Journal of Engineering and Technology for Industrial Applications

ISSN ONLINE: 2447-0228

ITEGAM-JETIA

Manaus, v.9 n.40, p. 4-9. Mar/Apr, 2023.
DOI: https://doi.org/10.5935/jetia.v9i40.849

RESEARCH ARTICLE OPEN ACCESS

EXPLANATION OF AN INNOVATIVE PROGRAMMING APPROACH TO CREATE A PROGRESSIVE TWO-DIMENSIONAL DATA FILE OF THREE INTER-DEPENDANT VARIABLES IN A FULLY AUTOMATED DATA ACQUISITION SYSTEM

Avijit Das*[1]

[1] Saha Institute of Nuclear Physics, 1/AF, Bidhan Nagar, Kolkata - 700064, West Bengal, India.

[1] http://orcid.org/0000-0001-5754-5245

Email *avijit.das@saha.ac.in

ARTICLE INFO

Article History
Received: March 23th, 2023
Accepted: April 26th, 2023
Published: April 29th, 2023

Keywords:
LabVIEW,
2D array,
Data acquisition,
Automation,
CSV file.

ABSTRACT

This technical document explains a programming model for recording data of a special case in a comma-separated (.csv) file. Generally, in any data acquisition system storing of data increments in one direction; that is in rows. In this article a different programming concept has been illustrated for an automated application to store data in two dimensions; that not only expands in rows but also in columns during the progress of next all successive data acquisition loops for the same set of parameters with a new updated value of the third variable. The effectiveness of the program me has been verified based on a practical proto type test made by monitoring and recording several sets of resultant values of DC current at different values of applied DC voltage and had performed these tests consecutively at different values of temperature. All these acquired data sets were saved in a single comma separated (.csv) data file. The full process can be completed in a single test run and most importantly, without any intervention from a human being. By following the logic and method demonstrated, a similar application could be developed also by other text-based programming languages like Java or Python.

I. INTRODUCTION

This article describes a fully automated data acquisition programming process in LabVIEW to implement a non-conventional idea of creating a comma-separated data file. Here an uncommon programming method demonstrates how several sequential measurement cycles of a set of variables against variation of a third variable can be saved in a data file without any human intervention. In general, in the case of any data acquisition system in a scientific or engineering laboratory, we normally measure and collect data from one or more variables, that vary against another common and periodically changing variables, like time, voltage or current, temperature etc. Such data collection cycles continue to add on rows of data one after another, which may be of two or more columns depending on the number of parameters under assessment. But sometimes a specific situation demands creating and saving sets of data by monitoring of the first variable against periodic variation of the second variable at different values of a third variable. One such a very common example is the output characteristic of an NPN transistor at common-emitter configuration, where individual sets of data are collected by monitoring of collector current (IC) against linear variation of collector-emitter voltage (VCE) at different values of base current (IB).

Conventionally, this can easily be done by performing several tests runs at different values of the third variable (like, IB) and manually combining all the results in an excel sheet. But in a completely automated data acquisition process suggested in this article without any attention from a human being, the automatic creation of a new column on the right side of the latest one is done. This process requires an additional programming trick, and that is the goal of writing this article. A sample example has been shown in Table 1 to illustrate my programming idea in an

apprehensible way. Where the initial set of measurements of current (first variable) in a load with linear variation of voltage (second variable) records at some temperature T1. The same acquisition cycles repeat with different values of temperature, such as T2, T3, T4, etc and all the sets of data is being saved in a single .csv file. In this document, LabVIEW graphical programming language (version 8.0 on Windows XP OS) has been used to explain the procedure [1]. But the same methodology can be followed for other popular high-level programming languages like C++, Java or Python.

Table 1: an example to show how a .csv data file creates different sets of data of variable-2 (I) against variable-1 (V) at different values of the third variable (T).

		Loops of several cycles						
		Loop 1	***Loop 2***	***Loop 3***	***Loop 4***	***Loops 5 to 8***	***Loop 9***	***Loop 10***
		T_1	T_2	T_3	T_4	T_5 to T_8	T_9	T_{10}
Acquisition Cycles	V_1	I_1	I_{11}	I_{21}	I_{31}	.....	I_{81}	I_{91}
	V_2	I_2	I_{12}	I_{22}	I_{32}	.....	I_{82}	I_{92}
	V_3	I_3	I_{13}	I_{23}	I_{33}	.....	I_{83}	I_{93}
	V_4 to V_9	.....	.....	.....	.....	.....	.....	.....
	V_{10}	I_{10}	I_{20}	I_{30}	I_{40}	.....	I_{90}	I_{100}
		Handles by Sub_Prog_1.vi	Handles by Sub_Prog_2.vi					

Source: Author, (2023).

II. PROGRAMME DESIGN METHOD

The programming model has been designed to handle three variables that are interlinked with each other. A single data acquisition cycle consists of monitoring the values of variable-2 (Current, abbreviated as I) against stepwise increment of variable-1 (Voltage, abbreviated as V). Several such sets of acquisition cycles had been performed at different values of the third variable (Temperature, abbreviated as T) and designated as acquisition loops.

Several graphical programme files had been developed (listed in Table-2) to demonstrate the process. Among these, the main file is Main_Prog_GUI.vi which acts as the user interface where data file name and other parameters can be entered in designated text boxes as well as acquired data is being displayed (Figure 1). Graphical interface for Sub_Prog_1.vi and Sub_Prog_2.vi was not compulsory to build, but I had made it to check the variables visually to understand the progress. An in-built *'File Dialog'* VI (virtual instrument used in LabVIEW programming) that displays a dialog box, where the user can specify a file name with a directory to save the data file with an *Open/Create/Replace File* option. The total programme architecture has been shown graphically in Figure 2. Different parts of the main or parent programme file have been marked with distinguished part numbers to help the readers to identify the segment of the programme under discussion.

In this programme, a *'Flat Sequence Structure'* (marked as FSS-1 in Figure 2) executes statements sequentially (P2, P3, and P4) one after another within a main programme '*While Loop'*. In the first sub-diagram or frame (P2) of this FSS-1, another internal *'Flat Sequence Structure'* (FSS-2) executes eleven times within a *'For Loop'* (For Loop 1) and creates eleven numbers of rows in the first data file. In its first frame (P2.1A), voltage increases to its next value and calls Sub_Prog_3.vi. Sub_Prog_3 sends a set of SCPI commands to the Keithley meter via GPIB interface method to source this voltage and measure the corresponding current.

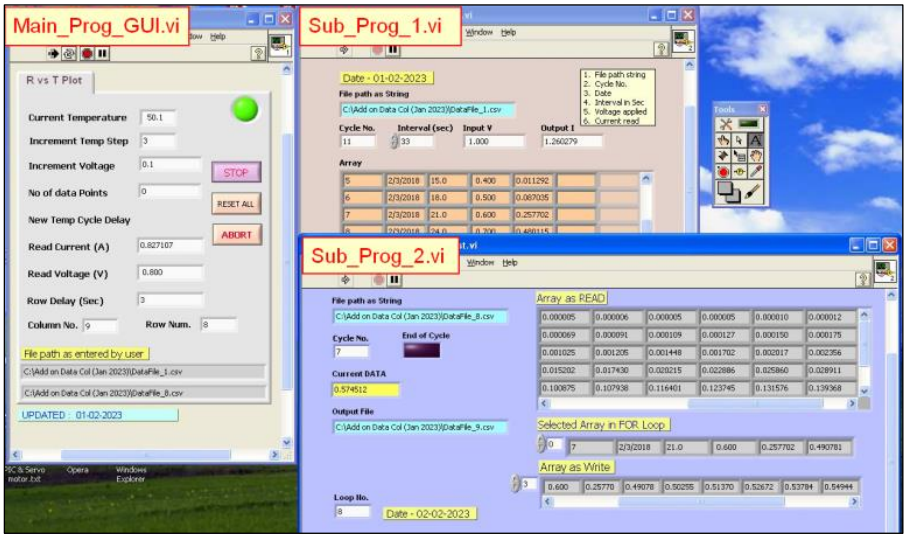


Figure 1: Screenshot of three graphical interface files among them, Main_Prog_GUI.vi control the parameters, and the other two display the data while acquisition is in progress.

Source: Author, (2023).

These parameters are then sent to Sub_Prog_1 to store eleven rows of data with five columns. It saves the first spreadsheet data file (DataFile_1.csv) using an in-built VI file called '*Write To Spreadsheet File.vi*' with serial numbers, date, elapsed time in second, the voltage applied (second variable) and noted current (first variable) - all are separated by comma (Figure 3). All these parameters were in string format and created rows of data with '*Build Array*' in-built function [2]. Every sequential call of this VI file (Sub_Prog_1.vi) from the programme-part P2.1A during each cycle within the For Loop-1 adds new rows of data with five columns. A cycle delay was added in frame-2 (P2.1B) of FSS-2 as desired by the user. Once the first round of data collection is made at an initial value of the third variable, this VI file (Sub_Prog_1.vi) is never invoked any further.

Table 2: List of project files and the functions performed by them.

Project file names mentioned in this article	Function / Purpose of the programme code
Main_Prog_GUI.vi	* Creates the main graphical user interface (GUI). It is likely to be modified according to the need. * It shows all necessary parameters in text boxes and clickable buttons.
Sub_Prog_1.vi	* Creates first spreadsheet file named as DataFile_1.csv
Sub_Prog_2.vi	* Creates a new data file and manipulates the old file name to generate a new data file name. * Reads latest spreadsheet file row by row and adds new data at the end of each row with a comma added afterward, * And saves all data sets in the newly created spreadsheet file.
Sub_Prog_3.vi	* Communicates with Keithley 2450 via GPIB Port. * Sources some voltage and reads output current from the DUT.
Sub_Prog_4.vi	* Read the current temperature from the Autonics PID controller
Sub_Prog_5.vi	* Send new Set Point (SP) to the Autonics PID controller.
Sub_Prog_6.vi	* Set Auto-Tune On or Off for the Autonics PID controller.9

Source: Author, (2023).

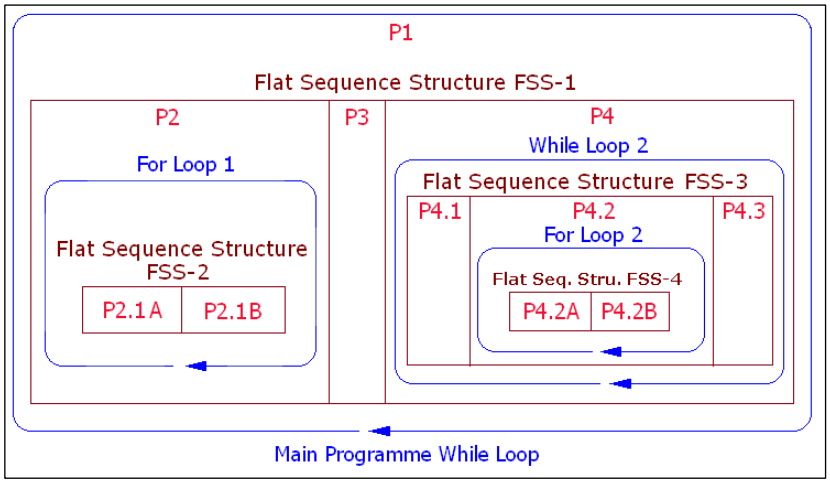
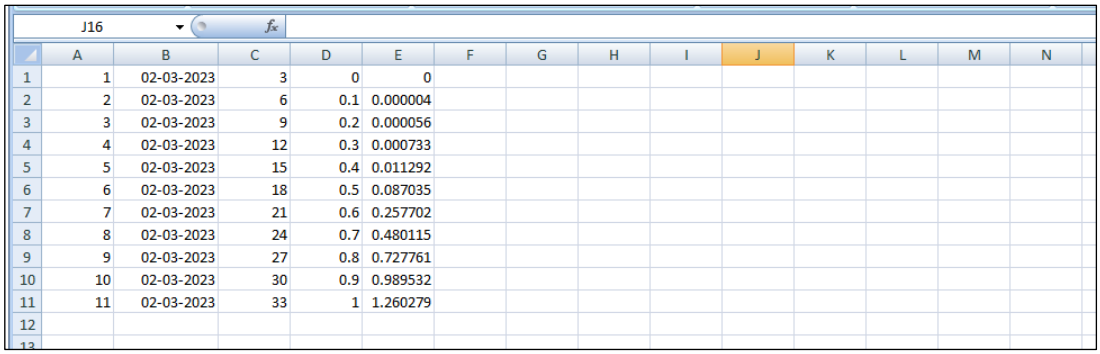


Figure 2: Block diagram of the programme Main_Prog_GUI.vi that revokes other two sub-programme files from '*For Loop 1*' and '*For Loop 2*'.

Source: Author, (2023).



J16

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1	02-03-2023	3	0	0									
2	2	02-03-2023	6	0.1	0.000004									
3	3	02-03-2023	9	0.2	0.000056									
4	4	02-03-2023	12	0.3	0.000733									
5	5	02-03-2023	15	0.4	0.011292									
6	6	02-03-2023	18	0.5	0.087035									
7	7	02-03-2023	21	0.6	0.257702									
8	8	02-03-2023	24	0.7	0.480115									
9	9	02-03-2023	27	0.8	0.727761									
10	10	02-03-2023	30	0.9	0.989532									
11	11	02-03-2023	33	1	1.260279									
12														

Figure 3: Screenshot of the data file (DataFile_1.csv) displayed within the Excel sheet after completing the first round of data acquisition in Loop-1 by the programme Sub_Prog_1.vi.

Source: Author, (2023).

Time is incremented at every 3 seconds and voltage is incremented by 0.1v at each step starting from zero to one volt. Column E is showing the resultant current.

In the next frame P3, the main programme calls another two sub-programmes Sub_Prog_4.vi and Sub_Prog_5.vi which communicate with the PID temperature controller from a USB port using the RS-485 method. A USB-to-RS485 converter was used for this purpose. Sub_Prog_4.vi reads the current temperature (PV) and Sub_Prog_5.vi writes a new set point (SP) of temperature to the controller. Instruction codes within these two sequences part P2 and P3 is executed only once and never called any further till the end of the programme.

In the third frame (P4) of FSS-1, programme performs the most important and little complex processes. In this part of the programme P4, another *'Flat Sequence Structure'* (FSS-3 in Figure 2) executes nine times within a conditional *'While Loop'* (While Loop 2). This number is restricted exclusively for my programme only and has been discussed later. All programme sequences inside this recursive *'While Loop'* repeats depending upon the number of data acquisition loops wanted by the programmer. Each such acquisition loop creates a new data file with an added data column.

Within the first frame (P4.1) of the Flat Sequence Structure (FSS-3) Sub_Prog_3.vi is called to reset the source meter voltage to zero and prepare it to start a new measurement cycle from the initial stage. Sub_Prog_4.vi is also called from this part to read the current process temperature.

In the second part (P4.2) of FSS-3 programme reads the first data file name and manipulates it to generate a new file name depending upon the serial number of the data column it is going to create. It reads the initial data file name string (i.e. DataFile_1.csv) and to determine a new file name, it subtracts the last five characters from the file name string using '*String Subset*' function.

With this character subtraction, the file name string becomes, "DataFile_"

It takes the current column number and adds the remaining string with this column number and .csv extension ("N.csv", where N is the current column or plot number).

While recording data in the second column, the file name becomes, "DataFile_" + "2.csv" = "DataFile_2.csv".

A *'For Loop'* (named as For Loop 2 in Figure 2) executes eleven times. Inside this For Loop, another set of programme instructions executes sequentially inside FSS-4. Inside the first frame (P4.2A) of FSS-4, Sub_Prog_3.vi is called to send commands to the Keithley sourcemeter to measure the load current against the application of some amount of source voltage incremented from its previous value. In the next frame, (P4.2B) Sub_prog_2.vi is called, which performs the addition of recent data (that is load current in our case) to a new and fresh data column on the right side of the previous column. Figure 4 can explain the process clearly. The programme reads each row as a 1D Array from the previous and latest spreadsheet file and adds the recent value of variable-1 (i.e. Current, I_{31}) as an array element using the in-built function '*Insert into Array*'. The sub-programme file 'Sub_Prog_2.vi' is called eleven times from the 'For Loop-2' to read all the rows sequentially as a 1-D array and add the newly acquired data at its end as a new array element. Within part P4 the 'While Loop-2' iterates maximum of nine times to create total ten (1+9 = 10) data collection loops. The reason for this number restriction has been discussed in the section 'Conclusion'. The output 1D array from the '*Insert into Array*' function writes into the new spreadsheet file that has been generated with a new file name (i.e. DataFile_4.csv in Figure 4) using the in-built VI file '*Write To Spreadsheet File.vi*'.

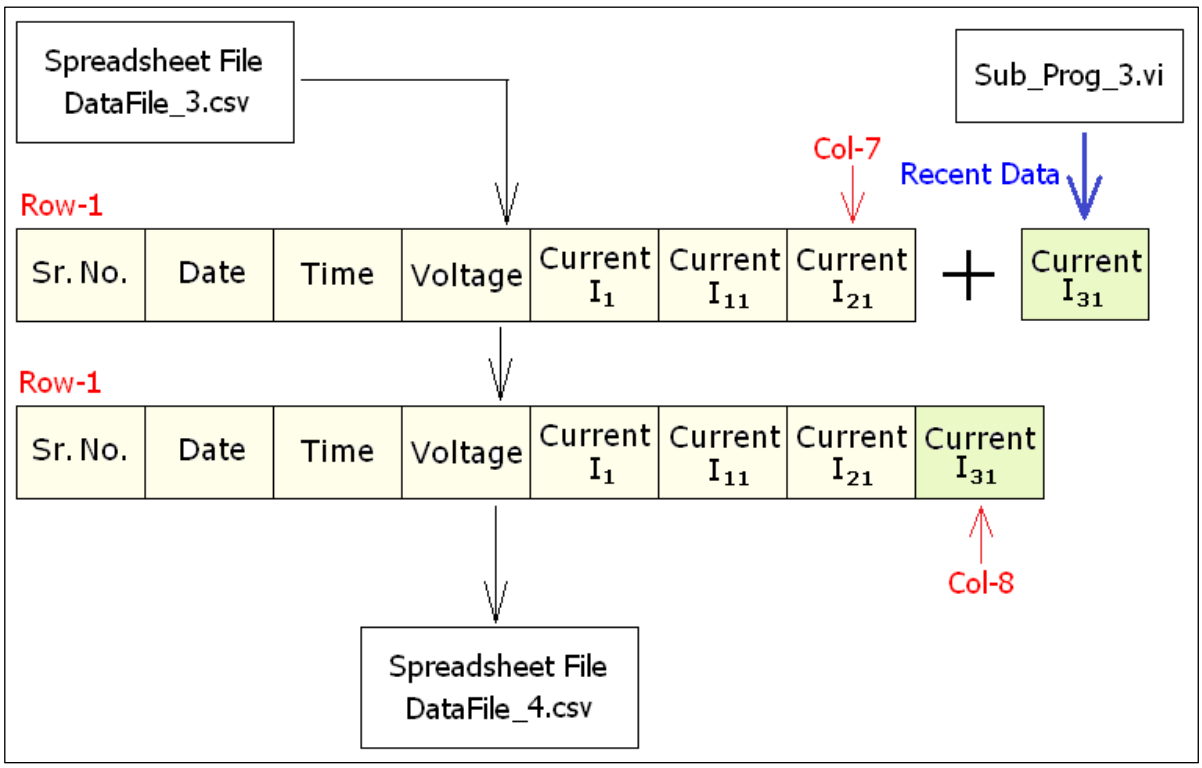


Figure 4: Reading of 1D array from a single row of the previous data file (DataFile_3.csv) and appending of recently collected latest data to it that saves in a newly created file DataFile_4.csv.
Source: Author, (2023).

As a next step of the *'Sequential Structure'* within the *'While Loop'*, in part P4.3 code instructs the PID controller to change the third variable. Thus it increments the temperature from T_2 to T_3, waits for 2 min. to allow the temperature to be stable at its new value, resets the voltage to zero (V_1) again to initiate a new data collection cycle in the next loop, and repeats the same process already defined. One such complete programme loop creates and saves another data files on the hard drive. The number of columns of variable-2 is created in a data file depending upon the number of programme loops iterates. Figure 5 shows the excel sheet of DataFile_10.csv after ten iteration loops.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1	02-03-2023	3	0	0	0.000004	0.000005	0.000006	0.000005	0.000005	0.00001	0.000012	0.000014	0.000017
2	2	02-03-2023	6	0.1	0.000004	0.000062	0.000069	0.000091	0.000109	0.000127	0.00015	0.000175	0.000214	0.00025
3	3	02-03-2023	9	0.2	0.000056	0.000858	0.001025	0.001205	0.001448	0.001702	0.002017	0.002356	0.002901	0.003405
4	4	02-03-2023	12	0.3	0.000733	0.013042	0.015202	0.01743	0.020215	0.022886	0.02586	0.028911	0.033252	0.036898
5	5	02-03-2023	15	0.4	0.011292	0.093493	0.100875	0.107938	0.116401	0.123745	0.131576	0.139368	0.149271	0.157594
6	6	02-03-2023	18	0.5	0.087035	0.26715	0.277639	0.287516	0.299174	0.309114	0.319617	0.329949	0.342414	0.353084
7	7	02-03-2023	21	0.6	0.257702	0.490781	0.502559	0.513709	0.526726	0.537844	0.549442	0.56128	0.574512	0.586357
8	8	02-03-2023	24	0.7	0.480115	0.739005	0.751341	0.763475	0.776839	0.788593	0.800904	0.813699	0.827107	0.839336
9	9	02-03-2023	27	0.8	0.727761	1.001112	1.013743	1.026497	1.040051	1.052179	1.064806	1.07805	1.091961	1.104061
10	10	02-03-2023	30	0.9	0.989532	1.271964	1.284771	1.297856	1.311613	1.323818	1.33695	1.350212	1.364786	1.376675
11	11	02-03-2023	33	1	1.260279	1.548791	1.561664	1.574889	1.588887	1.601122	1.614645	1.628063	1.642713	1.654591
12														

Figure 5: Screenshot of the data file (DataFile_10.csv) displayed within Excel sheet after completing tenth round of data acquisition in Loop-10 by the programme Sub_Prog_2.vi.
Source: Author, (2023).

IV. VERIFICATION OF THE CODE WITH A COUPLE OF PRACTICAL DATA ACQUISITION PROCESSES

The effectiveness of the programming process has been verified by two prototype data acquisition projects those are very common in a scientific or engineering laboratory. The first one was Current versus Voltage (I-V) characteristic of a Silicon diode (1N4007) against variation of temperature and in second case, the device under test (DUT) was a 470 ohm quarter watt resistance.

A Keithley Source meter (Model 2450) was used to generate voltage ranges from zero to 1v DC in the case of the diode and zero to 10v DC while the DUT was the resistance [3,4]. The source meter was interfaced with the desktop PC via GPIB interface. A PCI GPIB card (Make: National Instruments, USA) was used for this purpose. The front-panel connections are safety banana-type jacks.

To heat the copper sample holder, two 40 watt, and 12v cartridge heaters were used with a small size 3x2 mm^2 PT-100 RTD Sensor (Make: Hayashi Denko, Japan, Model: CRZ-2005-100-A-1) to sense the temperature of the copper block (Figure 6). A PID temperature controller (Make: Autonics, Model: TK4S-T4CR) with RS-485 interface facility was used to read, display and control the temperature with the help of an external relay [5,6]. A USB-to-RS485 converter (Model: SCM-US481) allows the device to interface with the PC through one of its USB port and it behaves just like a COM Port (Figure 7) [7].

Figure 6: (a) Keithley Model-2450 Source Meter and heater power supply integrated with an Autonics TK series temperature controller connected via a USB-to-RS-485 converter. (b) View of the copper-made heating plate mounted with two cartridge heaters, one PT-100 sensor, and a common silicon diode IN-4007 as the device under test (DUT).
Source: Author, (2023).

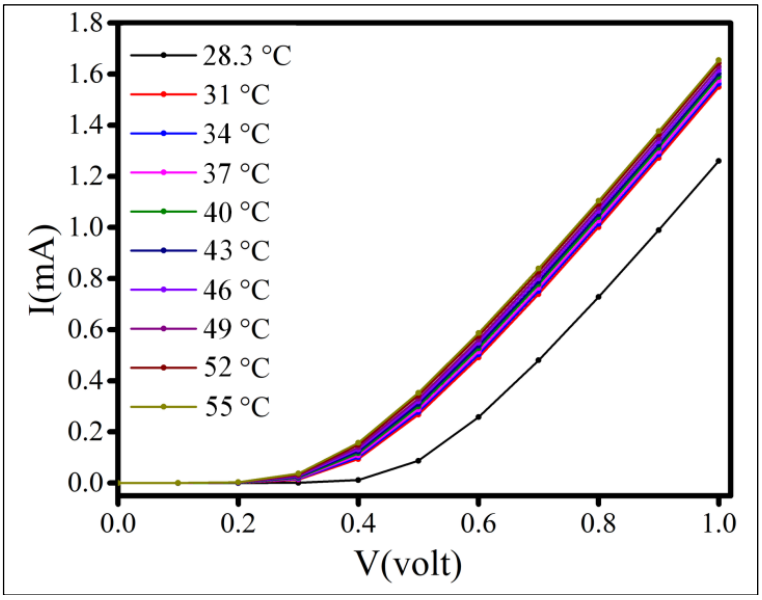


Figure 7. V-I curve of the Si diode plotted from DataFile_10.csv at different plate temperatures.
Source: Author, (2023).

A main graphical user interface file (Main_Prog_GUI.vi) was created to select the data file name and folder, range and increment value of applied DC voltage, display of monitored current, increment value of temperature at every cycle, etc. Several other LabVIEW vi files were prepared those were called from the main graphical user interface (GUI) file, and performed like subprograms of C++ or Java. One such VI file (Sub_Prog_3.vi) was used to source voltage and measure current by the Keithley source meter. Other VI files perform several other tasks like sending temperature set point to the Autonics controller, reading current process value (PV), setting the auto-tune facility on or off, etc.

V. DISCUSSION - RELEVANCE AND IMPORTANCE

A prototype test was performed to monitor the temperature dependency of a Silicon diode PN junction. The purpose of this sample study was to construct and develop an effective, efficient, and fully automated data acquisition system several independent measurement cycles can be performed automatically in a single test run without any human intervention.

VI. CONCLUSIONS

This software model has a few limitations that can be attended in the future for further improvement. I have kept the total number of acquisition loops within ten to avoid a problem during assigning new data file names when the number exceeds nine. If the file number becomes two digits i.e. more than nine then while manipulating the new file name by subtracting the last five characters (N.csv) from the end, it generates an erroneous file name. Interested programmers may focus on this point to avoid this limitation with improved logic.

The values of the third parameter (i.e. Temperature in my program me) are never recorded in the data file. It may be recorded at the end of the last row or prior to the first row to avoid ambiguity. Column titles are also not printed on the topmost row to identify the type of data for each column. That results in a kind of incompleteness in the data file. Both these drawbacks can be addressed in a future version either in LabVIEW or other programming languages.

VII. AUTHOR'S CONTRIBUTION

Conceptualization: Avijit Das.
Methodology: Avijit Das.
Investigation: Avijit Das.
Discussion of results: Avijit Das.
Writing – Original Draft: Avijit Das.
Writing – Review and Editing: Avijit Das.
Resources: Avijit Das.
Supervision: Avijit Das.
Approval of the final text: Avijit Das.

VII. ACKNOWLEDGMENTS

I am grateful to Mr. Subhadip Chowdhury, Senior Research Fellow of our institute, who generated the graphs from my data files and helped me to present it in my manuscript.

VIII. REFERENCES

[1] LabVIEW User Manual, National Instruments Corp.

[2] LabVIEW Arrays and Clusters Explained. Available in: https://www.ni.com/en-in/support/documentation/supplemental/08/labview-arrays-and-clusters-explained.html

[3] Keithley Model 2450 User Manual, Tektronix Inc. Available in: https://download.tek.com/manual/2450-900-01E_Aug_2019_User.pdf

[4] Keithley Model 2450 Reference Manual, Tektronix Inc. Available in: https://www.tek.com/en/keithley-source-measure-units/keithley-smu-2400-series-sourcemeter-manual/model-2450-interactive-sou

[5] Autonics TK series PID Controller communication. Manual, Available in: https://www.autonics.com/

[6] Autonics TK series PID Controller user manual. Available in: https://autonics.se/wp-content/uploads/2018/03/tk_en_manual_170829_hw.pdf

[7] Wikipedia – Modbus. Available in: https://en.wikipedia.org/wiki/Modbus

[8] What is the Modbus Protocol & How Does It Work?, National Instruments Corp. Available in: https://www.ni.com/en-in/innovations/white-papers/14/the-modbus-protocol-in-depth.html