A

PROJECT REPORT

ON

Development of Scientific Data Monitoring Platform using Scientific Linux and EPICS(Experimental Physics and Industrial Control System) Framework

Bachelor of Technology (CE)
Semester - VIII

Prepared By

Heli Desai (CE-024)
Sonee Patel (CE-062)
Vishrut Patel (CE-143)

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF TECHNOLOGY,

DHARMSINH DESAI UNIVERSITY
COLLEGE ROAD, NADIAD- 387001

April 2017

DHARMSINH DESAI UNIVERSITY
NADIAD-387001, GUJARAT

CERTIFICATE

This is to certify that the project carried out in the subject of Industrial Training entitled "Scientific Data Monitoring Platform using Scientific Linux and EPICS (Experimental Physics and Industrial Control System) Framework" and recorded in this report is a work of

Desai Heli	**CE-021**	**13CEUOG095**
Paghdar Soni	**CE-062**	**13CEUOS012**
Vishrut Patel	**CE-143**	**13CEUOS118**

of Department of Computer Engineering, semester VIII. They were involved in Project developing during academic session Dec-2016 to Apr-2017.

Prof. Brijesh S. Bhatt
(Project Guide),
Department of Computer Engineering,
Faculty of Technology,
Dharmsinh Desai University,Nadiad
Date:

Dr. C. K. Bhensdadia
Head of the department,
Department of Computer Engineering,
Faculty of Technology,
Dharmsinh Desai University, Nadiad
Date:

Scientific Data Monitoring Platform using EPICS and Scientific Linux

External Guide: Mr. Hitesk K. Gulati
Designation: Engineer-SF
Contact Number:079-23962152
Email-Id: hkg@ipr.res.in
Institution: Institute for Plasma Research
Address: Bhat Village,Near Indira Bridge,
Gandhinagar,
Gujarat 382428

Abstract

The work involves the study and understanding of open source EPICS framework software, its modules and required extensions which are running on Scientific Linux open source platform. This work also involves the installation, configuration of EPICS software on Linux.

The developed software will get data being generated in a buffer from PLC based hardware.

The data needs to be displayed real time and offline on another computer using graphical user interface developed using either in EPICS MEDM (Motif Editor and Display Manager).

Acknowledgements

With immense pleasure and commitment we would like to present the project assignment. The nature of project on the development of “Development of Scientific Data Monitoring Platform using Scientific Linux and EPICS(Experimental Physics and Industrial Control System) Framework” has given us wide opportunity to think, implement and interact with various aspects of management skills as well as the new emerging facilities and the technology used in architecture and the enhancements given to the students with a boon of spirituality and curricular activities.

Every work that one completes successfully stands on the constant encouragement, goodwill and support of the people around. We hereby avail this opportunity to express our gratitude to number of people who extended their valuable time, full support and cooperation in developing this software application .

We’d also like to take this opportunity to express our gratitude to Prof. Brijesh Bhatt Mr. Hitesh Gulati for his support and encouragement.
We express deep sense of gratitude towards our project guide Mr. Hitesh Gulati towards their innovative suggestions and efforts to make project a success. It is their sincerity that prompted us throughout the project to do hard work using the industry adopted technologies.

We are sincerely thankful to Head of CE department, Dr. C. K. Bhensdadia for the unconditional and an unbiased support during the whole session of study and development.

They altogether provide us favorable environment, without them we would not have achieved our goal.

Regards,

Desai Heli

Paghdar Soni

Patel Vishrut

Table of Contents

List of Figure

simulation an also inserts trigger in trigger file.

List of Tables

Chapter 1

Introduction

1.1 Background of Platform and EPICS framework

SCIENTIFIC LINUX

Scientific Linux is an Operating System developed by Fermilab as a sponsored project. It is primarily used by the High Energy and High Intensity Physics community.

CYGWIN

Cygwin is a Unix-like environment and command-line interface for Microsoft Windows. It provides native integration of Windows-based applications, data, and other system resources with applications, software tools, and data of the Unix-like environment.

EPICS FRAMEWORK

Experimental Physics and Industrial Control System (EPICS) is a set of Open Source software tools, libraries and applications developed collaboratively and used worldwide to create distributed soft real-time control systems for scientific instruments such as a particle accelerators, telescopes and other large scientific experiments.

FEATURES OF EPICS

- Provides a software infrastructure for use in building distributed control systems to operate devices and to allow communication between them.
- It uses client/server and publisher/subscriber model to communicate between systems .
- It is reliable, and provide facilities to ensure the resulting control system is maintainable and easily upgraded.

ASYN DRIVER

- ASYN driver is a general purpose facility for interfacing device specific code to low level communication drivers.
- Primary target of ASYN driver is for EPICS IOC device support.

EPICS BASE

EPICS Base is the core of EPICS, comprising the build system and tools, common and OS-interface libraries, Channel Access client server libraries, static and runtime database routines, the database processing code, and standard record, device and driver support.

1.2 CLIENT TOOLS

MEDM

MEDM is a Motif graphical user interface for designing and implementing control screens, called displays, that consist of a collection of graphical objects that display and/or change the values of EPICS process variables. The supported objects include buttons, meters, sliders, text displays/entries, and graphs. It has two modes of operation, EDIT and EXECUTE. Displays are created and edited in EDIT mode, and they are run in EXECUTE mode.

caQtDM

caQtDM is a package based on Qt developed at PSI as a successor of MEDM, a well known package used in the EPICS community for building synoptic displays. The caQtDM package uses the standard Qt GUI (Qt designer) for designing synoptic displays, containing all the controls graphical elements necessary to visualize and control a facility. Qt designer uses the custom widgets designed for this purpose and writes a description file (.ui file) that will be used by the synoptic viewer caQtDM.

STRIP TOOL

StripTool is a Motif application that allows you to view the time evolution of one or more process variables on a strip chart. It is designed to work with EPICS and is maintained as an EPICS Extension. There are two main windows: The Controls Dialogue and the Graph. The Controls Dialog allows you to specify and modify the process variable name and the graph parameters corresponding to each curve that is plotted.

ALARM HANDLER

The Alarm Handler is one of the major EPICS OPI Client applications, and is designed to provide an effective overview of any outstanding alarm conditions reported by the control system and also give the ability to manage the alarms in detail.

CASR

The caSaveRestore family of tools are intended to provide a facility to backup and restore process variable values.

CA Watcher

CA Watcher is a program which supervises EPICS variables. The user can receive warnings if desired. These warnings can be sounds on the local computer, SMS or emails. All detected alarm states including values are visible in a table and can be stored for further usage.

1.2 Problem Formulation

EPICS is used to communicate between various computers. One group of computers, here they can be called servers or input/output controllers, collect experimental and control data in real-time using the measurement instruments attached to it. This information is given to other group of computers, here clients using the Channel Access protocol. CA is a high bandwidth networking protocol, which is well suited to real-time applications such as scientific experiments.

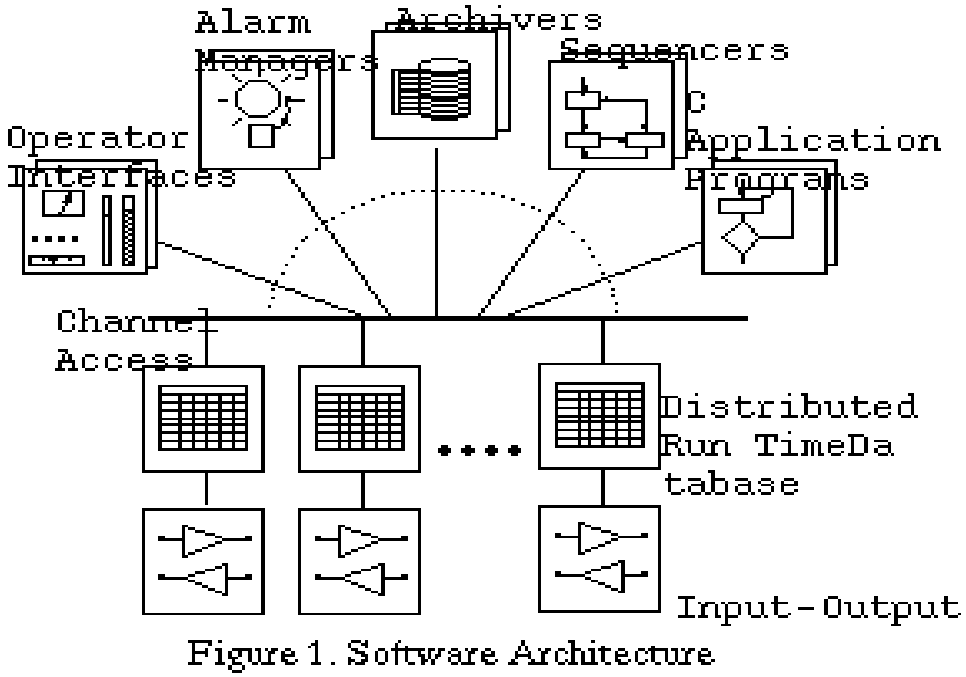

Figure 1. Software Architecture

Figure 1.1 Schematic showing architecture of EPICS

can be carried out in synchronization with any of the front end computers at a given time.

This data lies in the local front end machine unorganized and tough to find for future analysis. Hence, a system is needed so that the retrieved data can be efficiently monitored, plotted and analysed.

Chapter 2

About the System

2.1 Objective

To design and develop a Data Monitoring System that transfers the scientific data from the controller machine to the front- end machine and display it using MEDM.

2.2 System Description

The system is designed for monitoring experimental data for scientific analysis and research. Suppose an experiment being carried out and the controller server has the data which is to be sent to all the front end machines in the form of process variables(PVs).
All the clients can monitor the data received from server on MEDM screens.

The strip chart of the PVs received can be viewed using Strip tool. The PVs to be used to plot the graph are mentioned in the file.

If some threshold is crossed, there is an alarm handler that raises the alarm. The alarm handler reads the PVs to be monitored from .alhConfig file.

Data to be received is in the form of PVs(process variables) are mentioned in req1.req file, and are saved and stored using CASR (CA Save and Restore) in file snap1.snap, in case there is power failure or some problem with the machine. The PVs are then restored from snap1.snap to the previous state. There is also a .read.log file created each time, which has the logs for current session only.

////////////// The acquisition hardware and software for instance, LAB View or MATLAB will generate a text file with data values. These data values may be from different channels. Data values of different channels are comma separated in the main module file. Along with the text file, the software will also update a trigger file (in excel format) to record information as to when the sampling rate of the data changes.

Files that are used or created in the process include:

The main data fileis in .txt format and may have data with multiple channels. The values of data from multiple channels are comma separated. Each column of data indicates data from particular channel. The naming convention of this file is as: FEC1_M1.txt

where FEC1 indicates frontend computer-1 and M1 for module 1.

Apart from main data file, there's a trigger file in .xls or .xlsx format. The trigger file contains data regarding the sampling rate of the data. This information is used to time stamp data in chunk and segment files as well as where a new segment is to be started. The naming convention of this

file is as: FEC1_M1_Trigger.xlsx 4
where FEC1 indicates frontend computer-1 and M1 for module 1.

The channel file is a text file with data of just one particular channel. The naming convention of this file is as: FEC1_M1_Ch1.txt

where FEC1 indicates frontend computer-1, M1 for module 1 and Ch1 for Channel 1.

A chunk file is a text file with particular channel data divided in chunks as per user specified chunk size. This data in a chunk file is preceded by a timestamp to uniquely identify when the data value was acquired. The naming convention of this file is as: FEC1_M1_Ch1_Chunk_1.txt

where FEC1 indicates frontend computer-1, M1 for module 1, Ch1 for channel 1 and Chunk_1 for chunk 1.

The segment file is a text file with particular channel data divided in segments as per information from trigger file. This data in a segment file is preceded by a timestamp to uniquely identify when the data value was acquired. The naming convention of this file is as: FEC1_M1_Ch1_Seg_1.txt

where FEC1 indicates frontend computer-1, M1 for module 1, Ch1 for channel 1 and Seg_1 for segment 1.

Data Archival System has a graphical user interface to interact with end user on the front end computer where the experiment is performed or where the experimental data is acquired. Initially, the system asks front-end user for chunk size of modules. On the user input; as per operation type, the system processes the data from main experiment file and segregates different channel data. Directories are made of each channel and the segregated data is thus stored in a text file in its respective channel directory. The channel directory will also have two other directories made dynamically - Channel chunks directory and Channel segments directory.
As per chunk size mentioned by the end-user and the operation performed, the channel data is divided and stored in form of chunks in chunk files. These chunk files

reside in chunk directory of its particular channel. These chunk files are used in plotting and analytics purpose on retrieval.

Similarly, as per information in trigger file of the experiment module, channel data is divided into segments. Each new segment file indicates change in sampling rate of data values. These segment files are stored in segment directory of its particular channel.

All these files are uploaded to FTP server in the same hierarchy upon their respective completion.

Chapter 3

Software requirement specifications

3.1 Introduction

3.1.1 Purpose

The purpose of this document is to specify the detailed description of the Scientific Data Monitoring Platform that we will design and implement. This document is intended to the Customers and Developers. It will explain the purpose and features of the system, what the system will do, the constraints on which it must operate.

3.1.2 Scope

The Goal of the System is to develop a Standalone Data Monitoring Platform. The System provides two types of user privileged and nonprivileged user. The system will allow user to monitor as well as control system parameters based on the privileges. The system will have database of application at server side.

In Scope,

Operation Selection.

Server Selection.

Enter Chunk size.

Upload to FTP server

Quit

Not in scope,

No files can be processed accept .txt and .xls/.xlsx files.

Technology and Tools Used

- EPICS Framework
- ASYN
- MEDM
- Alarm Handler
- Strip Tool
- CASR (CA Save and Restore)
- Python

3.2 General Description

3.2.1 Hardware Interface

Memory with minimum 2 GB ram

Hard Disk of 40 GB

Monitor

Mouse

Keyboard

3.2.2 Software Interface

Operating System : Scientific linux and Cygwin

Backend : EPICS base

Frontend : MEDM, ALH, Strip Tool, CASR

3.2.3 Communication Interface

Communication between front end Computers and controller server takes place through Internet.

3.3 Functional Requirements

This section contains all of the functional and quality requirements of the system. It gives a detailed description of the system and all its features.

R 1) Installation

Installation of Scientific Linux and Epics base and relevant Client Tools
Input: Scientific Linux iso file and tar.gz compressed tar balls of all required softwares
Output: Framework ready for development of the application

R 2) Installation of Device drivers to interface with the Server PC

Input: Device Model
Output: Server interface with Device

R 3) Get Parameter values on Server

System get parameter values on server PC.

Input: Switched on power supply connected to server through RS-232 cable, asyn driver.

Output: Process Variable values on server.

R 4) Get Process Variable Values on clients

System gets PV values from power supply and broadcasts it with a specific PV name to all machines on the network through EPICS Application

Input: Process variable values generated by the Power Supply

Output: Process Variable Values on client

R 4.1) System Gets Parameter values on Server

Input: Parameter values generated by the Device(Power Supply)

Output: Parameter values on server

R 4.2) System gets Process variable values on EPICS

Input: Parameter values on server

Output: Process Variable

R 4.2.1) System Identifies variables through Database defined in EPICS

Input: Parameter values from Receieved on Server

Output: Identified parameter values

R 4.2.2) System gets parameter values on EPICS through .proto files defined in application

Input: Identified parameter value

Output: Data on EPICS

R 4.3) System binds parameter value to Process variable as defined in EPICS Database

Input: Identified parameter value

Output: Process Variable

R 4.4) System broadcasts process variables to clients on same network as server

Input: Process variable

Output: PV'S broadcasted on network

R 5) Alarm Handler

System raises alarm for Process variables based on conditions defined in EPICS Database

Input: Process Variable on channel

Output: Color display and alarm sound as per the state

R 5.1) System scans Process variables regularly after specific intervals

Input: Process variable on channel

Output: Process variable value in alarm handler application

R 5.2) System compares the Process variable value with the database

Input: Process variable value with alarm handler application, alarm state range for each Process Variable

Output: Alarm state

R 5.3) System raises specific alarm based on the state of Process variable

Input: Alarm state

Output: Colour display and alarm sound as per the state

R 6) CA Save Restore

System saves the state of the process variables when it is in normal state and restores those values when the system goes down

Input: ASCII file

Output: PV Values restored to power supply

R 6.1) System gets the process variables to be saved from ascii file

Input: ASCII file

Output: List of process variavles to be saved

R 6.2) System gets the process variable values from channel

Input: List of process variable

Output: PV and their values

R 6.3) System writes the PV to snap files

Input: PV and their values

Output: Snap file

R 6.4) System restores the PV to the power supply when required

Input: Snap file

Output: PV Values restored to power supply

R 7) MEDM and caQtDM (GUI monitor and controller)

System provides a Graphical User Interface to control as well as monitor the Process Variables based on the User Privileges

Input: PV Name, PV value

Output: PV Value or changed value on device

R 7.1) System scans the channel regularly and displays the current value of Process variable

Input: Process Variable from channel

Output: GUI Display of current value of PV

R 7.2) System allows privileged users to change the value of a PV

Input: PV Name and Value given by user

Output: PV value changed on the device

R 7.2.1) System takes the value from user

Input: GUI screen

Output: PV Value to be set

R 7.2.2) System sends the PV Value to the server

Input: PV Value given by user

Output: PV Value sent to server

R 7.2.3) Server changes the paramter value on the device

Input: PV Value received from client

Output: PV changed on device

R 8) Strip Tool

System display a strip chart of different process variables values with respect to time

Input: stp file

Output: Strip chart

R 8.1) System gets the process varible details of PV's that are to be plotted from stp file

Input: stp file

Output: PV details

R 8.2) System monitors all the process variables mentioned in stp file and plots it on strip chart

Input: PV details

Output: Strip Chart

R 9) CA Watcher(alarm handler and data storage module)

System saves the Process Variable values in a file with time stamp

Input: PV name, PV values

Output: Alarm, email, or data file

R 9.1) System gets the PV name from user

Input: GUI screen to enter PV name

Output: PV name

R 9.2) System monitors the values of specified PV's at regular intervals and stores it in a file

Input: PV Name and Values

Output: Data file

Chapter 4

Analysis

4.1 Usecase Diagram

Figure 4.1 UseCase Diagram of Data Archival System

4.2 Sequence Diagram

Figure 4.2 Sequence Diagram of Data Archival System

4.3 Activity Diagrams

4.3.1 Activity Diagram for pulse operation

Figure 4.3 Activity Diagram of Pulse Operation

4.3.2 Activity Diagram for continuous operation

Figure 4.4 Activity Diagram of Continuous Operation

4.4 Directory Structure

Figure 4.5 Directory Structure used for Data Archival System

Chapter 5

Implementation

The development of Data Archival system consisted developing 3 major modules:The Graphical user interface, module for pulse operation and module for continuous operation.

5.1 The Graphical User Interface

The Data Archival system needs an interface to interact with the user at the front-end computer (FEC) to gather details regarding experiment been carried out, the path of the hierarchy on the front-end computer (FEC) where the files with the experiment data are stored and the server details as to where the data is to be archived.

The GUI is for the system is programmed using python programming language. Python supports wide range of open source libraries needed to build the code base for the system.In order to build the graphical interface, Tkinter- a standard GUI library for Python is used. Python when combined with Tkinter provides a fast and easy way to create GUI applications.Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. As with most other modern Tk bindings, Tkinter is implemented as a Python wrapper around a complete Tcl interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

Definitions of Tkinter components used-

Window- This term has different meanings in different contexts, but in general it refers to a rectangular area somewhere on the user's display screen.

Top level window- A window that exists independently on the screen. It will be decorated with the standard frame and controls for the desktop manager. It can be moved around the desktop, and can usually be resized.

Widget- The generic term for any of the building blocks that make up an application in a graphical user interface. Examples of widgets: buttons, radio buttons, text fields, frames, and text labels.

Frame- In Tkinter, the Frame widget is the basic unit of organization for complex layouts. A frame is a rectangular area that can contain other widgets.

Child and parent-When any widget is created, a parent-child relationship is created. For example, if you place a text label inside a frame, the frame is the parent of the label.

The Graphical interface is designed using the above mentioned components from Tkinter module. Following is illustration of GUI designed for the system:

Figure 5.1 Graphical user interface

The GUI is divided into three frames-

Operation details, Data Server details and Action buttons.

Operation details

Experiment- The drop down menu provides the selection operation to be performed based on the type of experiment carried out. For the current system-Pulse operation and Continuous operation.

Directory path- A field is provided to get the path of the experiment files with a browse button providing convenient option to select path from windows explorer.

Data server details

Destination-Address - A drop down menu is provided to select the server from given list as to where the experiment files are to be archived. It lists three servers to select from for the current system.

User Directory Access

Username- The text field is to get username from the front end user as credential so as to log on to the server selected above.

Password- The password field takes password from front end user as credential so as to log on to the server selected above.

Action Buttons

Proceed-This action button is used to proceed to carry out the archival operation as per the details provided by the front-end user in the above fields.

Quit- This action button is used to exit the graphical user interface.

Along with the basic interface, the GUI also consists of prompt to get chunk size from the front-end user as well as alerts on successful completion of operation and errors in user input.

Figure 5.2 Prompt to get chunk size

Figure 5.3 Operation success alert

Figure 5.4 Error alert 18

5.2 Pulse Operation

The Pulse operation is to be performed for data archival of experiment that is complete. In such case, the experimental data file as well as the trigger file- that has information regarding sampling rate, has to be complete and should not be in use by any of the processes running. The pulse operation segregates all channels into its respective directory. The channel data is

then divided into chunks and segments. It iterates over all the modules present in the directory path specified, and performs the pulse operation on each of them in a sequential manner.

Figure 5.5 Module hierarchy

Two arguments are to be passed for the pulse operation- 1. Module number, for which pulse operation is to be carried out and the chunk size of that particular module. This is necessary if the module is complete and there is a requirement to carry out pulse operation on a particular module during continuous operation. Else, if pulse operation is to be carried out directly on several complete modules, the module argument must be string ‘all’ and the chunk size argument to be 0.

Figure 5.6 Module directory

The system needs the required files- the experiment data file and the trigger file, to be present and complete in the module directory to proceed with further implementation. The data file contains data values acquired by the acquisition hardware and software. These files are created and updated by softwares directly. The entire operation from archival to retrieval and plotting depends on the data in these files.

The main data file may have values from multiple channels. These values are comma separated. Each column in the data file represents data from particular channel. These channels are further split upon performing one of the two operations on the data.

Figure 5.7 Experimental data file with four channels

Rename operation on a file throws an Exception if the file is being used by any other application at a given time. Using this principle, a function was devised to know whether a file is being used by another process or not. The system alerts the user accordingly if the file is not complete. The system is also designed, so as to record the information about modules with missing files; hence show an alert to the user about the same at the end of operation.

Figure 5.8 Files not found alert

Figure 5.9 Files in use alert

Once the system acquires all the required files for implementation, it prompts the user to input chunk size for the particular module. Once the user inputs the chunk size, the system creates a ‘Module-Channels’ directory in the module directory on the local machine. Then, directories equal to number of channels in the data file are created within the ‘Module-Channels’ directory.

Figure 5.10 After pulse operation

Figure 5.11 Module-Channels directory

A channel file is written in the respective channel directory containing data of only that particular channel. The system iterates over each channel data sequentially to carry out the process. The channel file is made by fetching entire column stack from the main data file and writing it to the channel text file inside its respective channel folder.

Figure 5.12 Channel file

Figure 5.13 Channel directory

Apart from the channel file, two more directories are made - channel chunk directory and the channel segment directory. The system divides the channel data into number of chunks based on the chunk size provided by the user. The channel data is divided into segments based on the information from the trigger file. Both the chunk and segment files has data values along with a time stamp to uniquely identify acquisition of each data value.

getTriggerData (trigger_file): A function to get trigger data from the trigger file was made that returns array of all the rows within the module trigger file. The trigger file is read using xlrd

library of Python. xlrd is a python library to extract data from Microsoft Excel spreadsheet files. xlrd can extract data from Excel spreadsheets (.xls and .xlsx, versions 2.0 onwards) on any platform, is written in pure Python and is Unicode-aware.

Each row in the trigger file contains data as to from which data value the sampling rate of the data changes. The first column is the serial number, start sample in the second column and the sampling rate in the third column. This information is used to generate time stamps for each data value to be written in the chunk and segment files.

Figure 5.14 Trigger file

During pulse operation, the system iterates over all the rows of the trigger data and breaks the channel stack into segments based on trigger information. The segment stack sliced from the channel stack is preceded with time stamps. Hence, the sliced channel stack and time stack are concatenated to make a segment stack. This segment stack is then written to create particular segment file.

getTimeStack (range, dT, TS): This function takes in range, dT and TS as argument, where range is the range of time stack needed, dT=1/ [current sampling rate] and TS is the timestamp of data and is obtained for every data by performing TS=TS+dT. The function returns required time stack of range specified.

Figure 5.15 Segment directory

Figure 5.16 Segment file

The system then proceeds further to make chunks of channel data. A chunk is sliced channel data according to the chunk size specified by the end user. The sliced channel data is preceded by time stack created during segmentation. The sliced channel stack and time stack are concatenated to make a chunk stack. This chunk stack is then written to create particular chunk file.

Figure 5.17 Chunk folder

Figure 5.18Chunk file

With all the files and directory hierarchy created at the local machine, it is then uploaded to the server using FTP protocol. The uploading process is discussed later in the chapter.

5.3 Continuous operation

The Continuous operation is to be performed for data archival of experiment that is running. In such case, the experimental data file as well as the trigger file- that has information regarding sampling rate, is not complete and are in use by the acquisition hardware or software running the experiment. The continuous operation dynamically segregates all channels into its respective directories in chunks. Chunks and segments for the received data are made. It iterates over all the modules present in the directory path

specified, and performs the continuous operation on each of them by the chunk size specified for each module by the user.

Figure 5.19 Module directory (with no data file) before experiment

Once the experiment starts, the data file is created within the module directory and is dynamically updated. Along with the data file, the trigger file too is dynamically updated by the experiment upon change in sampling rate.

For continuous operation, the system is designed to monitor the module directory continuously and wait till it finds the data file. Once the experiment begins and the data file is created, the system prompts user for chunk sizes of modules. Now that chunk sizes are acquired by the system, it needs to process all the modules. The major issue with continuous operation over data is that the data is continuously required to be sent to server with little delay as possible so as it can be retrieved and plotted to be analysed in real time.

Figure 5.20 Continuous operation: waiting for required files

This operation can be done sequentially over all the modules, but is not a feasible option as it devotes all time to just one module while other running modules have to wait for their turn till current module is completed. Also, it can be done based on chunks, for instance- The system could switch between modules one completion of operation over one chunk of a module. This solution eliminates the problem of longer waiting time for the modules, but is still not feasible as it may have variations in behaviour with different chunk sizes. For instance- module 1 with chunk size 50 will have to wait for its next turn till 200 data values with module 2 chunk size 200. This will create 3 of the chunks for module 1 altogether. Although it works well for the archival, while not a good option

keeping in mind the retrieval scheme as it skips over 2 chunks from module 1 completely in the plot. Ultimately, a technique was devised to eliminate the disadvantages of the rest. Instead of iterating over modules sequentially, sequence of the modules are decided based on chunk sizes

of the particular modules. On getting number of data values in multiples of any of chunk sizes, the system iterates through modules that have chunk sizes either equal to the number of values or less. This helps in getting chunks of modules while less time for other modules to wait.

Figure 5.21 Continuous operation: experiment started

While system processes a chunk for given module, it does the process similar to that in pulse. Sliced channel stack is written to channel file. Channel files are created

from the data file into their respective directories. The same sliced stack is used to make a chunk stack by preceding it with time stack. The chunk stack is then written to create chunk file into chunk directory. While processing module for every chunk, the trigger file is read and trigger data is updated. If one or multiple rows are added to the trigger file for that particular chunk, the system iterates through all the updated rows and prepares segments accordingly.

The files are uploaded to the ftp server once they are created onto the local machine.

5.4 FTP Server and upload

Data is archived to the server in this system using File Transfer Protocol. The File Transfer Protocol (FTP) is a standard network protocol used to transfer computer files between a client and server on a computer network. FTP is built on a client-server model architecture and uses separate control and data connections between the client and the server.FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.

ftp_upload()- This function can be called at the start and after file creation in the directory hierarchy. It goes through the directory specified by the user and creates the directory structure similar to that on the local machine onto the server. When the performing operation creates files and directories inside module directory, this function walks through the entire hierarchy creating each directory and files similar to that created on the local machine. It also limits the uploading process to the modules inside specified shot directory only.

dir_exists(path)- This function is used to check whether the directory already exists on the server or not. This is also It also takes care that there be no duplication of the hierarchy structure. If user tries to upload content already existing on the server, the system alerts the user about it.

To execute the entire uploading process, ftplib library of python is used.This module defines the class FTP and a few related items. The FTP class implements the client side of the FTP protocol. It is used to write Python programs that perform a variety of automated FTP jobs.X

Functions used:

ftplib.FTP(host='', user='', passwd='', acct='', timeout=None, source_address=N one)

Return a new instance of the FTP class. When host is given, the methodX

call connect (host) is made.

FTP.login(user='anonymous', passwd='', acct='')

Log in as the given user. The passwd and acct parameters are optional and default to the empty string. his function should be called only once for each instance, after a connection has been established; it should not be called at all if a host and user were given when the instance was created. Most FTP commands are only allowed after the client has logged in.

FTP.cwd(pathname)

Set the current directory on the server.

FTP.mkd(pathname)

Create a new directory on the server.

FTP.dir(argument[, ...])

Produce a directory listing as returned by the LIST command

FTP.quit()

Send a QUIT command to the server and close the connection.
FTP.retrlines(cmd, callback=None)

Retrieve a file or directory listing in ASCII transfer mode.cmd should be an appropriate RETR commandor a command such as LIST or NLST.

FTP.storbinary(cmd, fp, blocksize=8192, callback=None, rest=None)

Store a file in binary transfer mode.fp is a file object (opened in binary mode) which is read until EOF using its read () method in blocks of size block size to provide the data to be stored. The blocksize argument defaults to 8192. callback is an optional single parameter callable that is called on each block of data after it is sent. X

Figure 5.22 FTP file uploading

Figure 5.23 Continuous operation: file upload and directory creation

Figure 5.24 Continuous operation: Module switching

Chapter 6

Testing

6.1 Introduction

Testing is the process carried out on software to detect the differences between its behaviour and the desired behaviour as stipulated by the requirements specifications.

Software Testing

Software testing is the process of evaluation a software item to detect differences between given input and expected output. Also to assess the feature of a software item. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words software testing is a verification and validation process.

Verification

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to.

Validation

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

6.2 Basics of software testing

There are two basics of software testing: blackbox testing and whitebox testing.

Blackbox Testing

Black box testing is a testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

Whitebox Testing

White box testing is a testing technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

Black box testing is often used for validation and white box testing is often used for verification.

6.3 Types of testing

There are many types of testing like

Unit Testing

Integration Testing

Functional Testing

System Testing

Stress Testing

Performance Testing

Usability Testing

Acceptance Testing

Regression Testing

Beta Testing

37
Unit Testing

Unit testing is the testing of an individual unit or group of related units. It falls under the class of white box testing. It is often done by the programmer to test that the unit he/she has implemented is producing expected output against given input.

Integration Testing

Integration testing is testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested in integration testing if software and hardware components have any relation. It may fall under both white box testing and black box testing.

Functional Testing

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

System Testing

System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing.

Stress Testing

Stress testing is the testing to evaluate how system behaves under unfavorable conditions. Testing is conducted at beyond limits of the specifications. It falls under the class of black box testing.

38
Performance Testing

Performance testing is the testing to assess the speed and effectiveness of the system and to make sure it is generating results within a specified time as in performance requirements. It falls under the class of black box testing.

Usability Testing

Usability testing is performed to the perspective of the client, to evaluate how the GUI is user-friendly? How easily can the client learn? After learning how to use, how proficiently can the client perform? How pleasing is it to use its design? This falls under the class of black box testing.

Acceptance Testing

Acceptance testing is often done by the customer to ensure that the delivered product meets the requirements and works as the customer expected. It falls under the class of black box testing.

Regression Testing

Regression testing is the testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. It falls under the class of black box testing.

Beta Testing

Beta testing is the testing which is done by end users, a team outside development, or publicly releasing full pre-version of the product which is known as beta version. The aim of beta testing is to cover unexpected errors. It falls under the class of black box testing.

39
6.4 Test Plan

The testing sub-process includes the following activities in a phase dependent manner:

Create Test Plans.

Create Test Specifications.

Review Test Plans and Test Specifications.

Conduct tests according to the Test Specifications, and log the defects.

Fix defects, if any.

When defects are fixed continue from activity.

Figure 6.1 Testing plan

6.5 Test Cases

Table 6.1 Test cases

Sr no	Test Case	Input	Expected Result	Output
1.	FTP Login	Valid Username and Password	Login successful and start pulse operation	Success
2.	FTP Login			

	Invalid Display error Success		
	Username or message		
	Password		
3.	Check Main Main file exists Continue pulse Success		
	File in Pulse and not in use operation		
	Operation		
4.	Check Main Main file does Display error Success		
	File in Pulse not exists message		

Operation

5.
Check Main
Main file does
Waiting for file
Success

File in
not exists

continuous

operation

6.
Check main file
Main file exists
Execute Pulse
Success

in continuous
and not in use
Operation

operation

7.
Check main file
Main file exists
Execute
Success

in continuous
and in use
Continuous

operation

Operation

8.
FTP Upload
Directory does
Upload to FTP
Success

not exists
server

9.

FTP Upload
Directory
Display error
Success

already exists
message

6.6 Screenshots

Figure 6.2 Login Fail

Figure 6.3 Main file absent from the directory during pulse operation

Figure 6.4 Directory already exists on FTP server prompt

6.7 Testing with scripts

Running the system with hardware and creating actual experiment environment was not feasible option to test some trivial functions and modifications. Hence in order to test and correct errors the system was tested under similar environment but simulated using scripts.

Especially continuous operation had been rigorously tested by simulating the test data file using scripts. Data files with one or more channels with different data was made for different modules in real time. And hence, testing and correction of the operation efficiency in absence of hardware, but similar environment was possible.

Figure 6.5 Script execution that generates comma separated data values simulation an also inserts trigger in trigger file.

Chapter 7

Conclusion and future extensions

7.1 Conclusion

The objective of the Thesis was to satiate the functional requirements of system required for archival of experimental data in an organised and efficient form for future analysis by retrieval. The graphical interface for the system is designed to provide user-friendly experience for the end user from any background. Data archival system is designed and developed in such a way so as to ensure dynamic archival of data for completed experiment as well as the one running.

7.2 Future extensions:

The Thesis was mostly aimed to satisfy and provide working as per needs for some specified experiments and their data hierarchy. It can be further more generalized and prepared for more broad range of experiments without being concerned about the directory structure of the scientific data.

The Thesis can further extended to work more dynamically and parallelism than the current version. For different types of experiments and needs for retrieval, more operations can be developed using functions and platform developed during current Thesis.

Bibliography

https://www.python.org/doc X

www.tutorialspoint.com X

http://stackoverflow.com X

[4] https://www.codecademy.comX

http://www.javatpoint.com/python-tutorial X

Head First Python by David Griffiths, author and Agile coach

A Byte of Python by Swaroop C H X

Python Programming by john zelle

The Python Tutorial by guido van rossum