initial value of 0 volts is needed, 0 is entered.
17. The final value is entered for each analog channel that has been entered in **Output Channels** box. For example, if 1 is entered in the **Output Channels** box and the final value of 0 volts is needed, 0 is entered.

The dialog box would look similar to the Figure 3.14 if Volts is chosen.

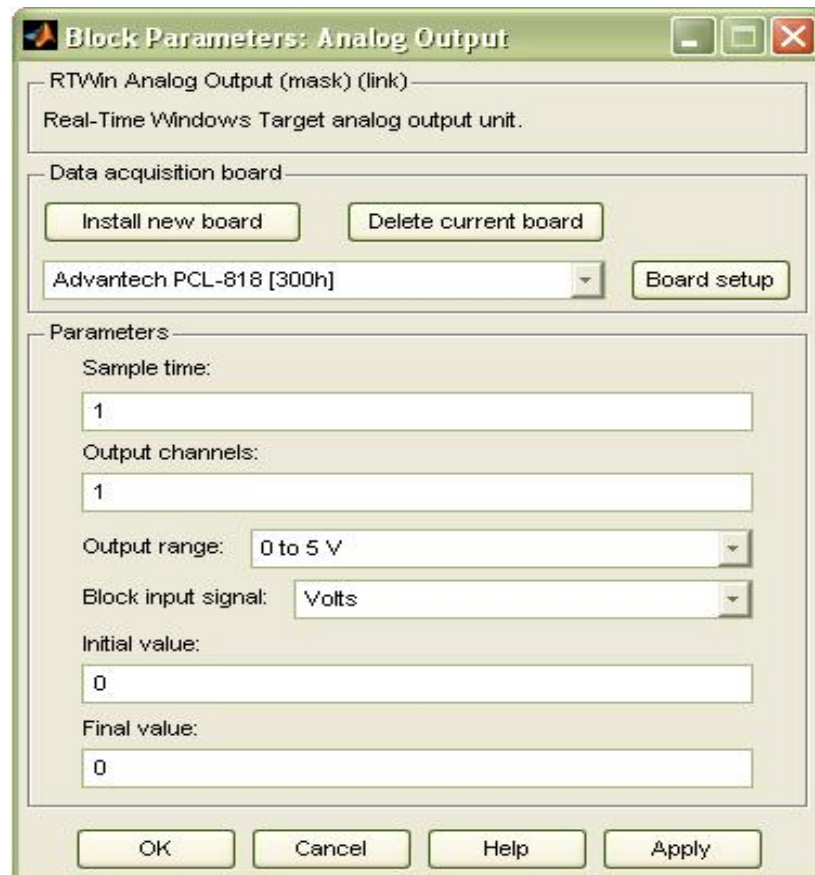


Figure 3.14: Block Parameters of Analog Output

18. One of following is executed:
 - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
 - **OK** is clicked to apply the changes to the model and the Block Parameters: Analog Output dialog box will close.
19. **Parameters** dialog box is closed and the parameter values are saved with the Simulink model.
20. In the Simulink window, the Scope block is double clicked.
A Scope window will open.
21. The Parameters button is clicked.
A Scope parameters dialog box will open.
22. The **General** tab is clicked. The number of graphs that is needed in one Scope window is entered in the **Number of axes** box. For example, 1 is entered for a single graph. Do not select the **floating scope** check box. In the **Time range** box, upper value the time range is entered. For example, 1 second is entered. From the **Tick labels** list, `bottom axis only` is chosen. From the **Sampling** list, `decimation` is chosen and 1 is entered in the text box.

The **Scope parameters** dialog box would look like such a Figure 3.15 as shown.



Figure 3.15: Scope Parameters Dialog Box

23. One of following done:
 - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
 - **OK** is clicked to apply the changes to the model and the **Scope parameters** dialog box is closed.
24. In the Scope window, it should be pointed to the y-axis and then right clicked.
25. Axes Properties is clicked from the pop-up menu.
26. The Scope properties: axis 1 dialog box is opened. In the Y-min and Y-max text boxes, the range for the y-axis is entered in the Scope window. For example, -2 and 2 are entered as shown in the Figure 3.16

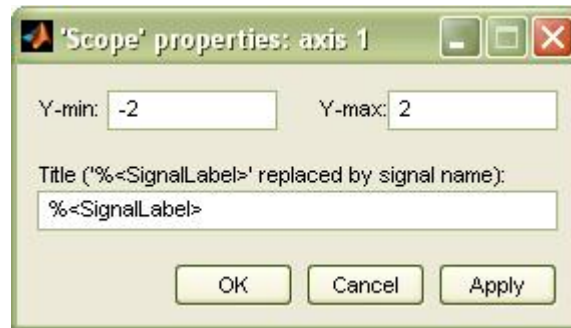


Figure 3.16: Scope Properties: axis 1

27. One of the following is done:
 - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
 - **OK** is clicked to apply the changes to the model and the **Axes Parameters** dialog box is closed.

The completed Simulink block diagram is shown in Figure 3.17.

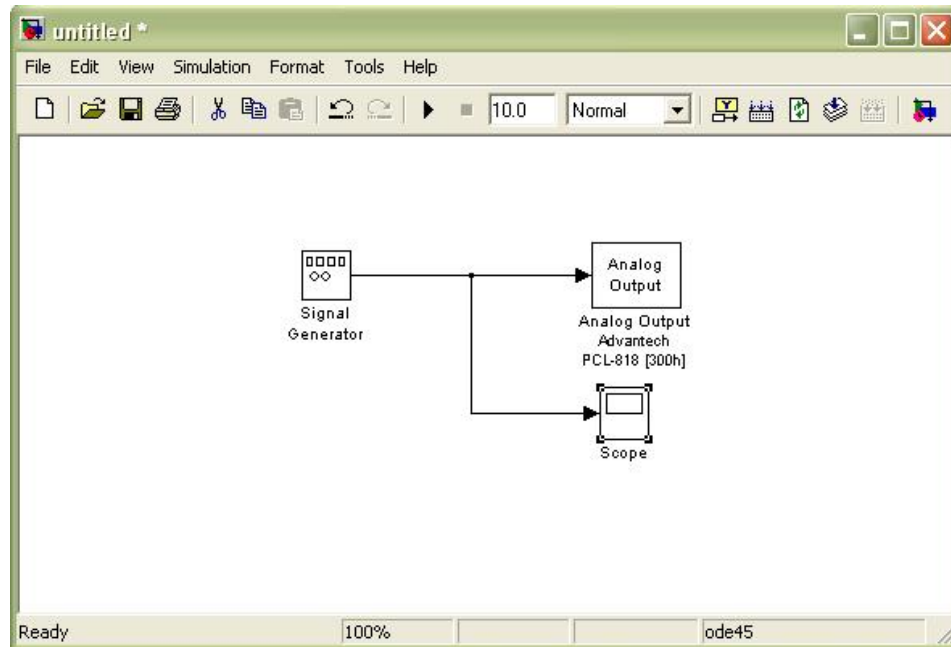


Figure 3.17: Completed Simulink Block Diagram

Save As is clicked from the **File** menu. The **Save As** dialog diagram box is opened. In the **File name** text box, a filename for the Simulink model is entered and **Save** is clicked. For example, `rtwin_model` is typed. Simulink saved the model in the file `rtwin_model.mdl`.

3.3.3.2 Entering Configuration Parameters for Simulink

The configuration parameters give information to Simulink for running a simulation. After create a Simulink model, the configuration parameters could be entered for Simulink.

1. In the Simulink window, **Configuration Parameters** is clicked from the Simulation menu. In the **Configuration Parameters** dialog box, the **Solver** tab is clicked.

The **Solver** pane will open.

2. In the **Start time** box, 0.0 is entered. In the **Stop time** box, the amount of time that the model needs to run is entered. For example, 99999 seconds is entered.
3. From the **Type** list, Fixed-step is chosen. Real-Time Workshop does not support variable step solvers.
4. From the **Solver** list, a solver is chosen. For example, the general purpose solver ode5 (Dormand-Prince) is chosen.
5. In the **Fixed step size** box, a sample time is entered. For example, 0.001 seconds is entered for the sample rate of 1000 samples/second.
6. From the **Tasking Mode** list, SingleTasking is chosen. Multitasking is chosen for models with blocks that have different sample times.

The **Solver** pane would look similar to the Figure 3.18.

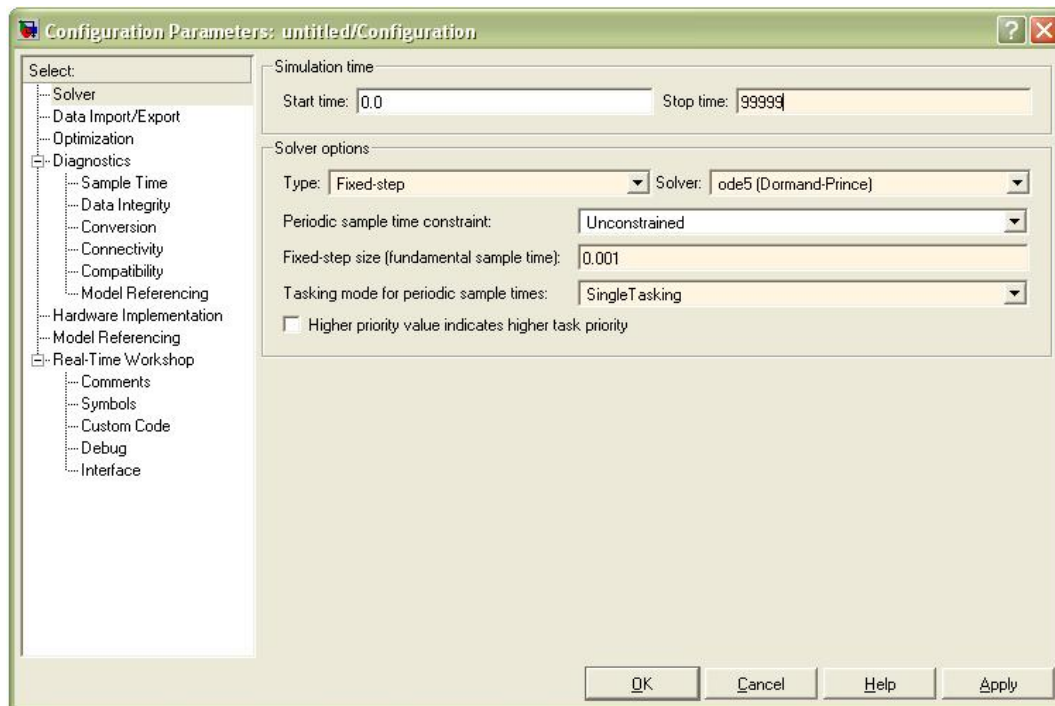


Figure 3.18: Configuration Parameters – Solver

7. One of following is done:
 - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
 - **OK** is clicked to apply the changes to the model and the **Configuration Parameters** dialog box is closed.

3.3.3.3 Entering Simulation Parameters for Real-Time Workshop

The Simulation Parameters are used by Real-Time Workshop for generating C code and building a real-time application.

1. In the Simulink window, **Configuration Parameters** is clicked from the **Simulation** menu as shown in Figure 3.19.
2. The **Hardware Implementation** node is clicked.
3. From the **Device type** list, 32-bit Real-Time Windows Target is chosen.

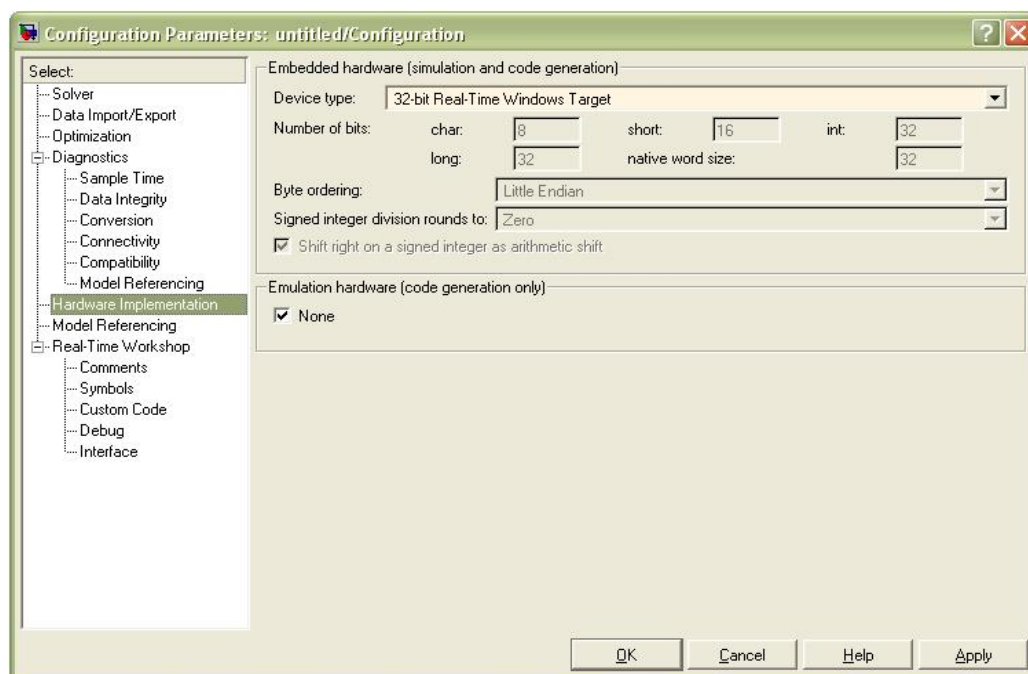


Figure 3.19: Configuration Parameters – Hardware Implementation

4. The **Real-Time Workshop** node is clicked.
The **Real-Time Workshop** pane will open.
5. In the **Target selection section**, the **Browse** button is clicked at the **RTW system target file** list. The **System Target File Browser** will open as shown in Figure 3.20.
6. The system target file is selected for the Real-Time Windows Target and **OK** is clicked.

| | |
|------------------|---|
| proosek.tlc | OSEK Target for 3Soft ProOSEK Implementat |
| rsim.tlc | Rapid Simulation Target |
| rtwin.tlc | Real-Time Windows Target |
| rtwsfcn.tlc | S-function Target |
| ti_c2000_ert.tlc | Embedded Target for TI C2000 DSP (ERT) |
| ti_c2000_grt.tlc | Embedded Target for TI C2000 DSP (GRT) |

Figure 3.20: System Target File Browser

The system target file `rtwin.tlc`, the template makefile `rtwin.tmf` and the make command `make_rtw` are automatically entered into the **Real-Time Workshop** pane.

Although not visible in the **Real-Time Workshop pane**, the external target interface MEX file `rtwinext` is also configured after **OK** is clicked. This allows external mode to pass new parameters to the real-time application and to return signal data from the real-time application. The data is displayed in Scope blocks or saved with signal logging.

The **Real-Time Workshop** pane would look similar to the Figure 3.21.

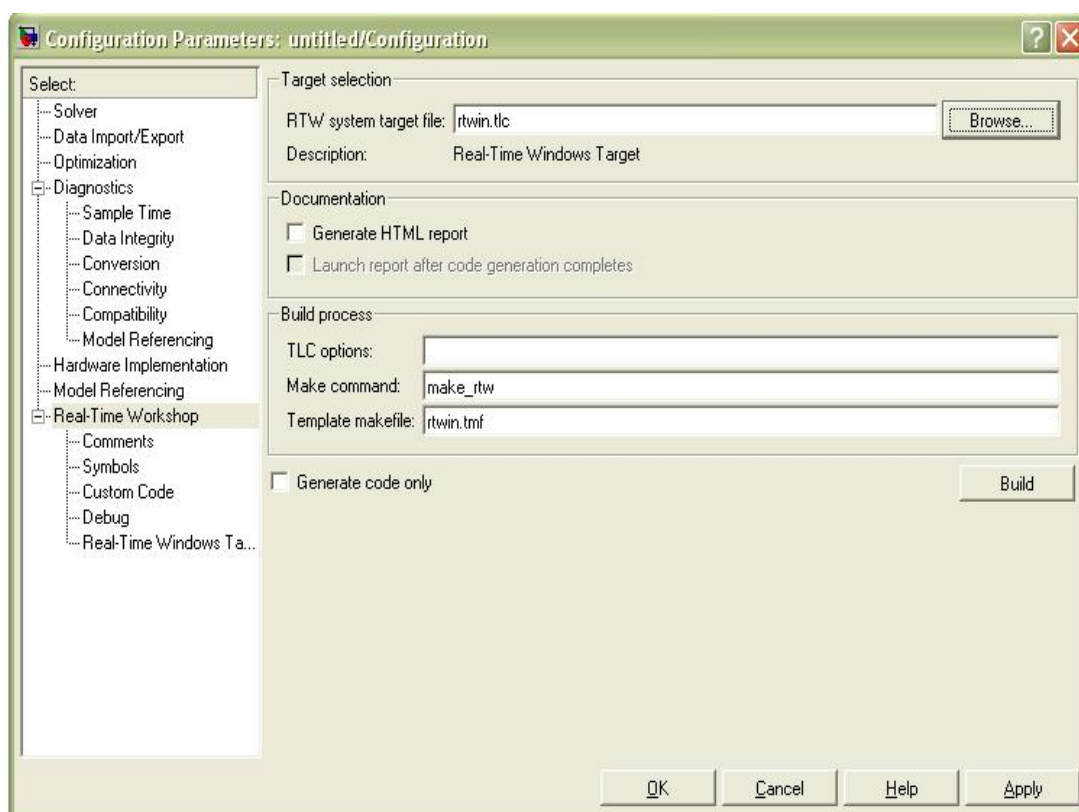


Figure 3.21: Configuration Parameters – Real-Time Workshop

7. One of following is done:
 - **Apply** is clicked to apply the changes to the model and the dialog box is left open.
 - **OK** is clicked to apply the changes to the model and the **Configuration Parameters** dialog box is closed.

3.3.3.4 Creating a Real-Time Application

Real-Time Workshop generates C code from the Simulink model and then the **Microsoft Visual Basic C++** compiler compiles and links that C code into a real-time application. After parameters are entered into the **Configuration Parameters** dialog box for Real-Time Workshop, a real-time application could be built.

1. In the Simulink window and from the Tools menu, it should be pointed to the Real-Time Workshop and then clicked Build Model. The build process does the following:
 - Real-Time Workshop creates the C code source files `rtwin_model.c` and `rtwin_model.h`.
 - The make utility `make_rtw.exe` creates the makefile `rtwin_model.mk` from the template makefile `rtwin.tmf`.
 - The make utility `make_rtw.exe` builds the real-time application `rtwin_model.rwd` using the makefile `rtwin_model.mk` created above. The file `rtwin_model.rwd` is binary files that refer to as the real-time application. The real-time application could be run with the Real-Time Windows Target kernel.
2. The Simulink model is connected to real-time application.
 After the real-time application is created, MATLAB could be closed and started again later and then the executable is connected and run without having to rebuild.


3.3.3.5 Running a Real-Time Application

The real-time application is run to observe the behavior of the model in real time with the generated code.

The process of connecting consist of

- Establishing a connection between your Simulink model and the kernel to allow exchange of commands, parameters and logged data.
- Running the application in real time.

After the real-time application is built, the model could be run in real time.

1. From the Simulation menu, External is clicked and then Connect To Target is connected from the Simulation menu, Also, it could be connected to the target from the toolbar by clicking . It can be seen I Figure 3.22.

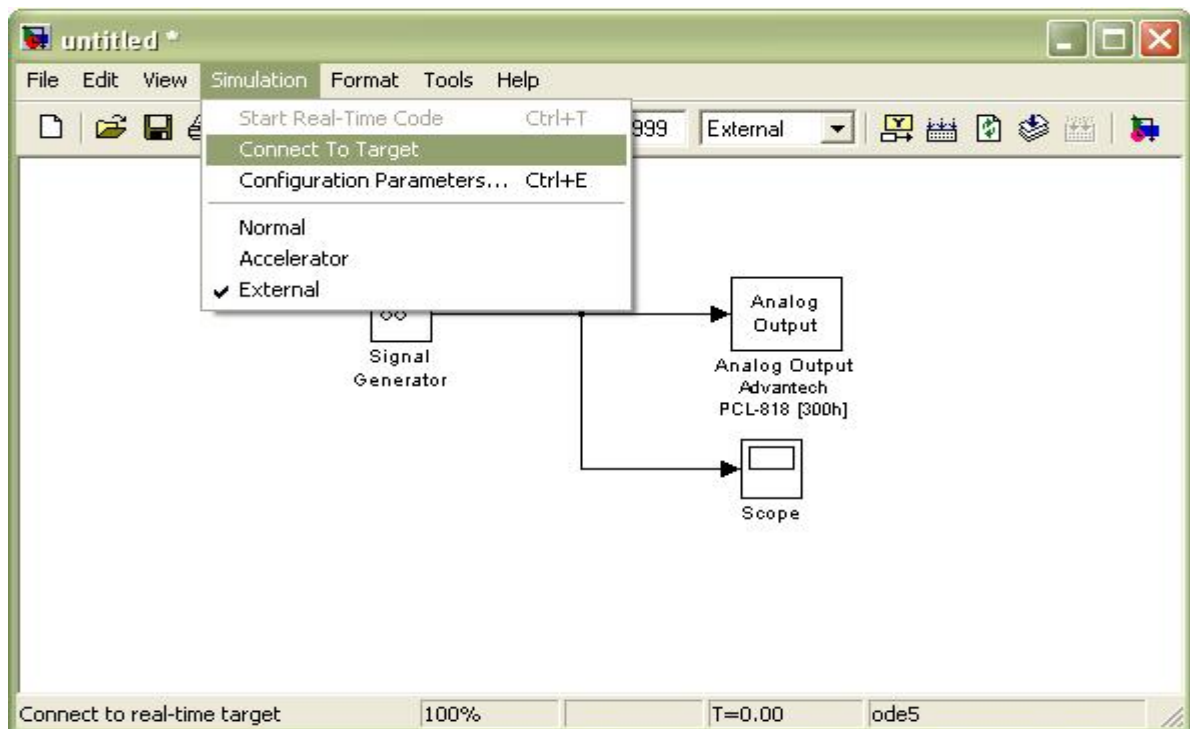


Figure 3.22: Connect to target from the Simulation menu

MATLAB will display the message

```
Model rtwin_model loaded
```


2. In the Simulation window and from the **Simulation** menu, **Start Real-Time Code** is clicked. The execution also could be started from the toolbar by clicking Start icon.

Simulink runs the execution and plots the signal data in the Scope window.

In the model, the Scope window displays 1000 samples in 1 second, increases the time offset and then displays the samples for the next 1 second.

Note:

Transfer of data is less critical than calculating the signal output at the selected sample interval. Therefore, data transfer runs at a lower priority in the remaining CPU time after real-time application computations are performed while waiting for another interrupt to trigger the next real-time application update. The result may be a loss of data points displayed in the Scope window.

3. One of the following is done:
 - The execution is let to be run until it reaches the stop time.
 - **Stop Real-Time Code** is clicked from the **Simulation** menu.

The real-time application is stopped.

4. In the Simulation window, **Disconnected From Target** is clicked from the **Simulation** menu.
5. From the **Simulation** menu, **External** is clicked

MATLAB will display the message

```
Model rtwin_model unloaded
```

3.3.4 Creating Graphical User Interfaces

MATLAB implements GUIs as figure windows containing various styles of uicontrol objects. You must program each object to perform the intended action when activated by the user of the GUI. In addition, you must be able to save and launch your GUI. All of these tasks are simplified by GUIDE, MATLAB's graphical user interface development environment.

3.3.4.1 GUI Development Environment

The process of implementing a GUI involves two basic tasks:

- (i) Laying out the GUI components
- (ii) Programming the GUI components

GUIDE primarily is a set of layout tools. However, GUIDE also generates an M-file that contains code to handle the initialization and launching of the GUI. This M-file provides a framework for the implementation of the *callbacks* – the functions that execute when users activate components in the GUI.

The Implementation of a GUI

While it is possible to write an M-file that contains all the commands to lay out a GUI, it is easier to use GUIDE to lay out the components interactively and to generate two files that save and launch the GUI:

- (i) A FIG-file – contains a complete description of the GUI figure and all of its children (uicontrols and axes), as well as the values of all object properties.
- (ii) An M-file – contains the functions that launch and control the GUI and the callbacks, which are defined as subfunctions. This M-file is referred to as the *application M-file* in this documentation.

3.3.4.2 Starting Guide

Start GUIDE by typing `guide` at the MATLAB command prompt. This displays the **GUIDE Quick Start** dialog, as shown in the following Figure 3.23.

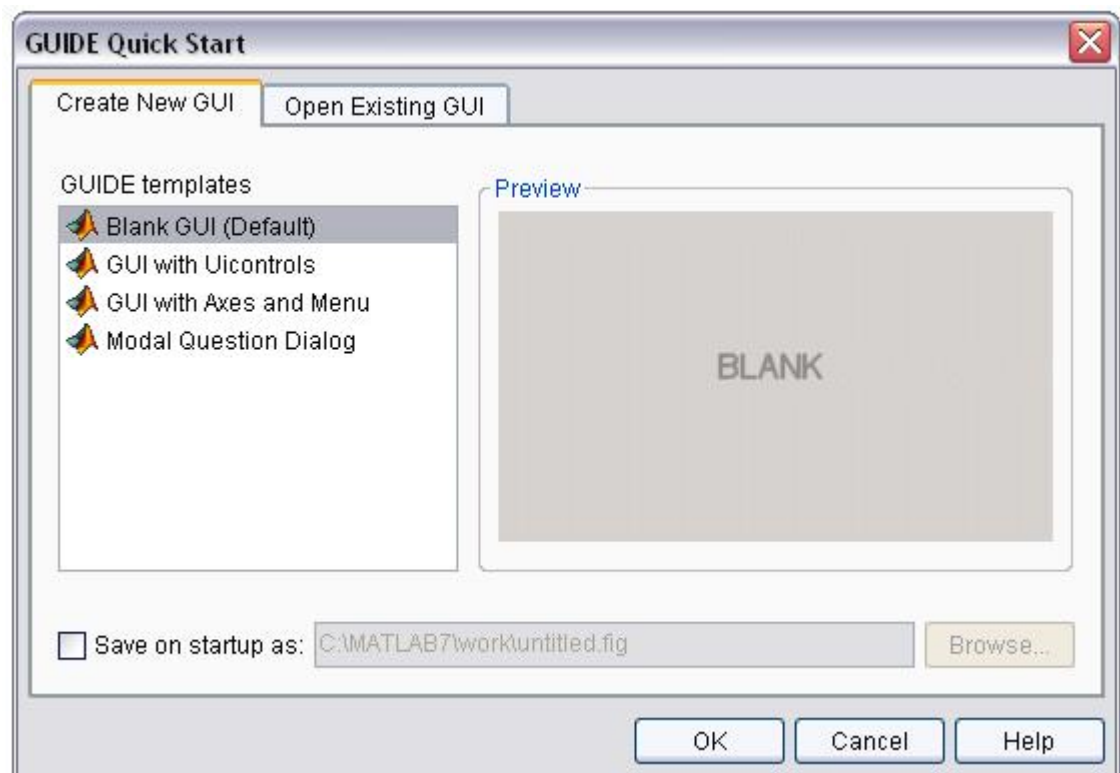


Figure 3.23: GUIDE Quick Start

From the Quick Start dialog, the user can:

- (i) Create a new GUI from one of the GUIDE templates.
- (ii) Open an existing GUI.

3.3.4.3 The Layout Editor

When the user opened a GUI in GUIDE, it is displayed in the Layout Editor, which is the control panel for all of the GUIDE tools. The following Figure 3.24 shows the Layout Editor with a blank GUI template.

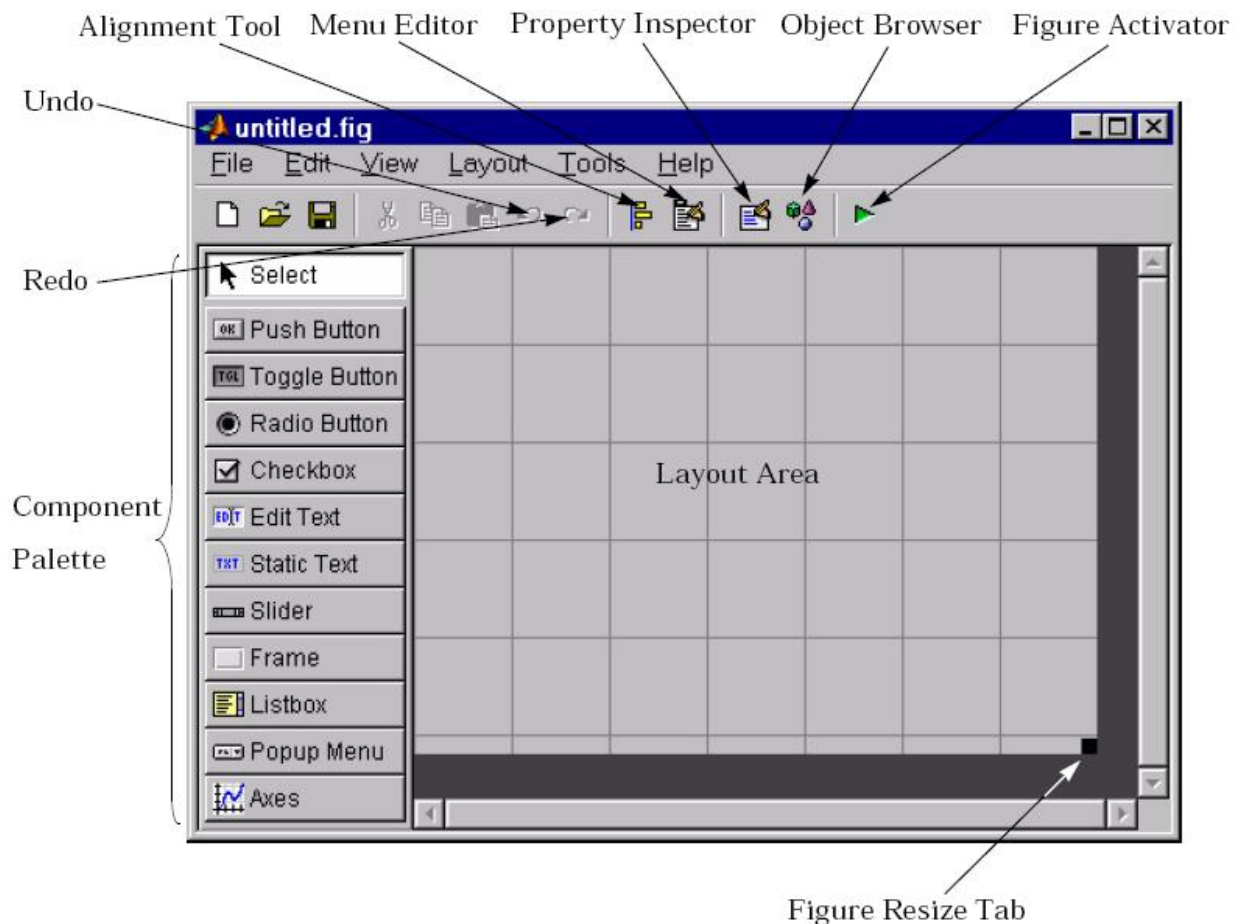


Figure 3.24: Layout Editor

The user can lay out the GUI by dragging components, such as panels, push buttons, pop-up menus, or axes, from the component palette, at the left side of the Layout Editor, into the layout area.

3.3.4.4 Programming a GUI

After laying out the GUI and setting component properties, the next step is to program the GUI. The user programs the GUI by coding one or more callbacks for each of its components. Callbacks are functions that execute in response to some action by the user. A typical action is clicking a push button.

A GUI's callbacks are found in an M-file that GUIDE generates automatically. GUIDE adds templates for the most commonly used callbacks to this M-file, but the user may want to add others. Use the M-file Editor to edit this file.

The following Figure 3.25 shows the Callback template for a push button.

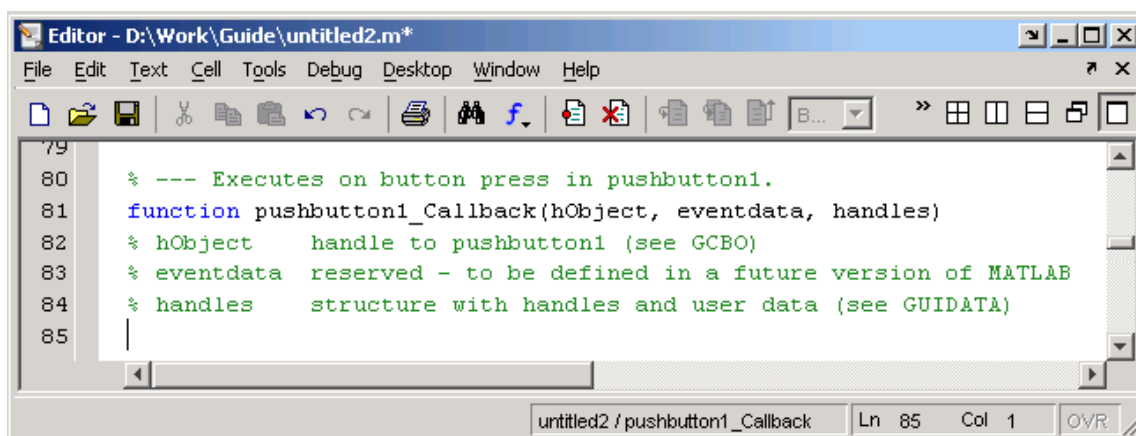


Figure 3.25: M-file Editor

CHAPTER 4

RESULT AND DISCUSSIONS

4.1 Results

The Table 4.1 shows the result of experiment that using the Resistance Temperature Detector, RTD. The results can show in the GUI Matlab that used in this experiment.

Table 4.1: Five-point calibration of temperature transmitter

| No % | MSU applied value, ($^{\circ}\text{C}$) | Desired UUT output, (mA) | 1 st Actual UUT output, (mA) | 2 nd Actual UUT output, (mA) | Output error % | Average | Standard deviation |
|---------|---|-----------------------------------|--|--|-------------------|---------|-----------------------|
| 0 | 50.0 | 4.0 | 3.8 | 3.89 | 5.00 | 3.845 | 0.00405 |
| 25 | 87.5 | 8.0 | 7.9 | 7.99 | 1.25 | 7.945 | 0.00405 |
| 50 | 125.0 | 12.0 | 12.0 | 11.98 | 0.00 | 11.99 | 0.0002 |
| 75 | 162.5 | 16.0 | 15.9 | 15.98 | 0.63 | 15.94 | 0.0032 |
| 100 | 200.0 | 20.0 | 19.9 | 19.89 | 0.50 | 19.895 | 5e-005 |

The equation below shows how to find the value of Desired UUT output and output error %.

Note: MSU = Master Standard Unit UUT = Unit under Test

$$\text{Desired output} = \frac{x}{100}(\text{URV} - \text{LRV}) + \text{LRV} \quad (1)$$

Where; x = i th point
 URV = Upper Range Value
 LRV = Lower Range Value

$$\text{Output error \%} = \frac{\text{Desired UUT output} - \text{Actual UUT output}}{\text{Desired UUT output}} \times 100\% \quad (2)$$

4.2 Calculation:

4.2.1 Desired UUT output:

a) 50.0 °C

$$\begin{aligned} \text{Desired output} &= \frac{0}{100}(20\text{m} - 4\text{m}) + 4\text{m} \\ &= 4\text{mA} \end{aligned}$$

b) 87.5 °C

$$\begin{aligned} \text{Desired output} &= \frac{25}{100}(20\text{m} - 4\text{m}) + 4\text{m} \\ &= 8\text{mA} \end{aligned}$$

c) 125.0 °C

$$\begin{aligned} \text{Desired output} &= \frac{50}{100}(20\text{m} - 4\text{m}) + 4\text{m} \\ &= 12\text{mA} \end{aligned}$$

d) 162.5 °C

$$\begin{aligned}\text{Desired output} &= \frac{75}{100}(20\text{m} - 4\text{m}) + 4\text{m} \\ &= 16\text{mA}\end{aligned}$$

e) 200.0 °C

$$\begin{aligned}\text{Desired output} &= \frac{100}{100}(20\text{m} - 4\text{m}) + 4\text{m} \\ &= 20\text{mA}\end{aligned}$$

4.2.2 Output error (%)

a) 50.0 °C

$$\begin{aligned}\text{Output error \%} &= \frac{4.0\text{m} - 3.8\text{m}}{4.0\text{m}} \times 100\% \\ &= 5\%\end{aligned}$$

b) 87.5 °C

$$\begin{aligned}\text{Output error \%} &= \frac{8.0\text{m} - 7.9\text{m}}{8.0\text{m}} \times 100\% \\ &= 1.25\%\end{aligned}$$

c) 125.0 °C

$$\begin{aligned}\text{Output error \%} &= \frac{12.0\text{m} - 12.0\text{m}}{12.0\text{m}} \times 100\% \\ &= 0\%\end{aligned}$$

d) 162.5 °C

$$\begin{aligned}\text{Output error \%} &= \frac{16.0\text{m} - 15.9\text{m}}{16.0\text{m}} \times 100\% \\ &= 0.63\%\end{aligned}$$

e) 200.0 °C

$$\begin{aligned}\text{Output error \%} &= \frac{20.0\text{m} - 19.9\text{m}}{12.0\text{m}} \times 100\% \\ &= 0.5\%\end{aligned}$$

4.2.3 Average

a) 50.0 °C

$$\begin{aligned}\text{Average1} &= \frac{3.8\text{m} + 3.89\text{m}}{2} \\ &= 3.845\end{aligned}$$

b) 87.5 °C

$$\begin{aligned}\text{Average2} &= \frac{7.9\text{m} + 7.99\text{m}}{2} \\ &= 7.945\end{aligned}$$

c) 125.0 °C

$$\begin{aligned}\text{Average3} &= \frac{12.0\text{m} + 11.98\text{m}}{2} \\ &= 11.99\end{aligned}$$

d) 162.5 °C

$$\begin{aligned}\text{Average4} &= \frac{15.9\text{m} + 15.98\text{m}}{2} \\ &= 15.94\end{aligned}$$

e) 200.0 °C

$$\begin{aligned}\text{Average5} &= \frac{19.9\text{m} + 19.89\text{m}}{2} \\ &= 19.895\end{aligned}$$

4.2.4 Standard Deviation

$$S(x_k) = \sqrt{\frac{1}{(n-1)} \sum_{j=1}^k (x_k - \bar{x})^2} \quad (3)$$

a) 50.0 °C

$$\begin{aligned} \text{Standard deviation1} &= \sqrt{\frac{1}{(2-1)} [(3.8 - 3.845)^2 + (3.89 - 3.845)^2]} \\ &= 0.00405 \end{aligned}$$

b) 87.5 °C

$$\begin{aligned} \text{Standard deviation2} &= \sqrt{\frac{1}{(2-1)} [(7.9 - 7.945)^2 + (7.99 - 7.945)^2]} \\ &= 0.00405 \end{aligned}$$

c) 125.0 °C

$$\begin{aligned} \text{Standard deviation3} &= \sqrt{\frac{1}{(2-1)} [(12.0 - 11.99)^2 + (11.98 - 11.99)^2]} \\ &= 0.0002 \end{aligned}$$

d) 162.5 °C

$$\begin{aligned} \text{Standard deviation4} &= \sqrt{\frac{1}{(2-1)} [(15.9 - 15.94)^2 + (15.98 - 15.94)^2]} \\ &= 0.003 \end{aligned}$$

e) 200.0 °C

$$\begin{aligned} \text{Standard deviation5} &= \sqrt{\frac{1}{(2-1)} [(19.9 - 19.895)^2 + (19.89 - 19.895)^2]} \\ &= 0.00005 \end{aligned}$$

4.3 Uncertainty Evaluation

4.3.1 Uncertainty Due To Repeatability of the Experiment

For determining the uncertainty contribution due to repeatability of experiment, we shall utilize the experiment results obtained for draft UUT calibration. Choose the row having the highest deviation between the MSU value and the UUT value. We calculate the standard deviation by using the formula.

$$S(x_k) = \sqrt{\frac{1}{(n-1)} \sum_{j=1}^k (x_k - \bar{x})^2} \quad (4)$$

The results are collated from the data in Table 4.1. We choose the worst case standard deviation. The u we are looking for the experimental standard deviation of the mean $s(\bar{x})$. This $s(\bar{x})$ is the estimation of the spread of the distribution of the means. We use a sample size n=2 the formula for standard deviation of mean is;

$$S(x) = \frac{S(X_k)}{\sqrt{2}} = \frac{0.00405}{\sqrt{2}} = 0.00286378$$

$$u_1 = \underline{0.00286378 \text{ kPa}} \text{ with a degree of freedom } \nu_1 = \underline{2-1=1}$$

4.3.2 Uncertainty Contribution Due To MSU Error

The MSU used in this calibration is the Model: MT220, Digital Manometer Standard. For the 700kPa range the accuracy specification for this instrument provided by the manufacturer is the following:

$$\pm (0.01\% \text{ of reading} + 0.005\% \text{ range}) \quad (5)$$

For a maximum reading of 200 V and a range of 700 V. Hence the error in MSU = $\pm ((0.0001 \times 200) + (0.00005 \times 700))$ V. Therefore the maximum error = $a \equiv \underline{0.055 \text{ kPa}}$

The uncertainty contribution due to MSU error is defined as u_2 and is given by $a/\sqrt{2} = 0.055/\sqrt{2} = \underline{0.0388909 \text{ kPa}}$.

The degree of freedom γ_2 for this assumed to be ∞ since the manufacturer is expected to provide the error data transfer a large number of tests.

$$u_2 = \underline{0.0388909 \text{ kPa}} \text{ and } \gamma_2 = \infty$$

4.3.3 Uncertainty Due To UUT Resolution/MSU resolution

For type B uncertainty, we can decide on resolution of MSU or resolution of UUT. Generally, if the UUT is analog, we will use the resolution of the MSU. If the UUT is digital, we will use the resolution of digital UUT. The resolution of the UUT Model EJX110A is using METHOD 1.

Considering the worst case scenario the maximum resolution of EJX110A is 0.06 kPa. The uncertainty u_3 is calculated as

$$u_3 = 0.06/\sqrt{2} = \underline{0.042426 \text{ kPa}} \quad (6)$$

We can consider the degree of freedom as ∞

$$u_3 = \underline{0.042426 \text{ kPa}} \text{ and } \gamma_3 = \infty$$

4.3.4 Combined Standard Uncertainty, u_c

The combined standard uncertainty u_c is determined from individual uncertainties u_1 , u_2 and u_3 by following formula;

$$\begin{aligned} u_c &= \sqrt{(u_1^2 + u_2^2 + u_3^2)} \\ &= \sqrt{(0.00286378^2 + 0.0388909^2 + 0.042426^2)} \\ &= 0.0576255 \end{aligned} \quad (7)$$

The effective degree of freedom ν_e is given by,

$$\nu_e = \frac{u_c^4}{\frac{u_1^4}{\nu_1}} \quad (8)$$

$$\nu_e = 163945.7714 = 163946$$

The total uncertainty at any confident level is determined using Student t-distribution. The coverage factor k is determined from students table. Referring to Appendix A for value $\nu = 163946 \geq 100$ and 95.45% confident interval $k = 2.00$

The confident limits are obtained by formula;

$$u = u_c \cdot k \quad (9)$$

$$\begin{aligned} U &= (0.057625) (2.00) \\ &= \pm \underline{0.11525 \text{ kPa}} \end{aligned}$$

The confident limits in a measurement are determined by the use of calibration techniques together with statistical principles.

4.4 RESULT

4.4.1 Result from plotting graph

Figure 4.1 shows the MSU value ($^{\circ}\text{C}$) vs. Actual UUT output (mA). This graph was plotted by using the five values of the 1st Actual UUT output (mA) with the five values of the MSU value ($^{\circ}\text{C}$). The graph was directly proportional.

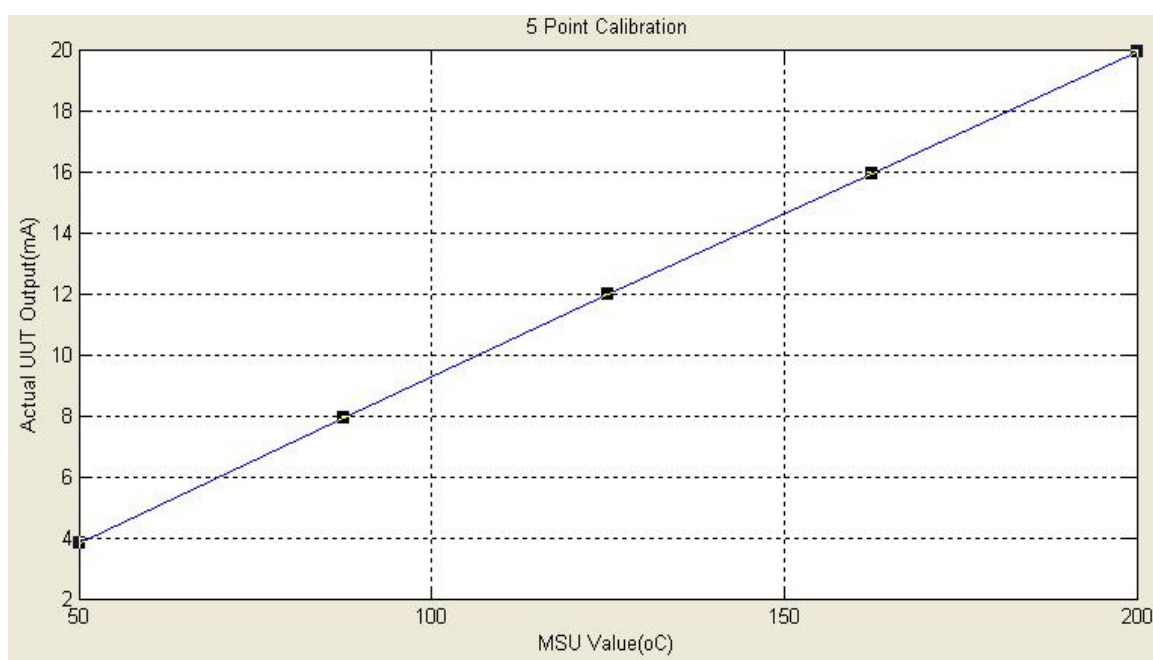


Figure 4.1: Graph of MSU value ($^{\circ}\text{C}$) vs. Actual UUT output (mA)

Figure 4.2 shows the MSU value ($^{\circ}\text{C}$) vs. Output error (%). This graph was plotted by using the five values of the output error (%) with the five values of the MSU value ($^{\circ}\text{C}$). The graph was plotted in smooth curve.

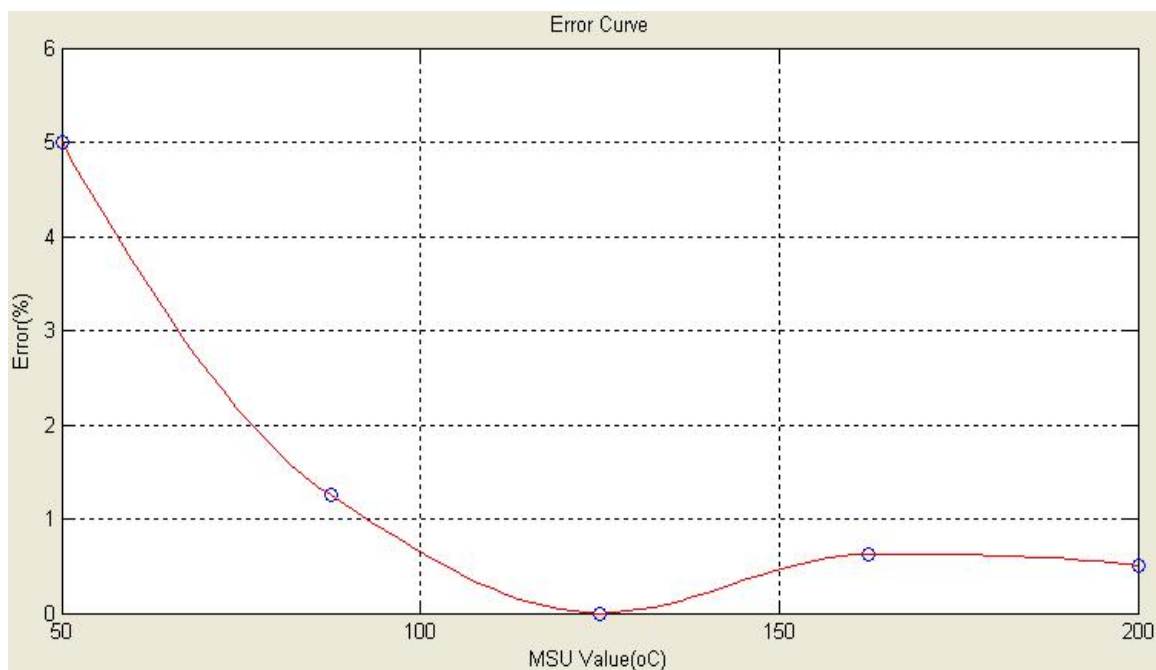


Figure 4.2: Graph of MSU value ($^{\circ}\text{C}$) vs. Output error (%)

4.4.2 GUI Using Matlab 7.0

The Graphical User Interface (GUI) is used as input to give instruction which position has been selected. This design is focused to be a tool that user friendly. The software was developed using Matlab application.

Figure 4.3 shows the design of starting software for this experiment. It called as Home. In this Home, there are five buttons that are Open Automatic, Open Manual, Abstract, Credit and Exit. The “Open Automatic” button used to open the Automatic section. The Automatic section is used when the user wanted to operate the system by automatic function. The “Open Manual” button used to open the Manual section that has the manual function. The “Abstract” button is used to open the abstract of this experiment. It summarized the operation of this system and the equipment that used in this experiment. The “Exit” button is used to exit this system.

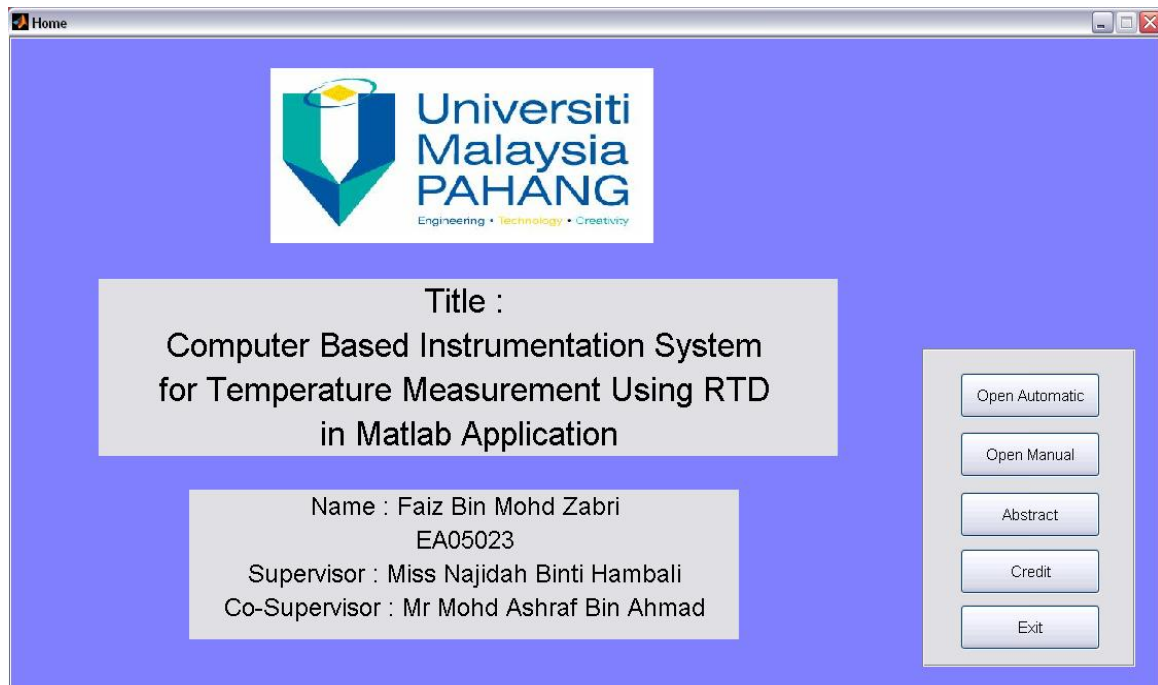


Figure 4.3: The starting software

For the Figure 4.4 shows the Automatic section. It used when the user wanted to collect the data from the instrument by interface with DAQ card. For example, when the temperature of RTD at 50°C. The user clicks the “Ok” button to get data from the instrument. The data that we get in this experiment is the voltage value. The user clicks the “Convert to Current” button to convert that value into the current. After this, the “Output Error” button is click to find the output error and can plot the graph after click the “Graph MSU” button and the “Graph Error” button. The graph also can be saving after click the “Save Image” button. To find the average value and the standard deviation, the user can click the “Open Manual” button to transfer the data into the Manual section.

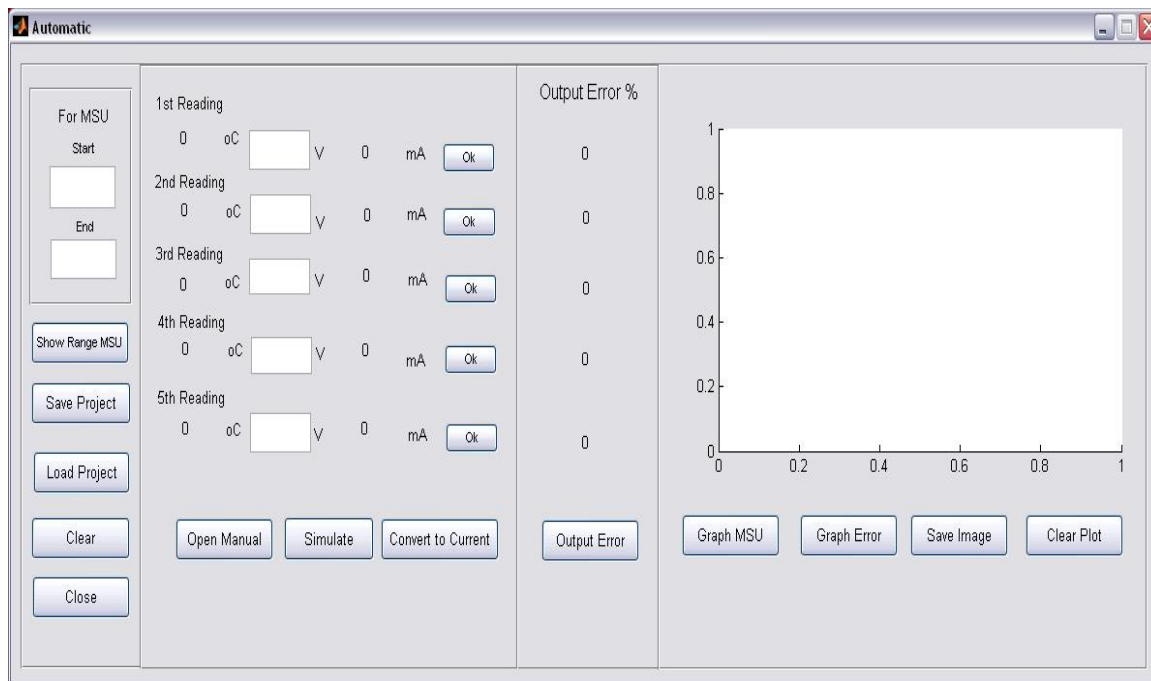


Figure 4.4: The Automatic section

Figure 4.5 shows the Manual section. This section used when the user wanted to analysis this experiment by manually. The user needs to enter the value of data in these forms. Then, the user clicks the “Calculate” button to find the Output Error (%), the Average (mA) and the standard deviation. After that, the user can plot the graph for the graph of MSU value ($^{\circ}\text{C}$) vs. Actual UUT output (mA) and the graph of MSU value ($^{\circ}\text{C}$) vs. Output error (%) by click the “Graph Output” button and the “Graph Error” button.

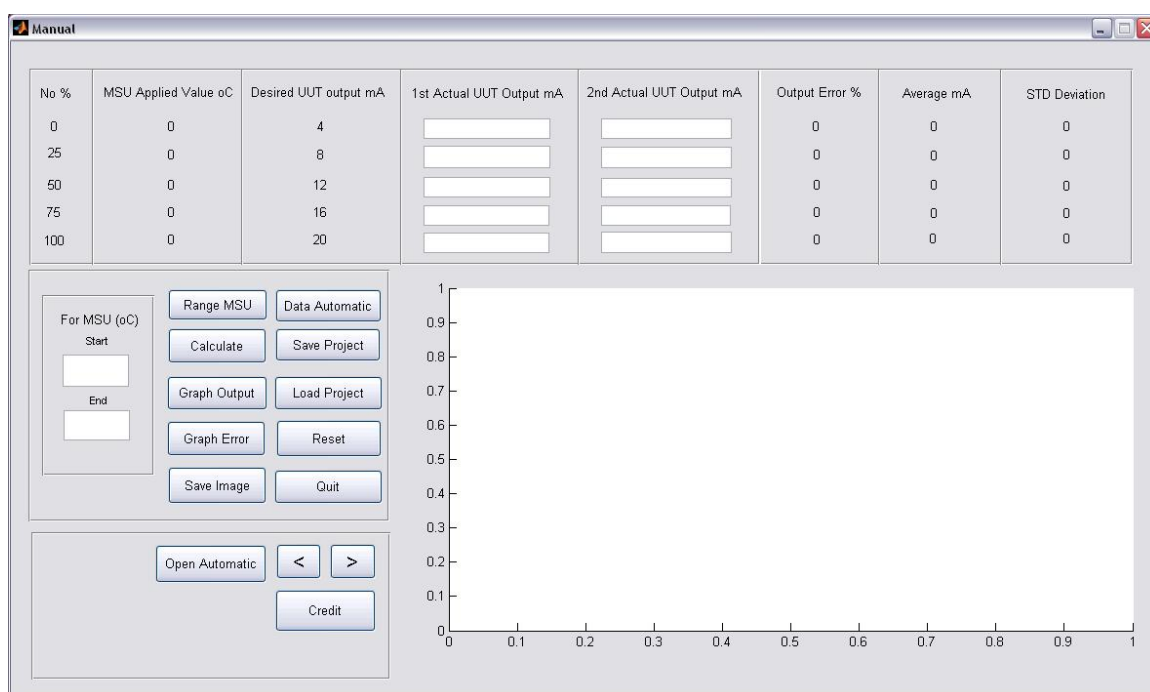


Figure 4.5: The Manual section

4.4.3 The Operation of system

The user entered the values of temperature at lower range and the higher range in the box “For MSU ($^{\circ}\text{C}$)” and clicked the “Range MSU” button. The MSU Applied Value ($^{\circ}\text{C}$) showed the range of temperatures. Figure 4.6 shows the range MSU.

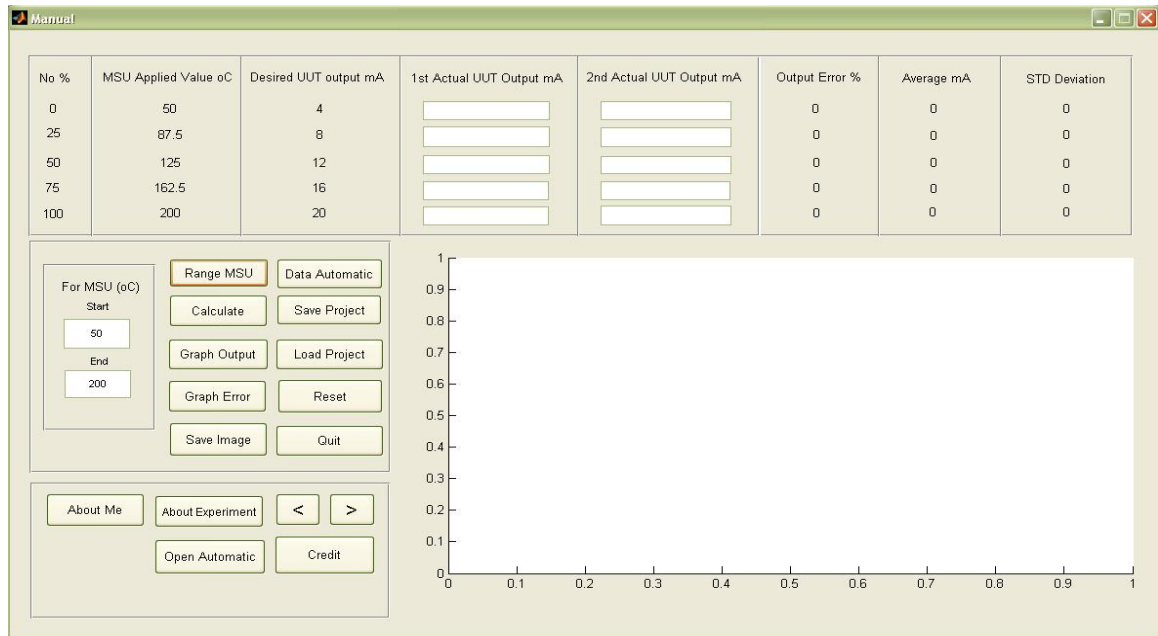


Figure 4.6: To show the range MSU

The user enters the values of 1st Actual UUT output and clicked the “Calculate” button to find output error %. Figure 4.7 shows the insert of 1st Actual UUT output and to find output error.

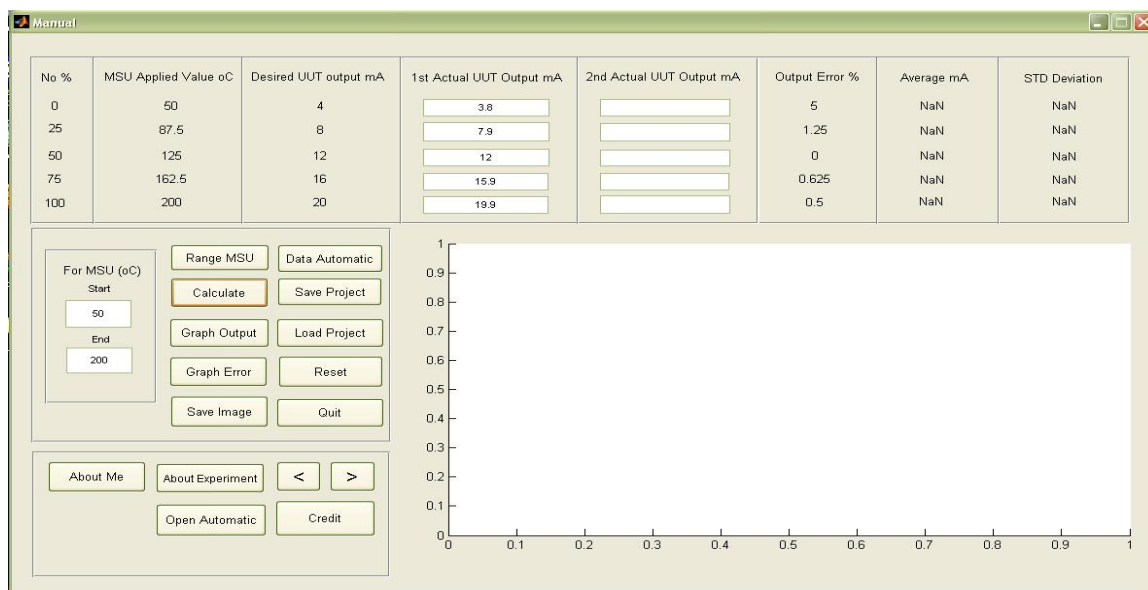


Figure 4.7: Insert value of 1st Actual UUT output and find output error

The user clicks the “Graph Output” button to plot graph 1st Actual UUT output (mA). The graph shows the MSU value (⁰C) vs. Actual UUT output (mA). Figure 4.8 shows Plotting Graph output.

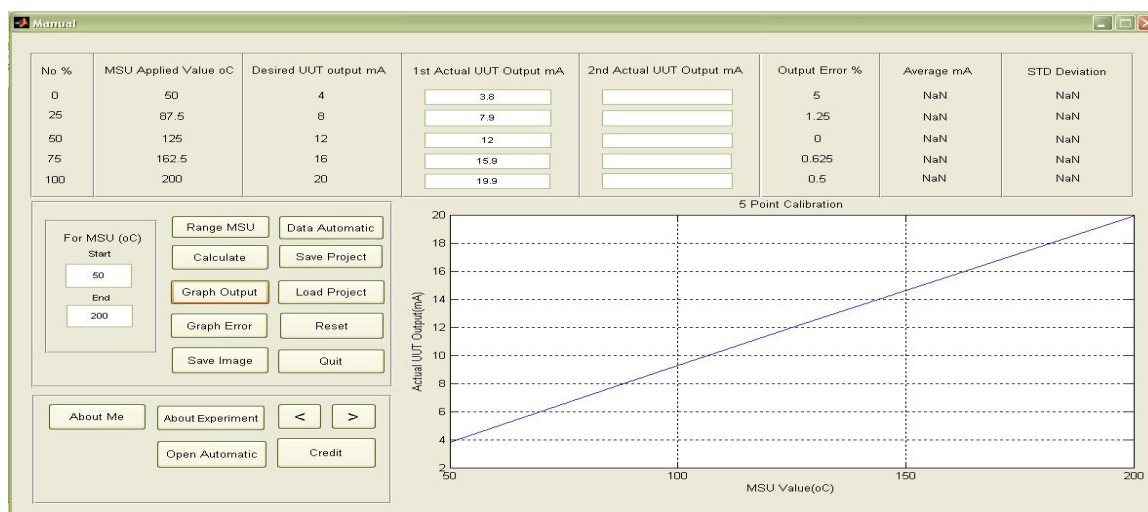


Figure 4.8: Plotting Graph output

The user clicks the “Graph Error” button to plot graph for Output Error %. The graph shows the graph of MSU value ($^{\circ}\text{C}$) vs. Output error (%). Figure 4.9 shows Plotting Graph output.

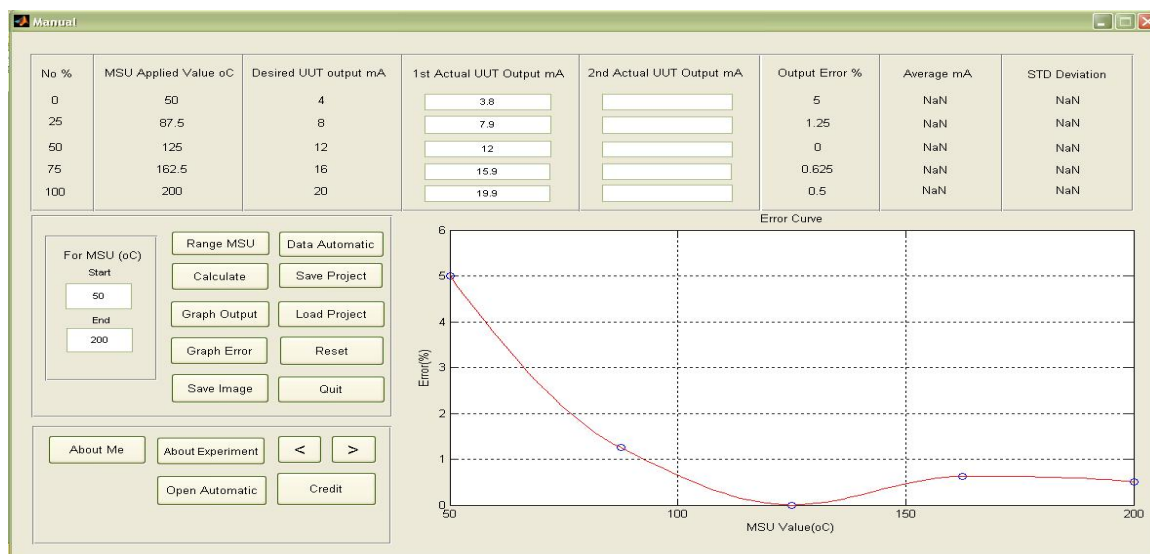


Figure 4.9: Plotting Graph output error

The user inserts the 2nd Actual UUT Output and click the “Calculate” button again to find the Average (mA) and Standard Deviation for this analysis. Figure 4.10 shows to calculate average and standard deviation.

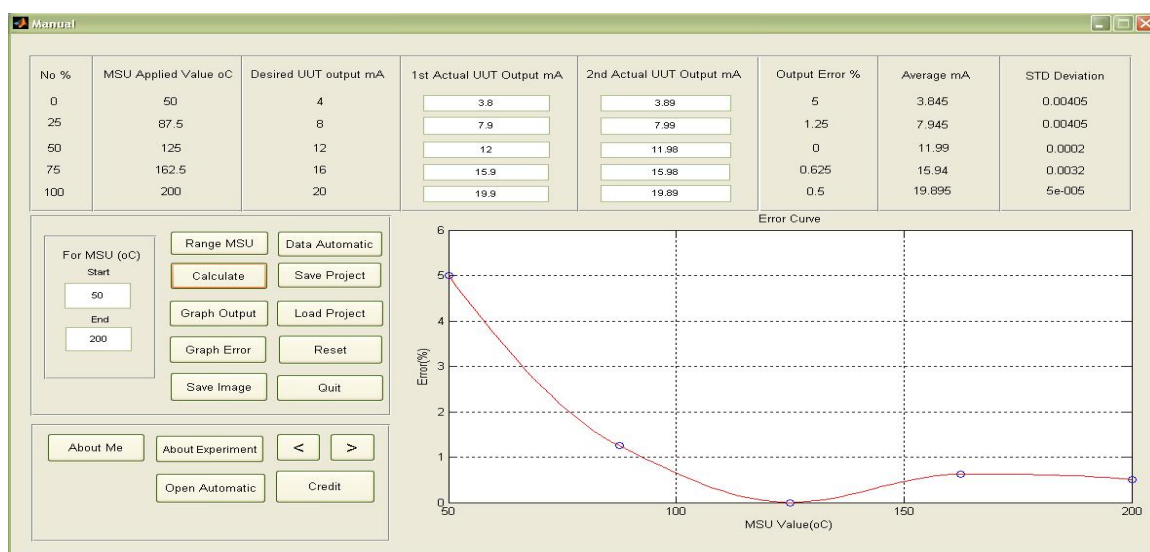


Figure 4.10: To calculate average and standard deviation

The user click the Arrow button [>] for calculate the uncertainties of this analysis. The panel shows the uncertainty panel. The user can enter the value to find the result of the uncertainty. Figure 4.11 shows the uncertainty panel.

| No % | MSU Applied Value oC | Desired UUT output mA | 1st Actual UUT Output mA | 2nd Actual UUT Output mA | Output Error % | Average mA | STD Deviation |
|------|----------------------|-----------------------|--------------------------|--------------------------|----------------|------------|---------------|
| 0 | 50 | 4 | 3.8 | 3.89 | 5 | 3.845 | 0.00405 |
| 25 | 87.5 | 8 | 7.9 | 7.99 | 1.25 | 7.945 | 0.00405 |
| 50 | 125 | 12 | 12 | 11.98 | 0 | 11.99 | 0.0002 |
| 75 | 162.5 | 16 | 15.9 | 15.98 | 0.625 | 15.94 | 0.0032 |
| 100 | 200 | 20 | 19.9 | 19.89 | 0.5 | 19.895 | 5e-005 |

For MSU (oC)
Start: 50
End: 200

Buttons: Range MSU, Data Automatic, Calculate, Save Project, Graph Output, Load Project, Graph Error, Reset, Save Image, Quit, About Me, About Experiment, Open Automatic, Credit, <, >

1) Uncertainty Due To Repeatability of The Experiment
Insert the worst case standard deviation
u1 = 0, DOF = 0
Buttons: Info, Calculate, Clear

2) Uncertainty Contribution Due To MSU Error
(0.01% of [] + 0.005% []) V
reading: 0, range: []
u2 = 0, DOF = 0
Buttons: Info, Calculate, Clear

3) Uncertainty Due To UUT Resolution
Insert the resolution of Recorder
[]
u3 = 0, DOF = 0
Buttons: Info, Calculate, Clear

4) Combined Standard Uncertainty, uc
Insert u1, u2 and u3
u1 = [], u2 = [], u3 = []
uc = 0
Effective degrees of freedom, Ve = 0
Buttons: Info, Calculate, Clear

The coverage factor k is determined from t-distribution. Refer to Appendix
k = []
Buttons: Info, Calculate, Clear

u = uc.k (kPa)
U = 0
Buttons: Calculate, Clear

Appendix button

Figure 4.11: Uncertainty panel

The result get after the user inserts the data to find the uncertainty result. If the user get problem for this panel, the user can click at the “info” button to find the solution, there is some information that can use in this calculation. Figure 4.12 shows the calculation uncertainty.

| No % | MSU Applied Value oC | Desired UUT output mA | 1st Actual UUT Output mA | 2nd Actual UUT Output mA | Output Error % | Average mA | STD Deviation |
|------|----------------------|-----------------------|--------------------------|--------------------------|----------------|------------|---------------|
| 0 | 50 | 4 | 3.8 | 3.89 | 5 | 3.845 | 0.00405 |
| 25 | 87.5 | 8 | 7.9 | 7.99 | 1.25 | 7.945 | 0.00405 |
| 50 | 125 | 12 | 12 | 11.98 | 0 | 11.99 | 0.0002 |
| 75 | 162.5 | 16 | 15.9 | 15.98 | 0.625 | 15.94 | 0.0032 |
| 100 | 200 | 20 | 19.9 | 19.89 | 0.5 | 19.895 | 5e-005 |

For MSU (oC)
Start: 50
End: 200

Buttons: Range MSU, Data Automatic, Calculate, Save Project, Graph Output, Load Project, Graph Error, Reset, Save Image, Quit, About Me, About Experiment, Open Automatic, Credit

1) Uncertainty Due To Repeatability of The Experiment
Insert the worst case standard deviation
u1 = 0.00286378
DOF = 1

2) Uncertainty Contribution Due To MSU Error
(0.01% of 200 + 0.005% 700) V
u2 = 0.0388909
DOF = Inf

3) Uncertainty Due To UUT Resolution
Insert the resolution of Recorder
u3 = 0.0707107
DOF = Inf

4) Combined Standard Uncertainty, uc
Insert u1, u2 and u3
uc = 0.0807509
Effective degrees of freedom, Ve = 632165
The coverage factor k is determined from t-distribution. Refer to Appendix
k = 1.96
u = uc.k (kPa) = 0.158272

Figure 4.12: Calculate uncertainty

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The development software using Matlab application has been presented in this project. The system for this experiment is work done and it can interface with the Advantech PCL-1710HG properly.

For the basic, the user can identify more about the function of each instrument that use for this analysis and also know how to find the reading of temperature using RTD. Otherwise, the user can interface the system with the instrument using DAQ card, the Advantech PCL-1710HG.

The system was developing for the educational purpose. So, the user can use this system to do analysis in the lab. It also suitable uses for the all computer that installed the Matlab software.

This system was developed in two way functions. There are in automatic function and the manual function. For the automatic function, the user can interface the system with DAQ card to collect the data from the instrument. If the user have problem with DAQ card, the user can use the manual function. The user need to enter the value that get from the analysis and then the system will operated to find the result of this analysis.

5.2 Recommendations

This system recommended that the future development should be considering for two things which are the instrument part and the software part, GUI Matlab. For this system, it successful operated. Therefore, there are some add on enhancement for improve the system. The suggestions that can apply in this system are:

- The software should be create creatively and more attractive ways. This system will operate smoothly and working properly without any problem related to instrument part.
- This system will develop for more efficiency and able to work with others hardware. It can interface with the other DAQ card if the Advantech PCL-1710HG have problem.
- The Graphical User Interface can be improved by adding many options such as it can connect with the internet to upload the data in the website. The user can get the data from the website to do the calculation at other places.

REFERENCE

- [1] http://en.wikipedia.org/wiki/Resistance_temperature_detector

- [2] What is a Resistance Temperature Detector (RTD)?
<http://www.omega.com/rtd.html>

- [3] About Temperature Sensors
<http://www.temperatures.com/rtds.html>

- [4] Application Note: RTD, Thermocouple or Thermistor?
<http://www.microdaq.com/accessories/choosing.php>

- [5] Continuous Resistance Temperature Detector Calibration Using Johnson Noise Thermometry, September 2004
http://www.ornl.gov/sci/engineering_science_technology/msd/Personnel/cbritton/clb_papers/S47.pdf

APPENDIX A

T-distribution Curve Table

| Degree of Freedom v | Fraction p in percent | | | | | |
|------------------------|-----------------------|-------|-------|-------|-------|--------|
| | 68.27* | 90.00 | 95.00 | 95.45 | 99.00 | 99.73* |
| 1 | 1.84 | 6.31 | 12.71 | 13.97 | 63.66 | 235.8 |
| 2 | 1.32 | 2.92 | 4.30 | 4.53 | 9.92 | 19.21 |
| 3 | 1.20 | 2.35 | 3.18 | 3.31 | 5.84 | 9.92 |
| 4 | 1.14 | 2.13 | 2.78 | 2.87 | 4.60 | 6.62 |
| 5 | 1.11 | 2.02 | 2.57 | 2.65 | 4.03 | 5.51 |
| 6 | 1.09 | 1.94 | 2.45 | 2.52 | 3.71 | 4.90 |
| 7 | 1.08 | 1.89 | 2.36 | 2.43 | 3.50 | 4.53 |
| 8 | 1.07 | 1.86 | 2.31 | 2.37 | 3.36 | 4.28 |
| 9 | 1.06 | 1.83 | 2.26 | 2.32 | 3.25 | 4.09 |
| 10 | 1.05 | 1.81 | 2.23 | 2.28 | 3.17 | 3.96 |
| | | | | | | |
| 11 | 1.05 | 1.80 | 2.20 | 2.25 | 3.11 | 3.85 |
| 12 | 1.04 | 1.78 | 2.18 | 2.23 | 3.05 | 3.76 |
| 13 | 1.04 | 1.77 | 2.16 | 2.21 | 3.01 | 3.69 |
| 14 | 1.04 | 1.76 | 2.14 | 2.20 | 2.98 | 3.64 |
| 15 | 1.03 | 1.75 | 2.13 | 2.18 | 2.95 | 3.59 |
| | | | | | | |
| 16 | 1.03 | 1.75 | 2.12 | 2.17 | 2.92 | 3.54 |
| 17 | 1.03 | 1.74 | 2.11 | 2.16 | 2.90 | 3.51 |
| 18 | 1.03 | 1.73 | 2.10 | 2.15 | 2.88 | 3.48 |
| 19 | 1.03 | 1.73 | 2.09 | 2.14 | 2.86 | 3.45 |
| 20 | 1.03 | 1.72 | 2.09 | 2.13 | 2.85 | 3.42 |
| | | | | | | |
| 25 | 1.02 | 1.71 | 2.06 | 2.11 | 2.79 | 3.33 |
| 30 | 1.02 | 1.70 | 2.04 | 2.09 | 2.75 | 3.27 |
| 35 | 1.01 | 1.70 | 2.03 | 2.07 | 2.72 | 3.23 |
| 40 | 1.01 | 1.68 | 2.02 | 2.06 | 2.70 | 3.20 |
| 45 | 1.01 | 1.68 | 2.01 | 2.06 | 2.69 | 3.18 |
| | | | | | | |
| 50 | 1.01 | 1.68 | 2.01 | 2.05 | 2.68 | 3.16 |
| 100 | 1.005 | 1.660 | 1.984 | 2.025 | 2.262 | 3.077 |
| | 1.000 | 1.645 | 1.960 | 2.000 | 2.576 | 3.000 |

*For a quality Z described by a normal distribution with expectation μ_z and standard deviation σ , the interval $v_z \pm k\sigma$ encompasses $p = 68.27, 95.45$ AND 99.73 percent of the distribution for $k = 1, 2$ and 3 respectively.

APPENDIX B**PCI-1710 HG Datasheet**

APPENDIX C

Coding Program

For Automatic Section

```
function varargout = Automatic(varargin)
% AUTOMATIC M-file for Automatic.fig
%   AUTOMATIC, by itself, creates a new AUTOMATIC or raises the existing
%   singleton*.
%
%   H = AUTOMATIC returns the handle to a new AUTOMATIC or the handle to
%   the existing singleton*.
%
%   AUTOMATIC('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in AUTOMATIC.M with the given input
%   arguments.
%
%   AUTOMATIC('Property','Value',...) creates a new AUTOMATIC or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before Automatic_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to Automatic_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help Automatic

% Last Modified by GUIDE v2.5 18-Sep-2008 01:35:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
                  'gui_Singleton', gui_Singleton, ...
                  'gui_OpeningFcn', @Automatic_OpeningFcn, ...
```

```

        'gui_OutputFcn', @Automatic_OutputFcn, ...
        'gui_LayoutFcn', [] , ...
        'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if narginout
    [varargout{1:narginout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Automatic is made visible.
function Automatic_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Automatic (see VARARGIN)

% Choose default command line output for Automatic
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Automatic wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Automatic_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in start2.
function start2_Callback(hObject, eventdata, handles)
% hObject    handle to start2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```



```

set(handles.stop2,'visible','on')
set(handles.start2,'visible','off')

% --- Executes on button press in stop2.
function stop2_Callback(hObject, eventdata, handles)
% hObject    handle to stop2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.stop2,'visible','off')
set(handles.start2,'visible','on')

% --- Executes on button press in close2.
function close2_Callback(hObject, eventdata, handles)
% hObject    handle to close2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close

function start3_Callback(hObject, eventdata, handles)
% hObject    handle to start3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of start3 as text
%        str2double(get(hObject,'String')) returns contents of start3 as a double

% --- Executes during object creation, after setting all properties.
function start3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to start3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in show2.
function show2_Callback(hObject, eventdata, handles)
% hObject    handle to show2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get user input from GUI

```

```

st3 = str2double(get(handles.start3,'String'));
ed3 = str2double(get(handles.end3,'String'));

% Calculate data
v3 = (ed3-st3)/4;
Aa2 = st3+(0*v3);
set(handles.msu12,'string',Aa2);
Bb2 = st3+(1*v3);
set(handles.msu22,'string',Bb2);
Cc2 = st3+(2*v3);
set(handles.msu32,'string',Cc2);
Dd2 = st3+(3*v3);
set(handles.msu42,'string',Dd2);
Ee2 = st3+(4*v3);
set(handles.msu52,'string',Ee2);

function data12_Callback(hObject, eventdata, handles)
% hObject   handle to data12 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data12 as text
%        str2double(get(hObject,'String')) returns contents of data12 as a double

% --- Executes during object creation, after setting all properties.
function data12_CreateFcn(hObject, eventdata, handles)
% hObject   handle to data12 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data22_Callback(hObject, eventdata, handles)
% hObject   handle to data22 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data22 as text
%        str2double(get(hObject,'String')) returns contents of data22 as a double

```

```

% --- Executes during object creation, after setting all properties.
function data22_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data22 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data32_Callback(hObject, eventdata, handles)
% hObject    handle to data32 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data32 as text
%        str2double(get(hObject,'String')) returns contents of data32 as a double

% --- Executes during object creation, after setting all properties.
function data32_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data32 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data42_Callback(hObject, eventdata, handles)
% hObject    handle to data42 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data42 as text
%        str2double(get(hObject,'String')) returns contents of data42 as a double

% --- Executes during object creation, after setting all properties.
function data42_CreateFcn(hObject, eventdata, handles)

```

```

% hObject    handle to data42 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in ok1.
function ok1_Callback(hObject, eventdata, handles)
% hObject    handle to ok1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%clear the workspace
clear

%make the handles structures available to the main workspace
h =gcf;
handles = guidata(h);

%To display data from workspace to GUI
load('simout.mat')
ok1=simout(10)
set(handles.data12,'String',ok1);

% --- Executes on button press in ok2.
function ok2_Callback(hObject, eventdata, handles)
% hObject    handle to ok2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%clear the workspace
clear

%make the handles structures available to the main workspace
h =gcf;
handles = guidata(h);

%To display data from workspace to GUI
load('simout.mat')
ok2=simout(10)

```

```

set(handles.data22,'String',ok2);

% --- Executes on button press in ok3.
function ok3_Callback(hObject, eventdata, handles)
% hObject    handle to ok3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%clear the workspace
clear

%make the handles structures available to the main workspace
h =gcf;
handles = guidata(h);

%To display data from workspace to GUI
load('simout.mat')
ok3=simout(10)
set(handles.data32,'String',ok3);

% --- Executes on button press in ok4.
function ok4_Callback(hObject, eventdata, handles)
% hObject    handle to ok4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%clear the workspace
clear

%make the handles structures available to the main workspace
h =gcf;
handles = guidata(h);

%To display data from workspace to GUI
load('simout.mat')
ok4=simout(10)
set(handles.data42,'String',ok4);

% --- Executes on button press in ok5.
function ok5_Callback(hObject, eventdata, handles)
% hObject    handle to ok5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%clear the workspace
clear

```

```

%make the handles structures available to the main workspace
h =gcf;
handles = guidata(h);

%To display data from workspace to GUI
load('simout.mat')
ok5=simout(10)
set(handles.data52,'String',ok5);

% --- Executes on button press in clear2.
function clear2_Callback(hObject, eventdata, handles)
% hObject    handle to clear2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.start3,'string','');
set(handles.end3,'string','');

set(handles.msu12,'string','0');
set(handles.msu22,'string','0');
set(handles.msu32,'string','0');
set(handles.msu42,'string','0');
set(handles.msu52,'string','0');

% --- Executes on button press in error2.
function error2_Callback(hObject, eventdata, handles)
% hObject    handle to error2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Data reading from automatic

value1d = str2double(get(handles.data12,'String'));
d1 = (value1d/250);
value2d = str2double(get(handles.data22,'String'));
d2 = (value2d/250);
value3d = str2double(get(handles.data32,'String'));
d3 = (value3d/250);
value4d = str2double(get(handles.data42,'String'));
d4 = (value4d/250);
value5d = str2double(get(handles.data52,'String'));
d5 = (value5d/250);

% Calculate percentage error2
total12 = ((0.004-d1)/(0.004))*100;
set(handles.error12,'string',total12);
total22 = ((0.008-d2)/(0.008))*100;

```

```

set(handles.error22,'string',total22);
total32 = ((0.012-d3)/(0.012))*100;
set(handles.error32,'string',total32);
total42 = ((0.016-d4)/(0.016))*100;
set(handles.error42,'string',total42);
total52 = ((0.020-d5)/(0.020))*100;
set(handles.error52,'string',total52);

% --- Executes on button press in graphmsu.
function graphmsu_Callback(hObject, eventdata, handles)
% hObject    handle to graphmsu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

axes(handles.graf2)
% Recall data from UUT
a2 = str2double(get(handles.data13,'String'));
b2 = str2double(get(handles.data23,'String'));
c2 = str2double(get(handles.data33,'String'));
d2 = str2double(get(handles.data43,'String'));
e2 = str2double(get(handles.data53,'String'));

st3 = str2double(get(handles.start3,'String'));
ed3 = str2double(get(handles.end3,'String'));

% Calculate data
v3 = (ed3-st3)/4;
Aa2 = st3+(0*v3);
set(handles.msu12,'string',Aa2);
Bb2 = st3+(1*v3);
set(handles.msu22,'string',Bb2);
Cc2 = st3+(2*v3);
set(handles.msu32,'string',Cc2);
Dd2 = st3+(3*v3);
set(handles.msu42,'string',Dd2);
Ee2 = st3+(4*v3);
set(handles.msu52,'string',Ee2);

% Create frequency plot

X = [Aa2 Bb2 Cc2 Dd2 Ee2];
Y = [a2 b2 c2 d2 e2]
plot(X,Y)
xlabel('MSU Value(oC)')
ylabel('Actual UUT Output(mA)')
title('5 Point Calibration')

```

grid on

```
% --- Executes on button press in Grapherror.
function Grapherror_Callback(hObject, eventdata, handles)
% hObject    handle to Grapherror (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

axes(handles.graf2)
% Recall data from UUT
a2 = str2double(get(handles.data13,'String'));
b2 = str2double(get(handles.data23,'String'));
c2 = str2double(get(handles.data33,'String'));
d2 = str2double(get(handles.data43,'String'));
e2 = str2double(get(handles.data53,'String'));

st3 = str2double(get(handles.start3,'String'));
ed3 = str2double(get(handles.end3,'String'));

% Calculate data
v3 = (ed3-st3)/4;
Aa2 = st3+(0*v3);
set(handles.msu12,'string',Aa2);
Bb2 = st3+(1*v3);
set(handles.msu22,'string',Bb2);
Cc2 = st3+(2*v3);
set(handles.msu32,'string',Cc2);
Dd2 = st3+(3*v3);
set(handles.msu42,'string',Dd2);
Ee2 = st3+(4*v3);
set(handles.msu52,'string',Ee2);

% Calculate data
A2 = ((4.0-a2)/4.0)*100;
B2 = ((8.0-b2)/8.0)*100;
C2 = ((12.0-c2)/12.0)*100;
D2 = ((16.0-d2)/16.0)*100;
E2 = ((20.0-e2)/20.0)*100;

% Create frequency plot

% Points in each interval
divider = 20;
X = [Aa2 Bb2 Cc2 Dd2 Ee2];
Y = [A2 B2 C2 D2 E2]
```



```

%-----
%do the magic
%-----
m = size(X);
n = m(2);
o = n - 1;
xi = [];
for tel = 1:o
    interval = (X(tel + 1) - X(tel))/(divider);
    xintervals = [X(tel):interval:X(tel + 1)];
    xi = [xi xintervals];
end
%-----
%plot the magic
%-----
plottools off
yi = interp1(X,Y,xi,'cubic');
plot(X,Y,'o');
hold on;
plot(xi,yi,'r');
hold off;

xlabel('MSU Value(oC)')
ylabel('Error(%)')
title('Error Curve')
grid on

% --- Executes on button press in save2.
function save2_Callback(hObject, eventdata, handles)
% hObject    handle to save2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

savePlotWithinGUI(handles.graf2)

% --- Executes on button press in clear3.
function clear3_Callback(hObject, eventdata, handles)
% hObject    handle to clear3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

cla(handles.graf2,'reset')

% --- Executes during object creation, after setting all properties.
function mula_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mula (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% --- Executes on button press in sim.
function sim_Callback(hObject, eventdata, handles)
% hObject handle to sim (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

%simulate the system
set_param(gcs,'Simulationmode','external')
set_param(gcs,'SimulationCommand','connect')
set_param(gcs,'SimulationCommand','start')
set_param(gcs,'SimulationCommand','stop')

function data33_Callback(hObject, eventdata, handles)
% hObject handle to data33 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data33 as text
% str2double(get(hObject,'String')) returns contents of data33 as a double

% --- Executes during object creation, after setting all properties.
function data33_CreateFcn(hObject, eventdata, handles)
% hObject handle to data33 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data43_Callback(hObject, eventdata, handles)
% hObject handle to data43 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data43 as text
% str2double(get(hObject,'String')) returns contents of data43 as a double

% --- Executes during object creation, after setting all properties.

```

```

function data43_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data43 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in convert.
function convert_Callback(hObject, eventdata, handles)
% hObject    handle to convert (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Collect data(voltage)

value1d = str2double(get(handles.data12,'String'));
value2d = str2double(get(handles.data22,'String'));
value3d = str2double(get(handles.data32,'String'));
value4d = str2double(get(handles.data42,'String'));
value5d = str2double(get(handles.data52,'String'));

% Convert voltage to current

d1 = (value1d/250)*1000;
set(handles.data13,'string',d1);
d2 = (value2d/250)*1000;
set(handles.data23,'string',d2);
d3 = (value3d/250)*1000;
set(handles.data33,'string',d3);
d4 = (value4d/250)*1000;
set(handles.data43,'string',d4);
d5 = (value5d/250)*1000;
set(handles.data53,'string',d5);

function data13_Callback(hObject, eventdata, handles)
% hObject    handle to data13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data13 as text

```

```

%      str2double(get(hObject,'String')) returns contents of data13 as a double

%stores the figure handle of Manual's GUI here
ManualFigureHandle = Manual;

%stores the GUI data from Manual's GUI here
%now we can access any of the data from Manual's GUI!!!
ManualData = guidata(ManualFigureHandle);

%store the input text from Daniel's GUI
%into the variable daniel_input
%daniel_input = get(danielData.editText_Daniel,'String');

%input text from Daniel's GUI
set(handles.data1,'String',data13);

% --- Executes on button press in save4.
function save4_Callback(hObject, eventdata, handles)
% hObject    handle to save4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%allow the user to specify where to save the settings file
[filename,pathname] = uiputfile('project','Save your GUI settings');

if pathname == 0 %if the user pressed cancelled, then we exit this callback
    return
end
%construct the path name of the save location
saveDataName = fullfile(pathname,filename);

%saves the gui data
hgsave(saveDataName);

% --- Executes on button press in load4.
function load4_Callback(hObject, eventdata, handles)
% hObject    handle to load4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%allow the user to choose which settings to load3
[filename, pathname] = uigetfile('*.fig', 'Choose the GUI settings file to load');

%construct the path name of the file to be loaded
loadDataName = fullfile(pathname,filename);

```

```
%this is the gui that will be closed once we load3 the new settings
theCurrentGUI = gcf;
```

```
%load3 the settings, which creates a new gui
hgload(loadDataName);
```

```
%closes the old gui
close(theCurrentGUI);
```

For Manual Section

```
function varargout = Manual(varargin)
% MANUAL M-file for Manual.fig
%   MANUAL, by itself, creates a new MANUAL or raises the existing
%   singleton*.
%
%   H = MANUAL returns the handle to a new MANUAL or the handle to
%   the existing singleton*.
%
%   MANUAL('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in MANUAL.M with the given input arguments.
%
%   MANUAL('Property','Value',...) creates a new MANUAL or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before Manual_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to Manual_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help Manual

% Last Modified by GUIDE v2.5 18-Sep-2008 01:33:55
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @Manual_OpeningFcn, ...
    'gui_OutputFcn', @Manual_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);
```

```

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if narginout
    [varargout{1:narginout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Manual is made visible.
function Manual_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Manual (see VARARGIN)

% Choose default command line output for Manual
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Manual wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Manual_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function data1_Callback(hObject, eventdata, handles)
% hObject    handle to data1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data1 as text
%        str2double(get(hObject,'String')) returns contents of data1 as a double
data1 = str2double(get(hObject,'String'));

```

```

if isnan(data1)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function data1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data2_Callback(hObject, eventdata, handles)
% hObject    handle to data2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data2 as text
%    str2double(get(hObject,'String')) returns contents of data2 as a double
data2 = str2double(get(hObject, 'String'));
if isnan(data2)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function data2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

function data3_Callback(hObject, eventdata, handles)
% hObject    handle to data3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data3 as text
%        str2double(get(hObject,'String')) returns contents of data3 as a double
data3 = str2double(get(hObject, 'String'));
if isnan(data3)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function data3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data4_Callback(hObject, eventdata, handles)
% hObject    handle to data4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data4 as text
%        str2double(get(hObject,'String')) returns contents of data4 as a double
data4 = str2double(get(hObject, 'String'));
if isnan(data4)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function data4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```



```
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function data5_Callback(hObject, eventdata, handles)
% hObject handle to data5 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of data5 as text
% str2double(get(hObject,'String')) returns contents of data5 as a double
data5 = str2double(get(hObject,'String'));
if isnan(data5)
    set(hObject,'String','');
    errordlg(' Input Must Be A Number !!','Error');
end
```

```
% --- Executes during object creation, after setting all properties.
function data5_CreateFcn(hObject, eventdata, handles)
% hObject handle to data5 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function start1_Callback(hObject, eventdata, handles)
% hObject handle to start1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of start1 as text
% str2double(get(hObject,'String')) returns contents of start1 as a double
```

```
start1 = str2double(get(hObject,'String'));
if isnan(start1)
    set(hObject,'String','');
```

```

    errordlg('    Value Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function start1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to start1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function end1_Callback(hObject, eventdata, handles)
% hObject    handle to end1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of end1 as text
%    str2double(get(hObject,'String')) returns contents of end1 as a double

end1 = str2double(get(hObject, 'String'));
if isnan(end1)
    set(hObject, 'String', '');
    errordlg('    Value Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function end1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to end1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in show1.

```

```

function show1_Callback(hObject, eventdata, handles)
% hObject    handle to show1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get user input from GUI
f = str2double(get(handles.start1,'String'));
g = str2double(get(handles.end1,'String'));

% Calculate data

h = (g-f)/4;
Aa = f+(0*h);
set(handles.msu1,'string',Aa);
Bb = f+(1*h);
set(handles.msu2,'string',Bb);
Cc = f+(2*h);
set(handles.msu3,'string',Cc);
Dd = f+(3*h);
set(handles.msu4,'string',Dd);
Ee = f+(4*h);
set(handles.msu5,'string',Ee);

% --- Executes on button press in plot1.
function plot1_Callback(hObject, eventdata, handles)
% hObject    handle to plot1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get user input from GUI
a = str2double(get(handles.data1,'String'));
b = str2double(get(handles.data2,'String'));
c = str2double(get(handles.data3,'String'));
d = str2double(get(handles.data4,'String'));
e = str2double(get(handles.data5,'String'));

% Recall data from MSU
f = str2double(get(handles.start1,'String'));
g = str2double(get(handles.end1,'String'));

h = (g-f)/4;
Aa = f+(0*h);
set(handles.msu1,'string',Aa);
Bb = f+(1*h);
set(handles.msu2,'string',Bb);
Cc = f+(2*h);

```

```

set(handles.msu3,'string',Cc);
Dd = f+(3*h);
set(handles.msu4,'string',Dd);
Ee = f+(4*h);
set(handles.msu5,'string',Ee);

% Calculate data

A = ((4.0-a)/4.0)*100;
B = ((8.0-b)/8.0)*100;
C = ((12.0-c)/12.0)*100;
D = ((16.0-d)/16.0)*100;
E = ((20.0-e)/20.0)*100;

% Create frequency plot

%Points in each interval
divider = 20;
X = [Aa Bb Cc Dd Ee];
Y = [A B C D E]
%-----
%do the magic
%-----
m = size(X);
n = m(2);
o = n - 1;
xi = [];
for tel = 1:o
    interval = (X(tel + 1) - X(tel))/(divider);
    xintervals = [X(tel):interval:X(tel + 1)];
    xi = [xi xintervals];
end
%-----
%plot the magic
%-----
plottools off
yi = interp1(X,Y,xi,'cubic');
plot(X,Y,'o');
hold on;
plot(xi,yi,'r');
hold off;

xlabel('MSU Value(oC)')
ylabel('Error(%)')
title('Error Curve')
grid on

```

```

% --- Executes on button press in call.
function call_Callback(hObject, eventdata, handles)
% hObject    handle to call (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get user actual UUT from GUI

value1a = str2double(get(handles.uut1,'String'));
value1b = str2double(get(handles.data1,'String'));
value2a = str2double(get(handles.uut2,'String'));
value2b = str2double(get(handles.data2,'String'));
value3a = str2double(get(handles.uut3,'String'));
value3b = str2double(get(handles.data3,'String'));
value4a = str2double(get(handles.uut4,'String'));
value4b = str2double(get(handles.data4,'String'));
value5a = str2double(get(handles.uut5,'String'));
value5b = str2double(get(handles.data5,'String'));

% Calculate percentage error

total1 = ((value1a-value1b)/(value1a))*100;
set(handles.error1,'string',total1);
total2 = ((value2a-value2b)/(value2a))*100;
set(handles.error2,'string',total2);
total3 = ((value3a-value3b)/(value3a))*100;
set(handles.error3,'string',total3);
total4 = ((value4a-value4b)/(value4a))*100;
set(handles.error4,'string',total4);
total5 = ((value5a-value5b)/(value5a))*100;
set(handles.error5,'string',total5);

% Calculate average

value1b = str2double(get(handles.data1,'String'));
value1c = str2double(get(handles.data6,'String'));
total1bc = ((value1b+value1c)/2);
set(handles.ave1,'string',total1bc);
value2b = str2double(get(handles.data2,'String'));
value2c = str2double(get(handles.data7,'String'));
total2bc = ((value2b+value2c)/2);
set(handles.ave2,'string',total2bc);
value3b = str2double(get(handles.data3,'String'));
value3c = str2double(get(handles.data8,'String'));
total3bc = ((value3b+value3c)/2);

```

```

set(handles.ave3,'string',total3bc);
value4b = str2double(get(handles.data4,'String'));
value4c = str2double(get(handles.data9,'String'));
total4bc = ((value4b+value4c)/2);
set(handles.ave4,'string',total4bc);
value5b = str2double(get(handles.data5,'String'));
value5c = str2double(get(handles.data10,'String'));
total5bc = ((value5b+value5c)/2);
set(handles.ave5,'string',total5bc);

%calculate std

value1b = str2double(get(handles.data1,'String'));
value1c = str2double(get(handles.data6,'String'));
total1bc = ((value1b+value1c)/2);
standard1 = (value1b-total1bc)^2+(value1c-total1bc)^2;
set(handles.std1,'string',standard1);

value2b = str2double(get(handles.data2,'String'));
value2c = str2double(get(handles.data7,'String'));
total2bc = ((value2b+value2c)/2);
standard2 = (value2b-total2bc)^2+(value2c-total2bc)^2;
set(handles.std2,'string',standard2);

value3b = str2double(get(handles.data3,'String'));
value3c = str2double(get(handles.data8,'String'));
total3bc = ((value3b+value3c)/2);
standard3 = (value3b-total3bc)^2+(value3c-total3bc)^2;
set(handles.std3,'string',standard3);

value4b = str2double(get(handles.data4,'String'));
value4c = str2double(get(handles.data9,'String'));
total4bc = ((value4b+value4c)/2);
standard4 = (value4b-total4bc)^2+(value4c-total4bc)^2;
set(handles.std4,'string',standard4);

value5b = str2double(get(handles.data5,'String'));
value5c = str2double(get(handles.data10,'String'));
total5bc = ((value5b+value5c)/2);
standard5 = (value5b-total5bc)^2+(value5c-total5bc)^2;
set(handles.std5,'string',standard5);

function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to data2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```
% Hints: get(hObject,'String') returns contents of data2 as text
%      str2double(get(hObject,'String')) returns contents of data2 as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to data3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of data3 as text
%      str2double(get(hObject,'String')) returns contents of data3 as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit13 as text
```

```

%      str2double(get(hObject,'String')) returns contents of edit13 as a double

% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to data5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data5 as text
%      str2double(get(hObject,'String')) returns contents of data5 as a double

% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in qt.
function qt_Callback(hObject, eventdata, handles)
% hObject    handle to qt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close all

% --- Executes on button press in cre1.

```



```

function cre1_Callback(hObject, eventdata, handles)
% hObject    handle to cre1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.txt1,'string','Idea and programming by Faiz EA05023');
pause(5);
set(handles.txt1,'string','');

function data6_Callback(hObject, eventdata, handles)
% hObject    handle to data6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data6 as text
%        str2double(get(hObject,'String')) returns contents of data6 as a double
data6 = str2double(get(hObject, 'String'));
if isnan(data6)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function data6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data8_Callback(hObject, eventdata, handles)
% hObject    handle to data8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data8 as text
%        str2double(get(hObject,'String')) returns contents of data8 as a double
data8 = str2double(get(hObject, 'String'));
if isnan(data8)
    set(hObject, 'String', '');

```

```

    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function data8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data8 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data10_Callback(hObject, eventdata, handles)
% hObject    handle to data10 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data10 as text
%       str2double(get(hObject,'String')) returns contents of data10 as a double
data10 = str2double(get(hObject, 'String'));
if isnan(data10)
    set(hObject, 'String', '');
    errordlg('    Input Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function data10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to data10 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function data7_Callback(hObject, eventdata, handles)
% hObject    handle to data7 (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data7 as text
% str2double(get(hObject,'String')) returns contents of data7 as a double
data7 = str2double(get(hObject,'String'));
if isnan(data7)
    set(hObject,'String','');
    errordlg(' Input Must Be A Number !!','Error');
end

function data9_Callback(hObject, eventdata, handles)
% hObject handle to data9 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of data9 as text
% str2double(get(hObject,'String')) returns contents of data9 as a double
data9 = str2double(get(hObject,'String'));
if isnan(data9)
    set(hObject,'String','');
    errordlg(' Input Must Be A Number !!','Error');
end

% --- Executes on button press in gra1.
function gra1_Callback(hObject, eventdata, handles)
% hObject handle to gra1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Recall data from UUT
a = str2double(get(handles.data1,'String'));
b = str2double(get(handles.data2,'String'));
c = str2double(get(handles.data3,'String'));
d = str2double(get(handles.data4,'String'));
e = str2double(get(handles.data5,'String'));

% Recall data from MSU
f = str2double(get(handles.start1,'String'));
g = str2double(get(handles.end1,'String'));

h = (g-f)/4;
Aa = f+(0*h);
set(handles.msu1,'string',Aa);
Bb = f+(1*h);
set(handles.msu2,'string',Bb);

```

```

Cc = f+(2*h);
set(handles.msu3,'string',Cc);
Dd = f+(3*h);
set(handles.msu4,'string',Dd);
Ee = f+(4*h);
set(handles.msu5,'string',Ee);

% Create frequency plot

X = [Aa Bb Cc Dd Ee];
Y = [a b c d e]
plot(X,Y)
xlabel('MSU Value(oC)')
ylabel('Actual UUT Output(mA)')
title('5 Point Calibration')
grid on

% --- Executes on button press in cal2.
function cal2_Callback(hObject, eventdata, handles)
% hObject    handle to cal2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Calculate u1
wstandard = str2double(get(handles.wstd,'String'));
uncert1 = (wstandard/sqrt(2));
set(handles.u1,'String',uncert1);

%Calculate DOF
dof1 = 2-1;
set(handles.dof1,'String',dof1);

% --- Executes on button press in cle2.
function cle2_Callback(hObject, eventdata, handles)
% hObject    handle to cle2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.wstd,'String','');
set(handles.u1,'String','0');
set(handles.dof1,'String','0');

function read1_Callback(hObject, eventdata, handles)
% hObject    handle to read1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of read1 as text
%       str2double(get(hObject,'String')) returns contents of read1 as a double

read1 = str2double(get(hObject, 'String'));
if isnan(read1)
    set(hObject, 'String', '');
    errordlg('    Value Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function read1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to read1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function range1_Callback(hObject, eventdata, handles)
% hObject    handle to range1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of range1 as text
%       str2double(get(hObject,'String')) returns contents of range1 as a double

range1 = str2double(get(hObject, 'String'));
if isnan(range1)
    set(hObject, 'String', '');
    errordlg('    Value Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function range1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to range1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.

```

```

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in calcu2.
function calcu2_Callback(hObject, eventdata, handles)
% hObject    handle to calcu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in cleu2.
function cleu2_Callback(hObject, eventdata, handles)
% hObject    handle to cleu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in back.
function back_Callback(hObject, eventdata, handles)
% hObject    handle to back (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.graf,'visible','on')
set(handles.uncpanel,'visible','off')

% --- Executes on button press in forward.
function forward_Callback(hObject, eventdata, handles)
% hObject    handle to forward (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.graf,'visible','off')
set(handles.uncpanel,'visible','on')

function wmr_Callback(hObject, eventdata, handles)
% hObject    handle to wmr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of wmr as text
%        str2double(get(hObject,'String')) returns contents of wmr as a double

wmr = str2double(get(hObject, 'String'));
if isnan(wmr)

```

```

    set(hObject, 'String', '');
    errordlg('    Value Must Be A Number !!','Error');
end

function un1_Callback(hObject, eventdata, handles)
% hObject    handle to un1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of un1 as text
%        str2double(get(hObject,'String')) returns contents of un1 as a double

un1 = str2double(get(hObject, 'String'));
if isnan(un1)
    set(hObject, 'String', '');
    errordlg('    Value Must Be A Number !!','Error');
end

function un2_Callback(hObject, eventdata, handles)
% hObject    handle to un2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of un2 as text
%        str2double(get(hObject,'String')) returns contents of un2 as a double

un2 = str2double(get(hObject, 'String'));
if isnan(un2)
    set(hObject, 'String', '');
    errordlg('    Value Must Be A Number !!','Error');
end

function un3_Callback(hObject, eventdata, handles)
% hObject    handle to un3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of un3 as text
%        str2double(get(hObject,'String')) returns contents of un3 as a double

un3 = str2double(get(hObject, 'String'));
if isnan(un3)
    set(hObject, 'String', '');
    errordlg('    Value Must Be A Number !!','Error');
end

% --- Executes on button press in cal3.

```

```

function cal3_Callback(hObject, eventdata, handles)
% hObject    handle to cal3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Calculate u2
reading = str2double(get(handles.read1,'String'));
range = str2double(get(handles.range1,'String'));
maxerror = ((0.0001*reading)+(0.00005*range));
uncert2 = (maxerror/sqrt(2));
set(handles.u2,'String',uncert2);

%Calculate DOF
dof2 = 2/0;
set(handles.dof2,'String',dof2);

% --- Executes on button press in cle3.
function cle3_Callback(hObject, eventdata, handles)
% hObject    handle to cle3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.read1,'String','');
set(handles.range1,'String','');
set(handles.u2,'String','0');
set(handles.dof2,'String','0');

% --- Executes on button press in cal4.
function cal4_Callback(hObject, eventdata, handles)
% hObject    handle to cal4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Calculate u3
wmaxres = str2double(get(handles.wmr,'String'));
uncert3 = (wmaxres/sqrt(2));
set(handles.u3,'String',uncert3);

%Calculate DOF
dof3 = 2/0;
set(handles.dof3,'String',dof3);

% --- Executes on button press in cle4.
function cle4_Callback(hObject, eventdata, handles)
% hObject    handle to cle4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```



```

% handles    structure with handles and user data (see GUIDATA)

set(handles.wmr,'String','');
set(handles.u3,'String','0');
set(handles.dof3,'String','0');

% --- Executes on button press in cal5.
function cal5_Callback(hObject, eventdata, handles)
% hObject    handle to cal5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Calculate uc
unc1 = str2double(get(handles.un1,'String'));
unc2 = str2double(get(handles.un2,'String'));
unc3 = str2double(get(handles.un3,'String'));
cmb = sqrt((unc1^2)+(unc2^2)+(unc3^2));
set(handles.uc,'String',cmb);

%Calculate Ve

Ve = (cmb^4)/((unc1^4)/(2-1));
set(handles.ve,'String',Ve);

% --- Executes on button press in cle5.
function cle5_Callback(hObject, eventdata, handles)
% hObject    handle to cle5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.un1,'String','');
set(handles.un2,'String','');
set(handles.un3,'String','');
set(handles.uc,'String','0');
set(handles.ve,'String','0');

function K_Callback(hObject, eventdata, handles)
% hObject    handle to K (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of K as text
%        str2double(get(hObject,'String')) returns contents of K as a double

K = str2double(get(hObject, 'String'));
if isnan(K)

```

```

    set(hObject, 'String', '');
    errordlg('    Value Must Be A Number !!','Error');
end

% --- Executes during object creation, after setting all properties.
function K_CreateFcn(hObject, eventdata, handles)
% hObject    handle to K (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in cle6.
function cle6_Callback(hObject, eventdata, handles)
% hObject    handle to cle6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.K,'String','');
set(handles.u,'String','0');

% --- Executes on button press in cal6.
function cal6_Callback(hObject, eventdata, handles)
% hObject    handle to cal6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Calculate U
unc1 = str2double(get(handles.un1,'String'));
unc2 = str2double(get(handles.un2,'String'));
unc3 = str2double(get(handles.un3,'String'));
cmb = sqrt((unc1^2)+(unc2^2)+(unc3^2));
k = str2double(get(handles.K,'String'));
U = cmb*k;
set(handles.u,'String',U);

% --- Executes on button press in save.
function save_Callback(hObject, eventdata, handles)
% hObject    handle to save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```

```

% handles    structure with handles and user data (see GUIDATA)

savePlotWithinGUI(handles.graf)

% --- Executes on button press in experiment.
function experiment_Callback(hObject, eventdata, handles)
% hObject    handle to experiment (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in Save3.
function Save3_Callback(hObject, eventdata, handles)
% hObject    handle to Save3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%allow the user to specify where to save the settings file
[filename,pathname] = uiputfile('project','Save your GUI settings');

if pathname == 0 %if the user pressed cancelled, then we exit this callback
    return
end
%construct the path name of the save location
saveDataName = fullfile(pathname,filename);

%saves the gui data
hgsave(saveDataName);

% --- Executes on button press in Load3.
function Load3_Callback(hObject, eventdata, handles)
% hObject    handle to Load3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%allow the user to choose which settings to load3
[filename, pathname] = uigetfile('*.fig', 'Choose the GUI settings file to load');

%construct the path name of the file to be loaded
loadDataName = fullfile(pathname,filename);

%this is the gui that will be closed once we load3 the new settings
theCurrentGUI = gcf;

%load3 the settings, which creates a new gui
hgload(loadDataName);

```

```

%closes the old gui
close(theCurrentGUI);

% --- Executes on button press in dataA.
function dataA_Callback(hObject, eventdata, handles)
% hObject    handle to dataA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%stores the figure handle of Automatic's GUI here
AutomaticFigureHandle = Automatic;

%stores the GUI data from Automatic's GUI here
%now we can access any of the data from Automatic's GUI!!!!
AutomaticData = guidata(AutomaticFigureHandle);

%store the input text from Automatic's GUI
%into the variable Automatic_input
Automatic1_input = get(AutomaticData.data13,'String');
Automatic2_input = get(AutomaticData.data23,'String');
Automatic3_input = get(AutomaticData.data33,'String');
Automatic4_input = get(AutomaticData.data43,'String');
Automatic5_input = get(AutomaticData.data53,'String');
Automatic6_input = get(AutomaticData.start3,'String');
Automatic7_input = get(AutomaticData.end3,'String');

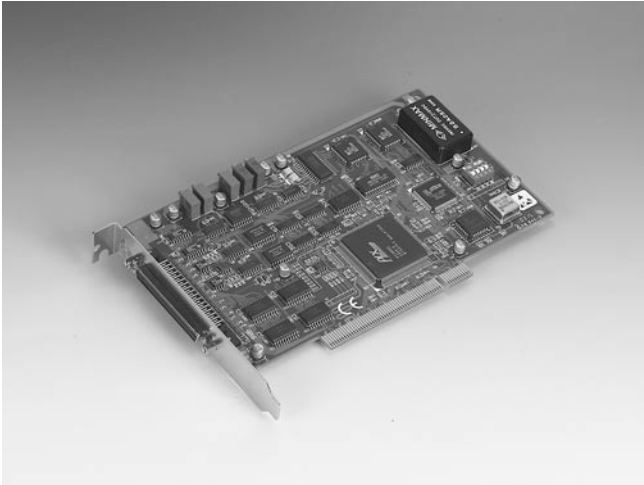
%set the static text on Manual's GUI to match the
%input text from Automatic's GUI
set(handles.data1,'String',Automatic1_input);
set(handles.data2,'String',Automatic2_input);
set(handles.data3,'String',Automatic3_input);
set(handles.data4,'String',Automatic4_input);
set(handles.data5,'String',Automatic5_input);
set(handles.start1,'String',Automatic6_input);
set(handles.end1,'String',Automatic7_input);

```

PCI-1710 PCI-1710HG

100 kS/s, 12-bit, 16-ch PCI Multifunction Card

100 kS/s, 12-bit, 16-ch PCI Multifunction Card with High Gain



Features

- 16 single-ended or 8 differential or a combination of analog inputs
- 12-bit A/D converter, with up to 100 kHz sampling rate
- Programmable gain
- Automatic channel/gain scanning
- Onboard FIFO memory (4096 samples)
- Two 12-bit analog output channels (PCI-1710/1710HG only)
- 16 digital inputs and 16 digital outputs
- Onboard programmable counter
- BoardID™ switch

Introduction

The PCI-1710 Series are multifunction cards for the PCI bus. Their advanced circuit design provides higher quality and more functions, including the five most desired measurement and control functions: 12-bit A/D conversion, D/A conversion, digital input, digital output, and counter/timer.

Specifications

Analog Input

- **Channels** 16 single-ended/ 8 differential (SW programmable)
- **Resolution** 12 bits
- **Max. Sampling Rate*** 100 kS/s
- **FIFO Size** 4096 samples
- **Overvoltage Protection** $\pm 30V_p-p$
- **Input Impedance** 1 G Ω
- **Sampling Modes** Software, onboard programmable pacer, or external
- **Input Range** (V, software programmable)

| PCI-1710/1710L | | | | | |
|--|----------|---------|-----------|------------|-------------|
| Bipolar | ± 10 | ± 5 | ± 2.5 | ± 1.25 | ± 0.625 |
| Unipolar | - | 0 ~ 10 | 0 ~ 5 | 0 ~ 2.5 | 0 ~ 1.25 |
| Accuracy (% of FSR $\pm 1LSB$) | 0.1 | 0.1 | 0.2 | 0.2 | 0.4 |

| PCI-1710HG/1710HGL | | | | | | | | |
|--|----------|---------|---------|-----------|-----------|------------|------------|-------------|
| Bipolar | ± 10 | ± 5 | ± 1 | ± 0.5 | ± 0.1 | ± 0.05 | ± 0.01 | ± 0.005 |
| Unipolar | - | 0 ~ 10 | - | 0 ~ 1 | - | 0 ~ 0.1 | - | 0 ~ 0.01 |
| Accuracy (% of FSR $\pm 1LSB$) | 0.1 | 0.1 | 0.2 | 0.2 | 0.2 | 0.2 | 0.4 | 0.4 |

- **Maximum Sampling Rate** (S/s, depending on PGIA setting time)

| Model | Gain | Max. Sampling Rate |
|--------------------|-----------------|--------------------|
| PCI-1710/1710L | 0.5, 1, 2, 4, 8 | 100 KS/s |
| | 0.5, 1 | 100 KS/s |
| PCI-1710HG/1710HGL | 5, 10 | 35 KS/s |
| | 20, 100 | 7 KS/s |
| | 500, 1000 | 770 S/s |

*Note:

The sampling rate and throughput depends on the computer hardware architecture and software environment. The rates may vary due to programming language, code efficiency, CPU utilization and so on.

Analog Output (PCI-1710/1710HG only)

- **Channels** 2
- **Resolution** 12 bits
- **Output Rate** Static update
- **Output Range** (V, software programmable)

| Internal Reference | Unipolar | 0 ~ +5 V @ -5 V 0 ~ +10 V @ -10 V |
|--------------------|----------|--|
| External Reference | | 0 ~ +x V @ -x V ($-10 \leq x \leq 10$) |

- **Slew Rate** 10 V/ms
- **Driving Capability** 3 mA
- **Operation Mode** Software polling
- **Accuracy** INLE: $\pm 1/2$ LSB, DNLE: $\pm 1/2$ LSB

Digital Input

- **Channels** 16
- **Compatibility** 5 V/TTL
- **Input Voltage** Logic 0: 0.8 V max.
Logic 1: 2.0 V min.

Digital Output

- **Channels** 16
- **Compatibility** 5 V/TTL
- **Output Voltage** Logic 0: 0.4 V max.
Logic 1: 2.4 V min.
- **Output Capability** Sink: 8.0 mA @ 0.8 V
Source: -0.4 mA @ 2.0 V

Pacer/Counter

- **Channels** 1
- **Resolution** 16 bits
- **Compatibility** 5 V/TTL
- **Max. Input Frequency** 1 MHz

Specifications Continued

General

- **Bus Type** PCI V2.2
- **I/O Connector** SCSI-68P female x 1
- **Dimensions (L x H)** 175 x 100 mm (6.9" x 3.9")
- **Power Consumption** Typical: 5 V @ 850 mA
Max: 5 V @ 1.0 A
- **Operating Temperature** 0 ~ 60° C (32 ~ 140° F) (refer to IEC 68-2-1, 2)
- **Storing Temperature** -20 ~ 70° C (-4 ~ 158° F)
- **Storing Humidity** 5 ~ 95% RH non-condensing (refer to IEC 68-2-3)

Ordering Information

- **PCI-1710** 100 kS/s, 12-bit multifunction card
- **PCI-1710L** 100 kS/s, 12-bit multifunction card without AO
- **PCI-1710HG** 100 kS/s, 12-bit high-gain multifunction card
- **PCI-1710HGL** 100 kS/s, 12-bit high-gain multifunction card without AO
- **PCLD-8710** SCSI-68 wiring terminal w/CJC, DIN-rail mount
- **PCL-10168-1** SCSI-68 Shielded Cable, 1 m
- **PCL-10168-2** SCSI-68 Shielded Cable, 2 m
- **ADAM-3968** SCSI-68 wiring terminal, DIN-rail mount

Pin Assignments

| | | | |
|-----------|----|----|-----------|
| AI0 | 68 | 34 | AI1 |
| AI2 | 67 | 33 | AI3 |
| AI4 | 66 | 32 | AI5 |
| AI6 | 65 | 31 | AI7 |
| AI8 | 64 | 30 | AI9 |
| AI10 | 63 | 29 | AI11 |
| AI12 | 62 | 28 | AI13 |
| AI14 | 61 | 27 | AI15 |
| AIGND | 60 | 26 | AIGND |
| *AO0_REF | 59 | 25 | AO1_REF* |
| *AO0_OUT | 58 | 24 | AO1_OUT* |
| AOGND | 57 | 23 | AOGND |
| DI0 | 56 | 22 | DI1 |
| DI2 | 55 | 21 | DI3 |
| DI4 | 54 | 20 | DI5 |
| DI6 | 53 | 19 | DI7 |
| DI8 | 52 | 18 | DI9 |
| DI10 | 51 | 17 | DI11 |
| DI12 | 50 | 16 | DI13 |
| DI14 | 49 | 15 | DI15 |
| DGND | 48 | 14 | DGND |
| DO0 | 47 | 13 | DO1 |
| DO2 | 46 | 12 | DO3 |
| DO4 | 45 | 11 | DO5 |
| DO6 | 44 | 10 | DO7 |
| DO8 | 43 | 9 | DO9 |
| DO10 | 42 | 8 | DO11 |
| DO12 | 41 | 7 | DO13 |
| DO14 | 40 | 6 | DO15 |
| DGND | 39 | 5 | DGND |
| CNT0_CLK | 38 | 4 | PACER_OUT |
| CNT0_OUT | 37 | 3 | TRG_GATE |
| CNT0_GATE | 36 | 2 | EXT_TRG |
| +12V | 35 | 1 | +5V |

*: Pins 23~25 and pins 57~59 are not defined for PCI-1710L/1710HGL