Phasor Data Concentrator Code Documentation

Programmers Manual

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology

by

Kedar Khandeparkar
&
Nitesh Pandit

under the guidance of

Prof. A.M. Kulkarni
Indian Institute of Technology, Bombay

I would like to express my sincere thanks and gratitude to my guide, Prof. V.Z. Attar, Prof. S.U. Ghumbre and Prof. A.M. Kulkarni for the constant motivation and valuable guidance they provided me throughout the course of the project. I am highly indebted to them for giving me this opportunity to work under them for this M.Tech project, and also clarifying my doubts and for bringing into perspective, the different aspects of the project topic. They constantly encouraged me by giving suggestions and criticisms on my work. Working under them has been a great learning experience for me.

Abstract

With information and measurement technology evolving rapidly within the electric power industry, wide-area measurement is deemed to be the key technology to improve power system reliability. Phasor Measurement Unit (PMU) deployment for Wide Area Measurement System(WAMS) results in large amountś of data being generated every second across the PMU network. Currently, many transmission operators do not have the ability to process, store, and utilize this data. We aim to build a Phasor data concentrator that can do these operations. This report gives the complete implementation details of the iPDC Phasor Data Concentrator.

Contents

List of Figures

Organization Of the Report

1. Chapter 1 describes the various architecture & design of iPDC.
2. Chapter 2 describes handling of data & configuration frames ar iPDC.
3. Chapter 3 describes iPDC Database.
4. Chapter 4 describes WAMS Security.
5. Chapter 5 describes the design of PMU Simulator.

1 General Working of iPDC

PMU's/iPDC's act as servers when they are communicating with another iPDC whom they are sending the data/configuration frames. The iPDC receiving data in this case acts as a client. However when the same iPDC sends data to another iPDC, it would act as a server. This pattern will be repeated in the WAMS topology with one peer acting as server and its counterpart a client.

The iPDC when acting as a server binds to 2 ports UDPPORT and TCPPORT. It would be listening for UDP connections on UDPPORT and TCP connections on TCP-PORT. The iPDC can then send the combined configuration frames to any number of other iPDC's. Both the communicating peers authenticate each other. iPDC authenticates for each received packets irrespective of communcation protocols used(TCP/UDP).

When the iPDC starts for the first time the user is prompted to enter iPDC Idcode, UDP Port, TCP Port and Database Server IP. The ports enable iPDC to receive requests from other iPDC and to send the combined data and configuration frames. Database Server IP is the IP address of the machine where the process dbserver is running. The default port on which dbserver is listening for data is 9000 and it is a UDP server. The data which the iPDC receives would also be directed to dbserver for storage in MySQL database.

The user is provided with the following options at iPDC

- Enter iPDC Setup
- Add a Source Device
- Remove a Source Device
- Turn OFF the data Transmission
- Turn ON the data Transmission
- Request Configuration frame
- Add a Destination Device
- Remove a Destination Device
- iPDC Connection Table

1.1 Enter iPDC Setup

This is the first pop up window that would ask the user to enter the iPDC setup information. It includes UDP & TCP ports on which the iPDC would receive command frame requests from other iPDC's. The two ports should be different. The combined data and configuration frames would be sent on these ports. iPDC Idcode also need to be entered. IP address of the machine where the data would be stored in MySQL databaseis also need to be entered.

This functionality is distributed in the following functions :- *void fill_pdc_details (), int validation_pdc_detail (GtkButton *button, gpointer udata)* from the file **ipdcGui.c**.

1.2 Add a Source Device

This would make an entry in the file ipaddress.txt for each newly added PMU. Before making the entry it would check if there is already a connection with the same node. If so, no entry is made. Otherwise a new thread would be created for each connection based on type of the protocols that have been selected by user (TCP/UDP). Also a new node of the type Lower_Layer_Details would be created.

This functionality is distributed in the following functions :- *int add_PMU(char pmuid[], char ip[], char port[], char protocol[]),void* connect_pmu_tcp(),void* connect_pmu_udp(), void add_PMU_Node()* from the file **new_pmu_or_pdc.c**
and *void PMU_process_ UDP(), void PMU_process_ TCP* from file **connections.c**.

1.3 Remove a Source Device

This would display the user with available PMU node entries and would prompt the user to enter the details of node to be removed. When user enters the correct details, the PMU node entry would be removed from the fie ipaddress.txt. Also the corresponding Lower_Layer Details node would be removed from doubly LL. A signal would be sent to the thread that handles this connection. On receival of the signal the thread would close the socket and exit.

This functionality is distributed in the following functions :- *int remove_Lower_Node(char pmuid[], char protocol[]), void remove_Lower_Node() and void* remove_llnode(void*)* from the file **new_pmu_or_pdc.c**.

1.4 Turn OFF the data Transmission

This would display the user with available PMU entries and would prompt the user to enter the details of node whose data transmission is to be put off. When user enters the correct details, the command frame would be sent to that node. The flag d́ata_transmission_off óf the corresponding node in Lower_Layer_Details structure doubly LL would be set to 1.

This functionality is distributed in the following functions :- *int put_ data_ transmission_ off (char pmuid[], char protocol[]) and void* data_ off_ llnode(void* temp)*
from the file **new_pmu_or_pdc.c**.

1.5 Turn ON the data Transmission

This would display the user with available PMU entries whose data transmission has been put off and would prompt the user to enter the details of node whose data transmission is to be put on. When user enters the correct details, the command frame to put off data tramsission would be sent to that node. The flag d́ata_transmission_off óf the corresponding node in Lower_Layer_Details structure doubly LL would be reset to 0.

This functionality is distributed in the following functions :- *int put_ data_ transmission_ on (char pmuid[], char protocol[]) and void* data_ on_llnode(void* temp)*
from the file **new_pmu_or_pdc.c**.

1.6 Request Configuration frame

This would display the user with available PMU entries and would prompt the user to enter the details of node to whom the configuration frame request is to be sent. When user enters the correct details, the command frame to send the configuration frame would be sent to that node.

This functionality is distributed in the following functions :- *int configuration_ request(char pmuid[], char protocol[]) and void* config_request(void* temp)*
from the file **new_pmu_or_pdc.c**.

1.7 Add a Destination Device

An entry is made in the file upperpdc_ip.txt. Also a node of the type Upper_Layer_Details is created. It includes ip, port and protocol details of other iPDC. iPDC when acting as a server may get connection requests on 6000 and 6001 ports for UDP and TCP repectively. The received command frames are authenticated againt the entries in the Upper_Layer_Details doubly LL. If the command frame is for configuration frame then, check is made if there are no Idcodes in the śtatus_change_pmupdcid ́ śtatus_change_pmupdcid ís a LL which maintains the idcodes of all PMUś whose configuration has beed changed. If the list is not empty then, poll till the list is empty. When a new Configuration frame of the corresponding ID codes arrives then its entry is removed from the list. After the list is empty create_cfgframe() is called that creates combined configuration frame from configuration objects. Then the flag UL_upper_pdc_cfgsent in the corresponding Upper_Layer_Details node of the PDC is set.

If the command frame is for data then, UL_upper_pdc_datasent is set and UL_data_transmission_off in the corresponding Upper_Layer_Details node of the iPDC is reset. The functions dispatch() in align_sort() will send the combined data on one of the sockets by checking the variables UL_use_udp, UL_upper_pdc_cfgsent and UL_data_transmission_off. If the command frame is for data transmisssion off then UL_data_transmission_off in the corresponding Upper_Layer_Details node is set.

This functionality is distributed in the following functions :- *int add_PDC(char ip[], char protocol[])* from the file **new_pmu_or_pdc.c** and *void* UL_udp() and void* UL_tcp()* from file **connections.c**.

1.8 Remove a Destination Device

This would remove the iPDC entry from the file upperpdc_ip.txt. Also the corresponding node of the type Upper_Layer_Details would be removed from the doubly LL.

1.9 iPDC Connection Table

This would display the connections tables showing PMU and iPDC details to which the iPDC is connected.

2 Source Code Documentation of iPDC

The code is distributed in the following files

- iPDC.c
- ipdcGui.c
- ipdcGui.h
- recreate.c
- recreate.h
- connections.c
- connections.h
- new_pmu_or_pdc.c
- new_pmu_or_pdc.h
- parser.c
- parser.h
- dallocate.c
- dallocate.h
- align_sort.c
- align_sort.h
- global.h

2.1 iPDC.c

It is the file that contains the main(). It internally calls 2 functions

- void recreate_cfg_objects()
- void setup()

2.2 ipdcGui.c

It is the file that contains all the GUI functions for iPDC

- int isNumber(char *s)
- void destroy (GtkWidget *widget, gpointer udata)
- void display_pdc_detail (GtkButton *widget, gpointer udata)
- void about_ipdc (GtkButton *widget, gpointer udata)
- void ipdc_help (GtkButton *but, gpointer udata)
- void validation_result (char *msg)
- void ipdc_colors ()
- void pdc_details (GtkButton *button, gpointer udata)
- void fill_pdc_details ()
- int validation_pdc_detail (GtkButton *button, gpointer udata)
- void add_pmu (GtkButton *but, gpointer udata)
- int add_pmu_validation (GtkButton *but, gpointer udata)
- void cmd_or_remove_pmu (GtkButton *but, gpointer udata)
- int cmd_or_remove_pmu_validation (GtkButton *but, gpointer udata)
- void add_new_pdc (GtkButton *but, gpointer udata)
- int new_pdc_validation (GtkButton *but, gpointer udata)
- void remove_pdc (GtkButton *but, gpointer udata)
- int remove_pdc_validation (GtkButton *but, gpointer udata)
- void connection_table (GtkButton *but, gpointer udata)

2.3 Functions in recreate.c

- void recreate_cfg_objects()
- void init_cfgparser(unsigned char st[])
- void recreate_Connection_Table()

2.4 Functions in connections.c

- void setup()
- void sigchld_handler()
- void* UL_udp()
- void* UL_tcp()
- void* UL_tcp_connection(void * newfd)
- void PMU_process_UDP(unsigned char *,struct sockaddr_in,int sockfd)
- void PMU_process_TCP(unsigned char tcp_buffer[],int sockfd)

2.5 Functions in new_pmu_or_pdc.c

- int add_PMU(char pmuid[], char ip[], char port[], char protocol[])
- void* connect_pmu_tcp(void *temp)
- void* connect_pmu_udp(void *temp)
- int remove_Lower_Node(char pmuid[], char protocol[])
- void* remove_llnode(void*)
- int put_data_transmission_off(char pmuid[], char protocol[])
- void* data_off_llnode(void* temp)
- int put_data_transmission_on(char pmuid[], char protocol[])
- void* data_on_llnode(void* temp)
- int configuration_request(char pmuid[], char protocol[])
- void* config_request(void* temp)

- int add_PDC(char ip[], char protocol[])
- int remove_PDC(char ip[], char port_num[], char protocol[])
- void display_CT()
- void create_command_frame(int type,int pmuid,char *)
- int checkip(char ip[])

2.6 Functions in parser.c

- void cfgparser(unsigned char [])
- int remove_old_cfg(unsigned char[])
- void dataparser(unsigned char[])
- int check_statword(unsigned char stat[])
- void add_pmuid_from_status_change_list(unsigned char idcode[])
- void remove_id_from_status_change_list(unsigned char idcode[])
- unsigned int to_intconvertor(unsigned char [])
- void long_int_to_ascii_convertor (long int n,unsigned char hex[])
- void copy_cbyc(unsigned char dst[],unsigned char *s,int size)
- int ncmp_cbyc(unsigned char dst[],unsigned char src[],int size)
- void byte_by_byte_copy(unsigned char dst[],unsigned char src[],int index,int n)
- unsigned long int to_long_int_convertor(unsigned char array[])
- uint16_t compute_CRC(unsigned char *message,char length)

2.7 Functions in dallocate.c

- void free_cfgframe_object(struct cfg_frame *cfg)
- void free_dataframe_object(struct data_frame *df)
- void free_2darray(char** array, int x)

2.8 Functions in align_sort.c

- void time_align(struct data_frame *df)
- void assign_df_to_TSB(struct data_frame *df,int index)
- void dispatch(int index)
- void sort_data_inside_TSB(int index)
- void clear_TSB(int index)
- void create_dataframe(int index)
- void create_cfgframe()

2.9 global.h

It contains the mutex variables and other global variables to be used across all the files.

2.10 Detailed Description

2.10.1 recreate.c

- **void recreate_cfg_objects()**
  recreate_cfg_objects() is present in the file recreate.c. When parsing the configuration frame, along with configuration objects creation the configuration frame is also written into file cfg.bin. If ./server aborts the configuration objects in the memory would be lost. So on restarting the ./server this function call would read the file cfg.bin and recreate configuration objects in memory. For this it calls init_cfgparser().

- **void init_cfgparser(unsigned char st[])**
  It is called by recreate_cfg_objects(). The function recreate_cfg_objects() reads file 'cfg.bin' line by line. Each line is a pre-stored cfg objects. For each line read, init_cfgparser(line) is called which creates the objetcs in the memory.

- **void recreate_Connection_Table()**

2.10.2 connections.c

- **void setup()**
  setup() is present in the file connections.c. It creates 2 sockets for UDP and TCP each. Separate ports for each are maintained, UDPPORT for UDP and TCPPORT for TCP so that the PMU can communicate using any communication protocol. It creates 2 threads by calling void* UL_udp and void* UL_tcp. Threads that are created are in NON-DETACHED mode. There is one more thread created by calling void* ADD_PMU_PDC(). This is a prompt to user provided with a set of options like add PMU etc. There is another socket created that binds to a port DBPORT (default 9000). IP specified by the user when the PDC is started. This socket is to send the data to a machine where dbserver is running.

- **void sigchld_handler()**

- **void* UL_udp()**
  Handles UDP connections on port UDPPORT. Responsible for handling command frames from iPDCś on a UDP communication protocol.

- **void* UL_tcp()**
  Accepts TCP connections of iPDC on port TCPPORT.

- **void* UL_tcp_connection(void * newfd)**
  Responsible for handling command frames from iPDCś on a TCP communication protocol.

- **void PMU_process_UDP(unsigned char *,struct sockaddr_in,int sockfd)**
  It checks the type of frame that has been received and calls appropriate parser. If d́ata frameís received, then it calls data_parser(). Based on the return value of the data_parser() take appropriate action such as if stat bit 10 is 1 then send the command frame to PMU to send its configuration frame. If ćonfiguration f́rame is received, then it calls cfgparser(). It also sends the command frame to PMU to start sending its data frames. If ćommandf́rame is received the action to be taken is not handled.

- **void PMU_process_TCP(unsigned char tcp_buffer[],int sockfd)**
  It checks the type of frame that has been received and calls appropriate parser. If d́ata frameís received, then it calls data_parser() present in file parser.c. Based on the return value of the data_parser() take appropriate action such as if stat bit

10 is 1 then send the command frame to PMU to send its configuration frame. If ćonfiguration f́rame is received, then it calls cfgparser() present in file. It also sends the command frame to PMU to start sending its data frames. If ćommand f́rame is received the action to be taken is not handled.

2.10.3 new_pmu_or_pdc.c

- **Explained under General working of iPDC**
  int add_PMU(char pmuid[], char ip[], char port[], char protocol[]), int remove_PDC(), void add_PMU(), void* connect_pmu_tcp(), void* connect_pmu_udp(), void add_PMU_Node(), void remove_Lower_Node(), void* remove_llnode(void*), void put_data_transmission_off(), void* data_off_llnode(void* temp),
  void put_data_transmission_on(), void* data_on_llnode(void* temp), void configuration_request(), void* config_request(void* temp), void display_CT()

- **void create_command_frame(int type,int pmuid,char *)**
  This would create 3 kinds of command frames, Command to send the CFG, Command to send the data ,Command to put off the data transmission.

- **int checkip(char ip[])** This function checks if the Ip address entered by the user is a valid IP address.

2.10.4 parser.c

- **void cfgparser(unsigned char [])**
  Similar to init_cfgparser(frame) except the frame is now the frame that actually comes from PMU over TCP/UDP. It parses and creates objects in memory.

- **int remove_old_cfg(unsigned char[])**
  If a changed cfg arrives at iPDC then, this function repaces the old entry in the file cfg.bin with the new frame.

- **void dataparser(unsigned char[])**
  Parses the data frame and creates data objects in memory. It first checks the configuartion objects that to find a match for a corresponding id. Then separates the fields of the data frame and creates data objects. Since there are no separaters for the data in data frame the corresponding configuration object gives these details. For examples it contains the number of pmu data,phnmr, annmr etc. These numbers can be used to separate the data frame by a fixed number of bytes. For example if cfg objects says phnmr = 2, and format is rectangular for phasor, then we allocate

memory for 2 phasors and separate 16 characters from the existing data frame from the current pointer position(As phasor with rectangular format and assuming data coming hexadecimal form each phasor measurement in the data frame would 8 characters in length).

- **int check_statword(unsigned char stat[])**
  stat word in the data frame is the most important. It indicates the status of the data block in the data frame. Hence we check the bits of of sts word. As per the error indications mentioned in the IEEEC37.118 Standard for each bit in the stat word we make a print on console. It reurn the bit numbers as errors.The programmer has reserved bits 03-00 whose value as 1111 or F́indicates the data has not arived for that PMU. This is set and is useful when a iPDC sends a combined data frame to other iPDC.

- **void add_pmuid_from_status_change_list(unsigned char idcode[])**
  If the error is pertaining to configuration change of PMU/iPDC then we add the its entry to a śtatus_change_pmupdcid ẃhich is a linked list maintaining the ids of all PMU/iPDC under the current iPDC whose status related to configuration has been changed.

- **void remove_id_from_status_change_list(unsigned char idcode[])**
  When a new configuration frame is received and there is call to cfgparser(), there is a call within this cfgparser() for a remove_id_from_status_change_list(char idcode[]). It looks if its idcode is in the list śtatus_change_pmupdcid ΄. If so it removes its entry from the list.

- **unsigned int to_intconvertor(unsigned char [])**
  Converts the binary 2 byte unsigned character value to equivalent int.

- **void long_int_to_ascii_convertor (long int n,unsigned char hex[])**
  Converts the unsigned long int value to a 4 byte unsigned character value.

- **void copy_cbyc(unsigned char dst[],unsigned char *s,int size)**
  Performs copy of size bytes to destination array.

- **int ncmp_cbyc(unsigned char dst[],unsigned char src[],int size)**
  Performs comaprison of size bytes of both arrays.

- **void byte_by_byte_copy(unsigned char dst[],unsigned char src[], int index,int n)**

Performs copy of size bytes to destination array from its index position to index + n.

- **unsigned long int to_long_int_convertor(unsigned char array[])**
  Converts the binary 4 byte unsigned character value to equivalent unsigned long int.
- **uint16_t compute_CRC(unsigned char *message,char length)** Calculates checksum of a frame of size length.

2.10.5 dallocate.c

- **void free_cfgframe_object(struct cfg_frame *cfg)**
  Since we dynamically allocate memory to cfg objetcs we need to free it once our task is done. This function does the same.
- **void free_dataframe_object(struct data_frame *df)**
  Since we dynamically allocate memory to data objetcs we need to free it once our task is done. This function does the same.
- **void free_2darray(char** array, int x)**
  Frees 2D Arrays that were allocated memory using malloc().

2.10.6 align_sort.c

- **void time_align(struct data_frame *df)**
  We use Circular queue to align the data frame objects as per their soc and fracsec. The size of circular queue is defined as MAXTSB. This function finds the corrects TSB[] and calls assign_df_to_TSB() by passing the index of the TSB to which the data frame object df is to be assigned to. Also if all TSB[] buffers are full it calls dispatch(rear), clear_TSB(rear), assign_df_to_TSB(df,rear) in the same order.
- **void assign_df_to_TSB(struct data_frame *df,int index)**
  It assigns df data_frame object to the TSB[index] at the end of list as indicated by struct data_frame *first_data_frame. It traverses the end of the list of data frame objects and then assigns df to the *dnext of last data frame object. Also if the TSB[] is used for the first time it allocates memory to the member variables of the TSB[index] and fills the ídlistẃith the pmu/pdc idś from whom data frames would arrive.

- **void dispatch(int index)**
  It internally calls void sort_data_inside_TSB(index), void create_dataframe(index), clear_TSB(index) in the same order.
- **void sort_data_inside_TSB(int index)**
  This function will sort the data frame object LL as per the idś in ídlistĹL.
- **void clear_TSB(int index)**
  This will clear TSB[index] member variables to 0́ ́.
- **void create_dataframe(int index)**
  This will create combined data frame to be sent to a Destination PDC when a command frame is received.
- **void create_cfgframe()**
  This will create combined configuration frame to be sent to a Destination PDC when a command frame is received.

2.11 Data Structures Used

- **Data Structure for Configuration Frame**
  cfg_frame, for_each_pmu, channel_names, dgnames, format are the data structures defined for the configuration frame. ́format ́is an additional data structure that has been defined. It simplifies the task of distinguishing the measurements as floating/fixed, polar/rectangular. As per IEEEC37.118 Synchrophasor Standard, format field in configuration frame is 2 bytes (4 characters in hexadecimal).
  Bits 15-4 Unused
  Bit 3:
  0 = FREQ/DFREQ 16 -bit integer,
  1 = floating point
  Bit 2:
  0 = analogs 16 -bit integer,
  1 = floating point
  Bit 1:
  0 = phasors 16 -bit integer,
  1 = floating point
  Bit 0:
  0 = phasor rectangular,
  1 = phasor rectangular

- **Data Structure for Data Frame**
  data_frame and data_for_each_pmu are the data structures defined for data frames.

- **Data Structure to store IDś of those PMU/iPDC whose CFG has changed**
  ́status_change_pmupdcid ́the data structure defined to maintain the idcodes of PMU/iPDC whose configuration has been changed. Id remaining in the memory till new Configuration object for that idcode arrives.

- **Data Structure to store Source Device Details**
  Lower_Layer_Details is the data structure defined to maintain the the details of source devices from which iPDC receives the data.

- **Data Structure to store Destination Device Details**
  Upper_Layer_Details is the data structure defined to maintain the the details of destination devices to which iPDC sends the data.

- **Data Structure for Time Stamp Buffer**
  TimeStampBuffer and pmupdc_id_list are the data structures defined to be used in time aligning and sorting of data frames.

```
struct  cfg_frame {

        unsigned char *framesize;
        unsigned char *idcode;
        unsigned char *soc;
        unsigned char *fracsec;
        unsigned char *time_base;
        unsigned char *num_pmu;
        struct for_each_pmu **pmu;
        unsigned char *data_rate;
        struct  cfg_frame *cfgnext;

}*cfgfirst;

struct for_each_pmu{

        unsigned char *stn;
        unsigned char *idcode;
        unsigned char *data_format;
        struct format *fmt;
        unsigned char *phnmr;
        unsigned char *annmr;
        unsigned char *dgnmr;
        struct channel_names *cnext;
        unsigned char **phunit;
        unsigned char **anunit;
        unsigned char **dgunit;
        unsigned char *fnom;
        unsigned char *cfg_cnt;
};

struct channel_names {

        unsigned char **phnames;
        unsigned char **angnames;
        struct dgnames *first;
};

struct dgnames {

        unsigned char **dgn;
        struct dgnames *dg_next;
};

struct format{

        unsigned char freq;
        unsigned char analog;
        unsigned char phasor;
        unsigned char polar;
};
```

Figure 1: **Configuration Frame.**

```
struct data_frame {

        unsigned char *framesize;
        unsigned char *idcode;
        unsigned char *soc;
        unsigned char *fracsec;
        int num_pmu;
        struct data_for_each_pmu **dpmu;
        struct data_frame *dnext;
};

struct data_for_each_pmu {

        unsigned char *stat;
        int phnmr;
        int annmr;
        int dgnmr;
        struct format *fmt;
        unsigned char **phasors;
        unsigned char **analog;
        unsigned char *freq;
        unsigned char *dfreq;
        unsigned char **digital;
};
```

Figure 2: **Data Frame.**

```
struct status_change_pmupdcid {

        char idcode[5];
        struct status_change_pmupdcid *pmuid_next;

}*root_pmuid;
```

Figure 3: **PMU Status Change.**

```
struct Lower_Layer_Details {

        unsigned int pmuid;
        char ip[16];
        int port;
        char protocol[4];
        int sockfd;
        int up; //used only in tcp
        struct sockaddr_in llpmu_addr;
        pthread_t thread_id;
        int data_transmission_off;
        int pmu_remove;
        int request_cfg_frame;
        struct Lower_Layer_Details *next;
        struct Lower_Layer_Details *prev;

}*LLfirst,*LLlast;
```

Figure 4: **Source Device Details**

```
struct Upper_Layer_Details {

        char ip[16];
        int port;
        char protocol[4];
        int sockfd;
        struct sockaddr_in pdc_addr;
        int config_change;
        int UL_upper_pdc_cfgsent;
        int UL_data_transmission_off;
        int address_set;
        struct Upper_Layer_Details *next;
        struct Upper_Layer_Details *prev;

}*ULfirst,*ULlast;
```

Figure 5: **Destination Device Details**

```
struct TimeStampBuffer {

        char *soc;
        char *fracsec;
        struct pmupdc_id_list *idlist;
        struct data_frame *first_data_frame;

}TSB[MAXTSB];

struct pmupdc_id_list {

        char *idcode;
        int num_pmu;
        struct pmupdc_id_list *nextid;
};
```

Figure 6: **Time Stamp Buffer**

3 Source Code Documentation of DBServer

The code is distributed in the following files

- dbServer.c
- recreate.c
- recreate.h
- connections.c
- connections.h
- parser.c
- parser.h
- dallocate.c
- dallocate.h
- global.h

3.1 dbServer.c

It is the file that contains the main(). It internally calls 2 functions

- void recreate_cfg_objects()
- void setup()

3.2 Functions in recreate.c

- void recreate_cfg_objects()
- void init_cfgparser(unsigned char st[])

3.3 Functions in connections.c

- void setup()
- void DB_udp()
- void* DB_udphandler(void * udp_BUF)
- void DB_process_UDP(unsigned char* udp_BUF)

3.4 Functions in parser.c

- void cfgparser(unsigned char [])
- void cfginsert(struct cfg_frame *)
- int remove_old_cfg(unsigned char[])
- void dataparser(unsigned char[])
- int check_statword(unsigned char stat[])
- unsigned int to_intconvertor(unsigned char [])
- unsigned long int to_long_int_convertor(unsigned char array[])
- float decode_ieee_single(const void *v)
- void copy_cbyc(unsigned char dst[],unsigned char *s,int size)
- int ncmp_cbyc(unsigned char dst[],unsigned char src[],int size)

3.5 Functions in dallocate.c

- void free_cfgframe_object(struct cfg_frame *cfg)
- void free_dataframe_object(struct data_frame *df)
- void free_2darray(char** array, int x)

3.6 global.h

It contains the mutex variables and other global variables to be used across all the files.

3.7 Detailed Description

3.7.1 recreate.c

Same as described above in recreate.c

3.7.2 connections.c

- void setup()
  This function created MySQL database connection and also binds to UDP port 9000.

- void DB_udp()
  It receives the udp data from iPDC, creates a thread by calling DB_udphandler() and passes the data to it.

- void* DB_udphandler(void * udp_BUF) Internally it calls DB_process_UDP().

- void DB_process_UDP(unsigned char* udp_BUF)
  It checks the type of frame that has been received and calls appropriate parser. If ́data frame ís received, then it calls data_parser(). If ćonfiguration f́rame is received, then it calls cfgparser().

3.7.3 parser.c

- **void cfgparser(unsigned char [])**
  Similar to init_cfgparser(frame) it parses and creates objects in memory.

- **void cfginsert(struct cfg_frame *)**
  Insert/Update the configuration frame in the configuration tables.

- **int remove_old_cfg(unsigned char[])**
  If a changed cfg arrives at iPDC then for a PMU, this function repaces the old entry in the file cfg.bin with the new frame.

- **void dataparser(unsigned char[])**
  Parse the data frame an insert into the data tables of iPDC database.

- **int check_statword(unsigned char stat[])**
  stat word in the data frame is the most important. It indicates the status of the data block in the data frame. Hence we check the bits of of sts word. As per the error indications mentioned in the IEEEC37.118 Standard for each bit in the stat word we make a print on console. It reurn the bit numbers as errors.The programmer has reserved bits 03-00 whose value as 1111 or F́índicates the data has not arived for that PMU. This is set and is useful when a PDC sends a combined data frame to other iPDC.

- **unsigned int to_intconvertor(unsigned char [])**
  Converts the binary 2 byte unsigned character value to equivalent int.
- **unsigned long int to_long_int_convertor(unsigned char array[])**
  Converts the binary 4 byte unsigned character value to equivalent long int.
- **float decode_ieee_single(const void *v)**
  Converts the binary 4 byte unsigned character value to equivalent float value.
- **void copy_cbyc(unsigned char dst[],unsigned char *s,int size)**
  Performs copy of size bytes to destination array.
- **int ncmp_cbyc(unsigned char dst[],unsigned char src[],int size)**
  Performs comaprison of size bytes of both arrays.

3.7.4 dallocate.c

Same as describe above.

3.8 Data Structures Used

- **Data Structure for Configuration Frame**
  cfg_frame, for_each_pmu, channel_names, dgnames, format are the data structures defined for the configuration frame. ́format ́is an additional data structure that has been defined. It simplifies the task of distinguishing the measurements as floating/fixed, polar/rectangular. As per IEEEC37.118 Synchrophasor Standard, format field in configuration frame is 2 bytes (4 characters in hexadecimal).
  Bits 15-4 Unused
  Bit 3:
    0 = FREQ/DFREQ 16 -bit integer,
    1 = floating point
  Bit 2:
    0 = analogs 16 -bit integer,
    1 = floating point
  Bit 1:
    0 = phasors 16 -bit integer,
    1 = floating point
  Bit 0:
    0 = phasor rectangular,

1 = phasor rectangular

- **Data Structure for Data Frame**
  data_frame and data_for_each_pmu are the data structures defined for data frames.

```
struct  cfg_frame {

        unsigned int framesize;
        unsigned int idcode;
        unsigned long int soc;
        unsigned long int fracsec;
        unsigned long int time_base;
        unsigned int num_pmu;
        struct for_each_pmu **pmu;
        unsigned int data_rate;
        struct  cfg_frame *cfgnext;

}*cfgfirst;

struct for_each_pmu{

        unsigned char stn[17];
        unsigned int idcode;
        char data_format[3];
        struct format *fmt;
        unsigned int phnmr;
        unsigned int annmr;
        unsigned int dgnmr;
        struct channel_names *cnext;
        float **phunit;
        float **anunit;
        unsigned char **dgunit;
        unsigned int fnom;
        unsigned int cfg_cnt;
};

struct channel_names {

        unsigned char **phnames;
        unsigned char **angnames;
        struct dgnames *first;
};

struct dgnames {

       unsigned char **dgn;
       struct dgnames *dg_next;
};

struct format{

        char freq;
        char analog;
        char phasor;
        char polar;
};
```

Figure 7: **DBServer Configuration Frame**

```
struct data_frame {

        unsigned char *framesize;
        unsigned char *idcode;
        unsigned char *soc;
        unsigned char *fracsec;
        int num_pmu;
        struct data_for_each_pmu **dpmu;
        struct data_frame *dnext;
};

struct data_for_each_pmu {

        unsigned char *stat;
        int phnmr;
        int annmr;
        int dgnmr;
        struct format *fmt;
        unsigned char **phasors;
        unsigned char **analog;
        unsigned char *freq;
        unsigned char *dfreq;
        unsigned char **digital;
};
```

Figure 8: **DBServer Data Frame**

4 iPDC Database

Script db.sql gives the DDL statements to create the schema. In the MySQL database named openPDC there are 9 tables named:-

- **MAIN_CFG_TABLE**
- **SUB_CFG_TABLE**
- **PHASOR**
- **ANALOG**
- **DIGITAL**
- **PHASOR_MEASUREMENTS**
- **ANALOG_MEASUREMENTS**
- **FREQUENCY_MEASUREMENTS**
- **DIGITAL_MEASUREMENTS**

Configuration frames are stored in MAIN_CFG_TABLE, SUB_CFG_TABLE, PHASOR, ANALOG, DIGITAL. Data frames are stored in PHASOR_MEASUREMENTS, ANALOG_MEASUREMENTS, FREQUENCY_MEASUREMENTS, DIGITAL_MEASUREMENTS. The schema diagram shows the functional dependencies among the tables.

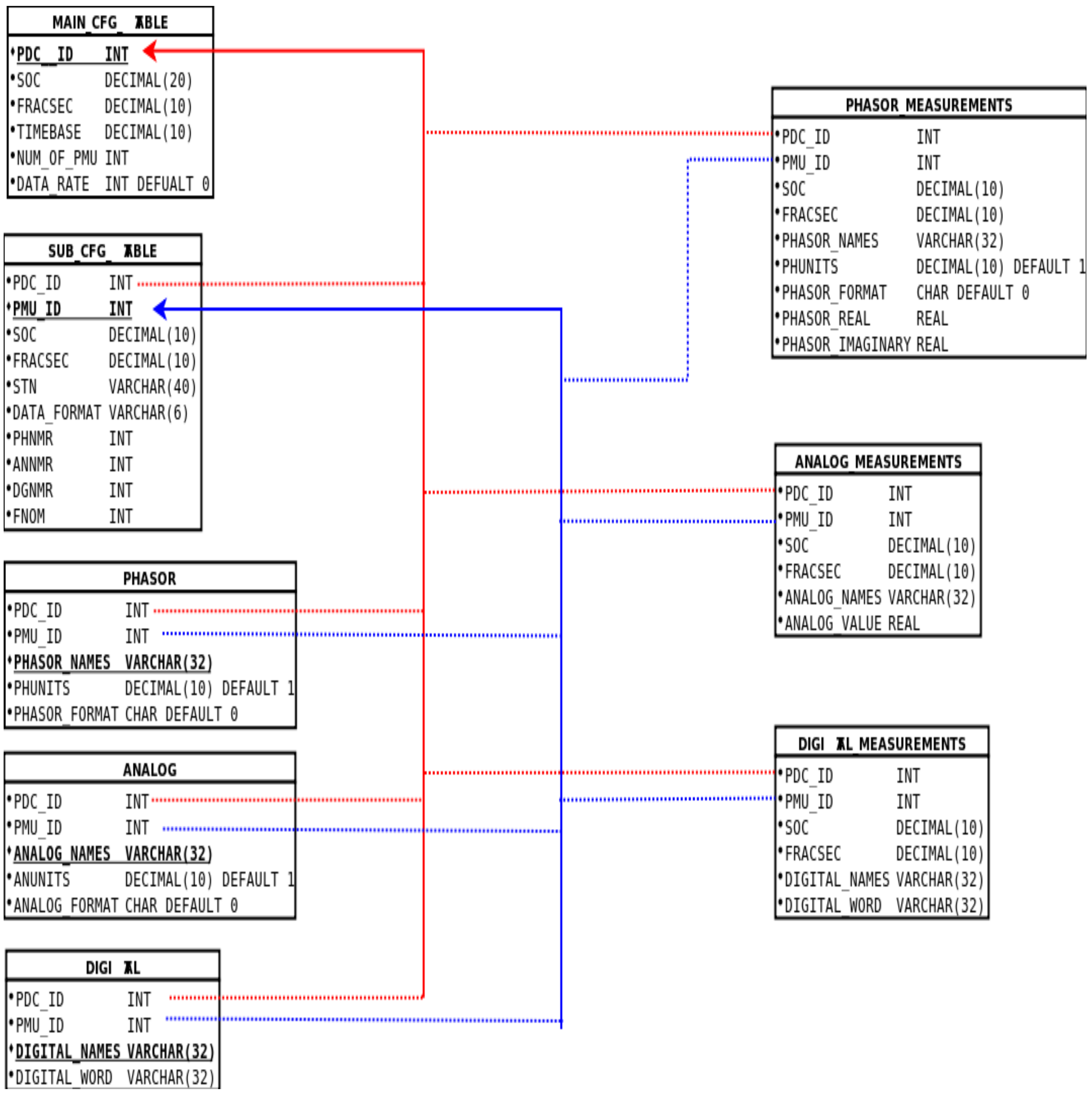

Figure 9: iPDC Database

5 General Working of PMU Simulator

PMU Simulator act as servers when communicating with PDC whom they are sending the data/configuration frames. The PDC receiving data in this case acts as a client. This pattern will be repeated in the WAMS topology with one peer acting as server and its counterpart a client.

PMU Simulator acting as a server and binding a port for UDP or TCP. It would be listening for UDP connections on UDPPORT or for TCP connections on TCPPORT. The PMU Simulator receives the command frames from PDC and send the configuration frame and data frames to PDC. Both the communicating peers authenticate each other through ID Code.

When the PMU Simulator starts for the first time the user is prompted to enter PMU Setup details Port number and Protocol. The ports enable PMU Simulator to receive requests from PDC and to send the data and configuration frames. PMU Simulator generates the data and configuration frames in IEEE C37.118 format.

The user is provided with the following options at PMU Simulator

- Enter PMU Setup
- PMU Configuration Setup
- Header Frame Generation
- Update PMU Configuration
- View PMU Details
- PMU Exit

5.1 Enter PMU Setup

This is the first pop up window that would ask the user to enter the PMU setup information. It includes port on which the PMU would receive command frame requests from PDC. The data and configuration frames would be sent on this port. Communication protocol (UDP/TCP) also need to be entered. This details would be save in the file PmuServer.txt, and at the beginning of PMU restart it would read the entries and start PMU Server.

This functionality is distributed in the following functions :- *void pmu_server (), int validation_pmu_server (GtkButton *button, gpointer udata)* from the file **PmuGui.c** and *void start_server()* from file **ServerFunction.c**.

5.2 Create Configuration Frame

After finishing the PMU Setup, PMU configuration objects would be craeted if the file cfg2.bin available, otherwise user has to enter the configuration details. In order to complete the configuration frame user has to fill details through multiple pop up windows, and finally store in the file cfg2.bin. change.bin file also created and write 0 in it.

This functionality is distributed in the following functions :- *void cfg_create_function (GtkWidget *widget, gpointer udata), void channel_names_for_phasor (), void channel_names_for_analog (), void channel_names_for_digital (), int validation_cfg_create ()* from the file **CfgGuiFunctions.c** and *int create_cfg()* from file **CfgFunction.c**.

5.3 Header Frame Generation

User can enter details about algorithm, scaling, filtering and other informantion about PMU, and this will store in the file header.bin. Header frame also followse the IEEE C37.118 Standard.

This functionality is distributed in the following functions :- *void hdr_create_function (GtkWidget *widget, gpointer udata), void validation_hdr_frm (GtkWidget *widget, gpointer udata)* from the file **CfgGuiFunctions.c**
and *void header_frm_gen(int len)* from file **CfgFunction.c**.

5.4 Update PMU Configuration

User can change the STAT Word bit in data frame by using the Update PMU Configuration. It can also modifies in the configuration frame to add or remove channels.

This functionality is distributed in the following functions :- *void Change_pmu_configuration (GtkWidget *widget, gpointer udata), void new_phasor_num(GtkWidget *widget, gpointer udata), void new_analog_num (), void remove_phasor_num (GtkWidget *widget, gpointer udata), void remove_analog_num (), void change_data_rate_option(), int final_cfg_update_call (GtkButton *but, gpointer udata)* from the file **CfgGuiFunctions.c**, and *void reconfig_cfg_CC()* from the file **CfgFunction.c**.

5.5 View PMU Details

This would display the PMU Server details and latest PMU Configuration details to user.

This functionality is distributed in the following functions :- *void show_pmu_details (GtkWidget *widget, gpointer udata),* from the file **CfgFunction.c**.

5.6 PMU Exit

It will close all the threads and processes of PMU Simulator and terminate it.

This functionality is distributed in the following functions :- *void destroy (GtkWidget *widget, gpointer udata)* from the file **PmuGui.c**.

6 Source Code Documentation of PMU Simulator

The code is distributed in the following files

- pmu.c
- PmuGui.c
- PmuGui.h
- ServerFunction.c

- ServerFunction.h
- CfgGuiFunctions.c
- CfgGuiFunctions.h
- CfgFunction.c
- CfgFunction.h
- function.c
- function.h

6.1 pmu.c

It is the file that contains the main(). It internally calls 3 functions

- void pmu_server ()
- void create_cfg ()
- void pmu_colors()

6.2 Functions in PmuGui.c

- int checkip(char ip[])
- int isNumber(char *s)
- void destroy (GtkWidget *widget, gpointer udata)
- void destroy1 (GtkWidget *widget, gpointer udata)
- void about_pmu (GtkButton *widget, gpointer udata)
- void Pmu_Help (GtkButton *but, gpointer udata)
- void validation_result (char *msg)
- void pmu_colors()

6.3 Functions in ServerFunction.c

- void frame_size()
- void generate_data_frame()
- void* udp_send_data()
- void* pmu_udp()
- void* tcp_send_data(void * newfd)
- void* new_pmu_tcp(void * nfd)
- void* pmu_tcp()
- void start_server()

6.4 Functions in CfgGuiFunctions.c

- void cfg_create_function (GtkWidget *widget, gpointer udata)
- int validation_cfg_create ()
- void channel_names_for_phasor ()
- int validation_phasor_names(GtkWidget *widget, gpointer udata)
- void channel_names_for_analog ()
- int validation_analog_names(GtkWidget *widget, gpointer udata)
- void channel_names_for_digital ()
- int validation_digital_names(GtkWidget *widget, gpointer udata)
- void final_cfg_create ()
- void cfg_chng_options(GtkWidget *widget, gpointer udata)
- void cfg_STAT_change (GtkWidget *widget, gpointer udata)
- void Change_pmu_configuration(GtkWidget *widget, gpointer udata)
- void new_phasor_num(GtkWidget *widget, gpointer udata)
- void new_analog_num ()

- int validation_ph_an_num (GtkButton *but, gpointer udata)
- void new_channel_names_for_phasor ()
- int validation_new_phasor_names(GtkWidget *widget, gpointer udata)
- void new_channel_names_for_analog ()
- int validation_new_analog_names(GtkWidget *widget, gpointer udata)
- void change_data_rate_option()
- void enter_new_data_rate(GtkWidget *widget, gpointer udata)
- int validation_new_data_rate (GtkButton *but, gpointer udata)
- void remove_phasor_num (GtkWidget *widget, gpointer udata)
- void remove_analog_num ()
- int validation_remove_ph_an_num (GtkButton *but, gpointer udata)
- void hdr_create_function (GtkWidget *widget, gpointer udata)
- void validation_hdr_frm (GtkWidget *widget, gpointer udata)
- int final_cfg_update_call (GtkButton *but, gpointer udata)

6.5 Functions in CfgFunction.c

- void header_frm_gen(int len)
- void reconfig_cfg_CC()
- void show_pmu_details (GtkWidget *widget, gpointer udata)
- int create_cfg()

6.6 Functions in function.c

- void B_copy(unsigned char main[], unsigned char tmp[], int ind, int n)
- void H2S(char a[], unsigned char temp_6[])
- void i2c (int t, unsigned char temp[])
- void li2c (long int t1, unsigned char temp_1[])

- void f2c (float f, unsigned char temp_4[])
- int c2i (unsigned char temp_2[])
- long int c2li (unsigned char temp_3[])
- uint16_t compute_CRC (unsigned char *message,char length)
- void sigchld_handler (int s)