SCS

Sirveen Control Systems Pvt Ltd

2016

PHMU Design Document

Sirveen Control Systems

24-Feb-16

Revision History

Revision	Date	Author	Review	Review Description
1.0	24-Feb-16	Md Shahid		Initial revision

Definitions

Term	Definition
PHMU	Point Health Monitoring Unit
CMU	Central Monitoring Unit

References

Following are the reference document used in preparing this document.

Ser No	Documents
1	PHMU System Requirement
2	PHUM Hardware Schematics
3	Driver Frame Work for STM32 based boards
4	Software Engineering A Practitioner’s Approach by Roger S Pressman

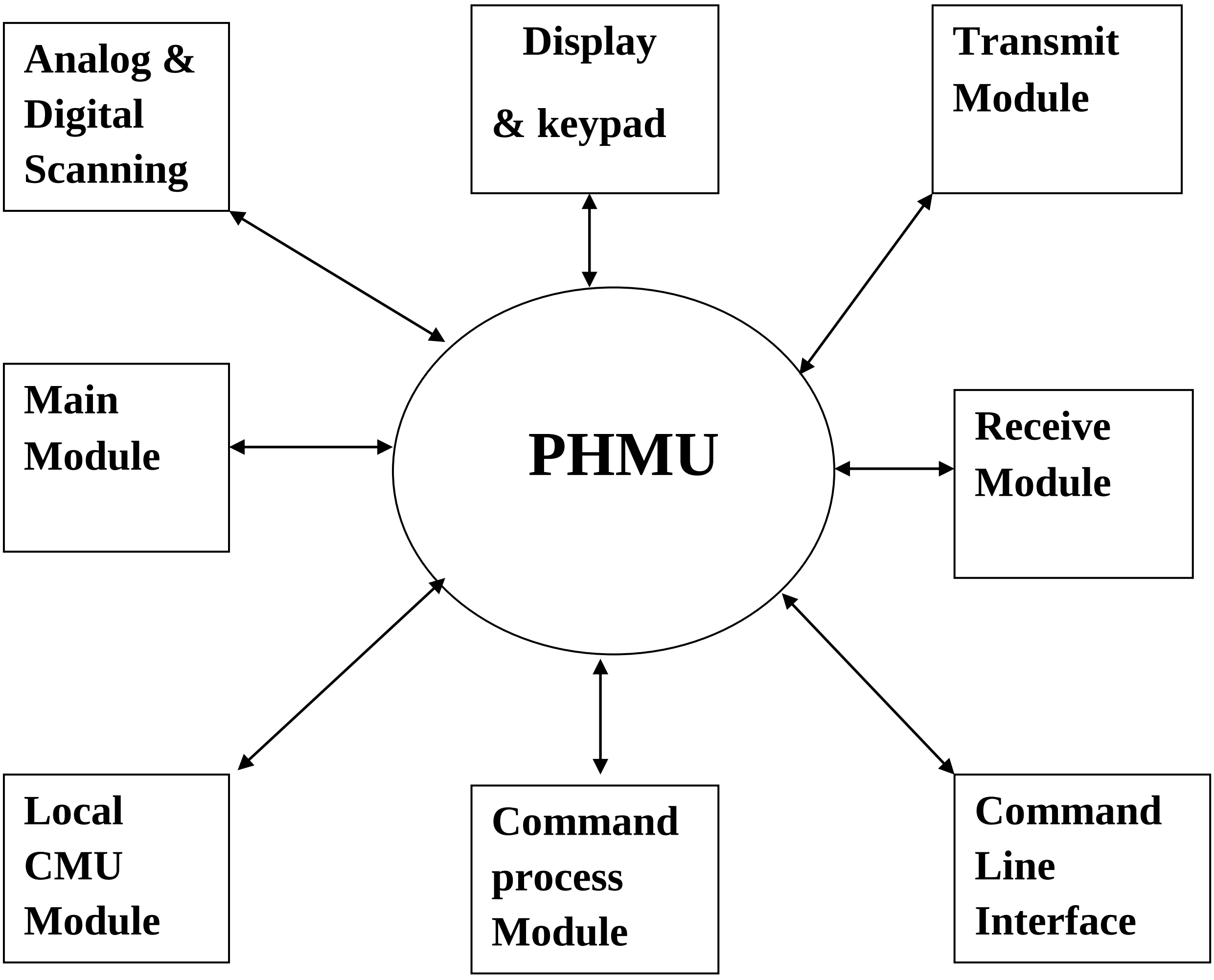
Introduction

This document describes the software design for Point health Monitoring Unit also called as PHMU.

Data Flow Diagram

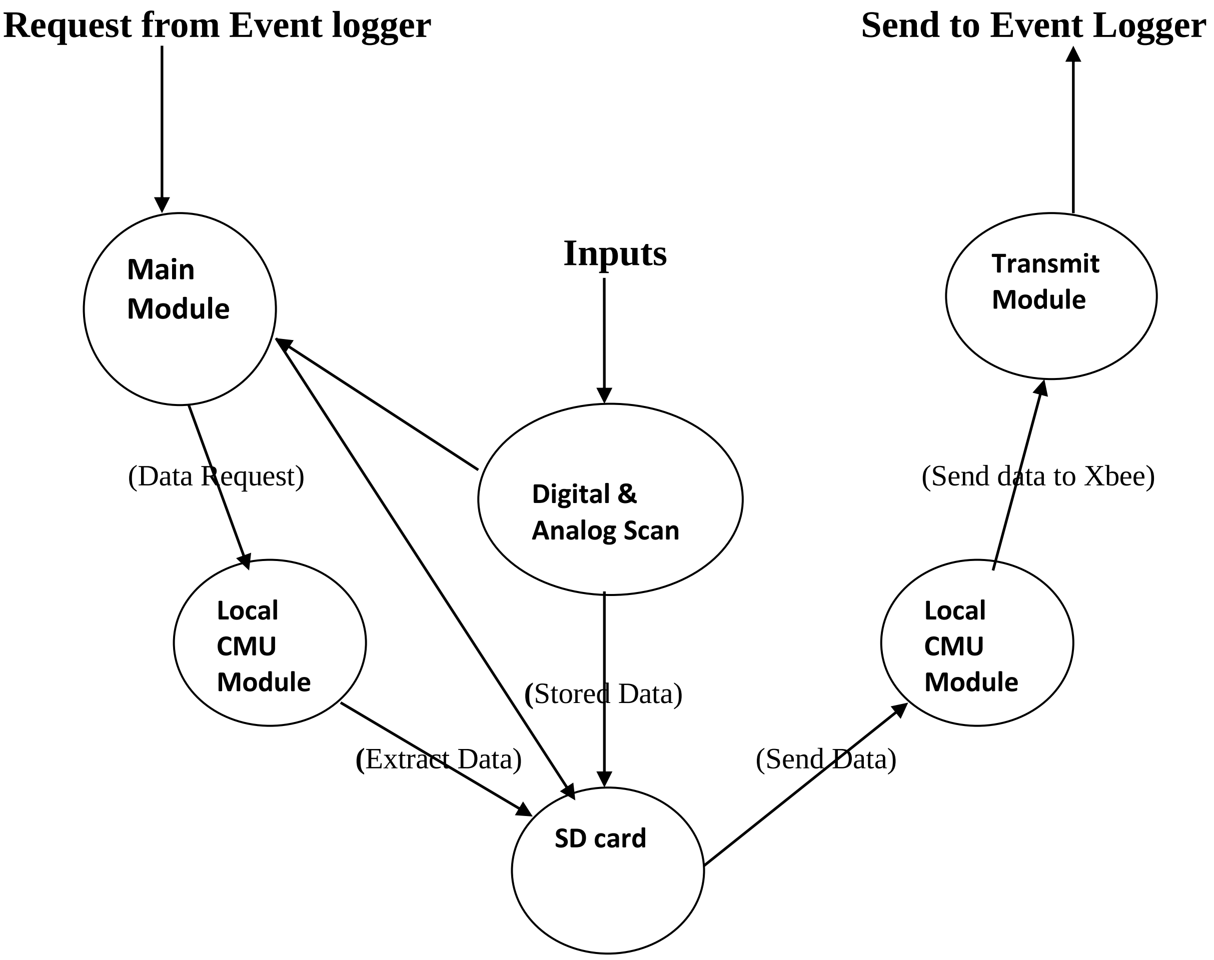
Level 0 data flow diagram (DFD)

Following is the level 0 data flow diagram (DFD), also called context diagram. This diagram clearly shows the external entities with which the PHMU software is interacting.



Level 1 data flow diagram (DFD)

In the level 0, complete PHMU software is shown as a single processing unit by representing as circle. In this level 1, DFD, the internal processing modules are identified.



Modules in PHMU Software

1. **Analog and Digital Scanning module**
2. **Transmit module**
3. **Receive module**
4. **Display User Interface module**
5. **Command Line interface module**
6. **Command process module**
7. **Local CMU module**
8. **Main module**

1. Analog and Digital Scanning module

This module is responsible for scanning of Digital Input, Internal ADC and Externally Connected Analog Pins with GPIO pins. Every one scanned the data for particular time span.

Digital Scan-It reads the GPIO (E, G) pins create packets and store the data into SD card. Its compare current data bit by bit with previous data if changed found then creates packet and store in to SD card.

Analog Scan-It reads internal analog channels get the average data and stored in buffer creates packet and store in to SD card.

Analog External –It reads the GPIO pins generate the interrupt on rising edge, calculate the no of pulses and stored in buffer creates packet and store in to SD card.

2. Transmit module

This module contains functions that implement transmission logic. This involves transmitting log records over both the channels. Also transmitting records received over one channel over the other.

3. Receive module

This module contains functions that implement receive logic. This involves process, validate, extract Packet, and check For Duplicate.

4. Display User Interface module

This module is mainly responsible for communication with display & keypad unit which is connected with the serial port RS232. This is mainly responsible for communicating with different modules and to display requested data.

5. Command Line Interface module

This module is mainly responsible for providing the command line interface through the serial port. This interface is meant for the system engineers for the status monitor, diagnostics and debugging. This

Interface provides commands to display the PHMU performance statistics, to log the PHMU data periodically for later analysis.

6. Command Process Module

This module is mainly responsible for processes the command and respond with ACK and executes command and write to fd.

7. Local CMU module

This module is responsible for mainly communication between Event logger and PHMU.It receive data request get the data from Sd card and send back to event logger. In this process so many function is implemented.

8. Main module

This module is mainly responsible for integrating all the modules into a single application.

1. Analog and Digital Scanning module

This module is responsible for scanning of Digital Input, Internal ADC and Externally Connected Analog Pins with GPIO pins. Every one scanned the data for particular time span.

This module is implemented in ‘digscan.c’ file.

Global Variables:

```
uint32_t flagRead=0,timerFlag=0,flagScanOver=0,vtofDataReady=0;
uint32_t AnalogFreq=0,AnalogFreq1=0,AnalogFreq2=0,AnalogFreq3=0;
uint32_t AnalogFreq4=0,AnalogFreq5=0,AnalogFreq6=0,AnalogFreq7=0;
int32_t capture[8]={0};
float voltage=0;
```

Data Types

```
struct adcData
{
  uint16_t chnData[MAX_AVG_VAL];
  int curIx;
  unsigned char stat;
};
```

Data Definitions

```
struct adcData AdcInfo[10];
```

Global Functions:

digital_Init()

Prototype	void digital_Init(void)
Parameter	void
Return Value	void
Description	configure the GPIOs (E,G) pin for digital Input

digital_IO_Scan

Prototype	void digital_IO_Scan(uint8_t *buf, uint32_t noofbytes)
Parameter	Integer buffer pointer and no of bytes
Return Value	void
Description	Read input data on GPIO E, G pins and stored in buffer

getAdcval

Prototype	uint16_t getAdcval(struct adcData *padc,uint16_t val)
Parameter	Structure pointer of adcData and val
Return Value	unsigned integer
Description	Get ADC value make average and retun

printAdcval

Prototype	void printAdcval(unsigned int ix)
Parameter	Integer ix
Return Value	void
Description	print Adc Data

readandprocDIdata

Prototype	void readandprocDIdata(void)
Parameter	None
Return Value	None
Description	Read and process digital input data

ADCInit()

Prototype	void ADCInit()
Parameter	None
Return Value	None
Description	Initialize the PA-4,5,6,7 PC-0,1,2 & PB-0,1 pins as ADC3

analog_IO_Scan

Prototype	uint32_t analog_IO_Scan(uint8_t *buf, uint32_t noofchn)
Parameter	Integer pointer and no of channels
Return Value	Return 0 if Success
Description	Scan Analog Inputs

vtofInputProcess1

Prototype	void vtofInputProcess1(void)
Parameter	Integer pointer and no of channels
Return Value	None
Description	To print the buffer

readandprocAIdata

Prototype	uint8_t readandprocAIdata(void
Parameter	None
Return Value	Integer
Description	To print the buffer

EXTI15_10_IRQHandler

Prototype	void EXTI15_10_IRQHandler(void)
Parameter	None
Return Value	None
Description	IRQ handler for external interrupt

EXTILine15_10_Config

Prototype	void EXTILine15_10_Config(void)
Parameter	None
Return Value	None
Description	External interrupt pin configuration

EXTILine9_5_Config

Prototype	void EXTILine9_5_Config(void)
Parameter	None
Return Value	None
Description	External interrupt pin configuration

EXTI19_5_IRQHandler

Prototype	void EXTI9_5_IRQHandler(void)
Parameter	None
Return Value	None
Description	IRQ handler for external interrupt

vtofInputProcess(void)

Prototype	void vtofInputProcess(void)
Parameter	None
Return Value	None
Description	Voltage to frequency Process

vtofscan(void)

Prototype	uint8_t vtofscan(void)
Parameter	None
Return Value	Integer
Description	Voltage to frequency scanning

readVTOFdata

Prototype	void readVTOFdata(uint8_t *vtofData,uint8_t len)
Parameter	Integer pointer and total length
Return Value	None
Description	Read Voltage to frequency data

2. Transmit module

This file contains functions that implement transmission logic. This involves transmitting log records over both the channels. Also transmitting records received over one channel over the other.
File: txproc.c

Global Functions:

getRecordsCnts

Prototype	int getRecordsCnts(struct channelData *pcd, int *lrCnt, int *crCnt)
Parameter	pointer to struct channelData,integer pointer lrCnt and crCnt
Return Value	SUCCESS
Description	Get records count

sendTxBatchRecs

Prototype	int sendTxBatchRecs(struct channelData *pcd, struct txRecBatch *ptxbch)
Parameter	pointer to struct channelData, structure txRecBatch pointer
Return Value	0 if success
Description	Writes the packets present in the given batch to serial channel

sendTxBatch

Prototype	int sendTxBatch(struct channelData *pcd, struct txRecBatch *ptxbch)
Parameter	pointer to struct channelData, structure txRecBatch pointer
Return Value	0 if success
Description	Writes the packets present in the given batch to serial channel

resendTxBatch

Prototype	int resendTxBatch(struct channelData *pcd, struct txRecBatch *ptxbch)
Parameter	pointer to struct channelData, structure txRecBatch pointer
Return Value	0 if success
Description	Resend the packets present in the given batch to serial channel

startSendingNewTxBatch

Prototype	int startSendingNewTxBatch(struct channelData *pcd, i32_t lrCnt, i32_t crCnt)
Parameter	pointer to struct channelData,integer lrCnt and crCnt
Return Value	if success Tx_done
Description	Start Sending New TxBatch

restartRetxTimers

Prototype	i32_t restartRetxTimers(struct channelData *pcd)
Parameter	pointer to struct channelData
Return Value	0 if success
Description	Restart retransmit Timer

processRetransmits

Prototype	i32_t processRetransmits(struct channelData *pcd)
Parameter	pointer to struct channelData
Return Value	RETX_DONE if success
Description	looks for retransmit timer expiry in every tx batch.If timer expires

txProcessChannel

Prototype	i32_t txProcessChannel(struct channelData *pcd)
Parameter	pointer to struct channelData
Return Value	SUCESS if successed
Description	Transmit and process the channel data

compareAckWithTxBatches

Prototype	i32_t compareAckWithTxBatches(struct channelData *pcd, ui8_t *ackid)
Parameter	pointer to struct channelData,integer pointer
Return Value	SUCESS if successed
Description	gives given ack number with the every pending record.

processAckPkt

Prototype	i32_t processAckPkt(struct channelData *pcd)
Parameter	pointer to struct channelData
Return Value	0 if success
Description	Top level functions for processing ACK packet.

dispTxBatchStat()

Prototype	int dispTxBatchStat()
Parameter	None
Return Value	0 if success
Description	This is a CLI (command line user interface) function for displaying the status and statistics transmission data structures

3. Receive module

This module contains functions that implement receive logic. This involves process, validate, extract Packet, and check For Duplicate.

File: rxproc.c

Global Functions:

readSerialData

Prototype i32_t readSerialData(struct channelData *pcd)
Parameter pointer to struct channelData
Return Value SUCCESS if successed
Description Reads data from channel like A or B, and puts into cirque

extractPacket

Prototype i32_t extractpacket(struct channelData *pcd)
Parameter pointer to struct channelData
Return Value SUCCESS if successed
Description Extract message from the rxq buffer and copy complete packet to 'pktbuf' to process

validatePacket

Prototype i32_t validatePacket(struct channelData *pcd)
Parameter pointer to struct channelData
Return Value SUCCESS if successed
Description validate the buffer that it is ack,command and frame packet

getChannelRec2

Prototype i32_t getChannelRec2(struct channelData *pcd,i32_t chnId, ui8_t *recbuf)
Parameter pointer to struct channelData,Integer chnId,integer pointer to buffer
Return Value SUCCESS if successed
Description gets Channel Records

getAvlChnRecs

Prototype i32_t getChannelRec2(struct channelData *pcd,i32_t chnId, ui8_t *recbuf)
Parameter pointer to struct channelData
Return Value No of records count if successed
Description getAvlChnRecs(

getChannelRec2

Prototype i32_t getChannelRec2(struct channelData *pcd,i32_t chnId, ui8_t *recbuf)
Parameter pointer to struct channelData,Integer chnId,integer pointer to buffer
Return Value SUCCESS if successed
Description gets Channel Records

processPacket

Prototype i32_t processPacket(struct channelData *pcd)

Parameter pointer to struct channelData
Return Value return Integer if successed
Description process Packet,packet can be ack,data and command

processDataPkt

Prototype	i32_t processDataPacket(struct channelData *pcd)
Parameter	pointer to struct channelData
Return Value	return Integer if successed
Description	processDataPkt for C-PORT, packet can be ack,data and command

checkForDuplicate

Prototype	i32_t checkForDuplicate(struct channelData *pcd, i32_t fullID)
Parameter	Pointer to ChannelData, and fullID (dlID and serial no. forms fullID
Return Value	SUCCESS if successed
Description	Returns SUCCESS, if the Rx data is a duplicate.

getRecordFullID

Prototype	i32_t getRecordFullID(struct channelData *pcd)
Parameter	Pointer to ChannelData
Return Value	returns an integer consisting dlID, and serial no. forms a fullID of data packet.
Description	Get full Record Id

ackForRxPackets

Prototype	i32_t ackForRxPackets(struct channelData* pcd)
Parameter	Pointer to ChannelData
Return Value	SUCCESS if successes
Description	Frames ACK packet for the last three Rx data packets and write to chnFd.

processPendAckTmout

Prototype	i32_t ackForRxPackets(struct channelData* pcd)
Parameter	Pointer to ChannelData
Return Value	SUCCESS if successes
Description	process Pend Ack Time out

sendBuffullCmd

Prototype	i32_t sendBuffullCmd(struct channelData* pcd)
Parameter	Pointer to ChannelData
Return Value	SUCCESS if successes
Description	command for full buffer

sendBuffreeCmd

Prototype	i32_t sendBuffreeCmd(struct channelData* pcd)
Parameter	Pointer to ChannelData
Return Value	SUCCESS if successes
Description	command for free buffer

rxProcessChannel

Prototype	int rxProcessChannel(struct channelData *pcd)
Parameter	Pointer to ChannelData
Return Value	Total No of packet count if successes
Description	Command for free buffer

logCommStatEvt

Prototype	int logCommStatEvt(ui8_t serNum, ui32_t cnt, ui8_t *pkdtm, ui8_t year)
Parameter	integer serial no,count, integer pointer ,and year
Return Value	0 if successes
Description	log communication state event

dispRxQ

Prototype	int dispRxQ(unsigned int portno)
Parameter	integer port no
Return Value	0 if successes
Description	Display Receive queue

dispRecQ()

Prototype	int dispRecQ()
Parameter	None
Return Value	0 if successes
Description	Display Record queue

dispChanStat

Prototype	int dispChanStat(unsigned int portno)
Parameter	None
Return Value	0 if successes
Description	Display channel statistic

4. Display User Interface module

This module is mainly responsible for communication with display & keypad unit which is connected with the serial port RS232. This is mainly responsible for communicating with different modules and to display requested data.

Files : kpdrv.c,lcdDrv,displlay.c

// **kpdrv.c**//

Data Types

```
struct KpDrvData
{
  uint16_t curKpState;
  struct KpEvt kpEvtQ[MAX_KEY_Q];
  int8_t wix;
  int8_t rix;
  int8_t cnt;
  struct timeval baseTime;
};
```

Data Definitions

```
struct KpDrvData kpData;
char keypadBuf[]="*0#D789R456L123U";
```

Global Functions:

KeypadGpioinit

Prototype	void KeypadGpioinit(void)
Parameter	None
Return Value	None
Description	initialization of GPIO pins for keypad

keypad4x4_ReadChar

Prototype	static int keypad4x4_ReadChar(int col)
Parameter	Column No
Return Value	return row value
Description	reading the character pressed

writeKpEvt

Prototype	static int writeKpEvt(struct KpEvt *ke)
Parameter	pointer to structure of KpEvt
Return Value	return 0 if success
Description	writing event

kpDrvReadEvt

Prototype	int kpDrvReadEvt(struct KpEvt *ke)
Parameter	pointer to structure of KpEvt
Return Value	return 0 if success
Description	Reading event

kpDrvscan

Prototype	int kpDrvReadEvt(struct KpEvt *ke) void kpDrvscan(void)
Parameter	None
Return Value	None
Description	scanning the key pressed

kpDrvInit

Prototype	void kpDrvInit(void)
Parameter	None
Return Value	None
Description	scanning the key pressed

// **lcdDrv**//

Data Types

```
struct LcdDrvData
{
  struct LcdEntry lcdQue[LCD_Q_SZ];
  uint32_t rix;
  uint32_t wix;
  uint32_t cnt;
  uint32_t busy;
  uint32_t delay;
  struct timeval wrtTime;
};
```

Data Definitions

struct LcdDrvData lcdData;

lcdGpioInit

Prototype	void lcdGpioInit(void)
Parameter	None
Return Value	None
Description	Initializing GPIO Pin for lcd

lcdBkltOn

Prototype	void lcdBkltOn(void)
Parameter	None
Return Value	None
Description	Back light on , display on

lcdBkltOff

Prototype	void lcdBkltOff(void)
Parameter	None
Return Value	None
Description	Back light off but display on

lcdCurOff

Prototype	void lcdCurOff(void)
Parameter	None
Return Value	None
Description	curser blinking off

lcdCurOn

Prototype	void lcdCurOn(void)
Parameter	None
Return Value	None
Description	curser blinking on

lcdInit

Prototype	void lcdInit(void)
Parameter	None
Return Value	None
Description	Initialization of lcd

lcdClear

Prototype	void lcdClear(void)
Parameter	None
Return Value	None
Description	clears the lcd

lcdWriteStr

Prototype	int lcdWriteStr(char *str,int len)
Parameter	Data pointer, total length
Return Value	0 if success
Description	writes the data on LCD

lcdGoToAddr

Prototype	static void lcdGoToAddr(char addr)
Parameter	Address on which to display
Return Value	None
Description	writes the data on LCD on particular address

lcdGoToLC

Prototype	void lcdGoToLC(char line, char col)
Parameter	line and coloumn where have to display
Return Value	None
Description	writes the data on LCD on particular line and coloumn

lcdWriteDirect

Prototype	void lcdWriteDirect(uint8_t val,uint8_t type)
Parameter	value and type means command or data
Return Value	None
Description	writes the data on LCD

lcdProcessQue

Prototype	void lcdProcessQue(void)
Parameter	None
Return Value	None
Description	Displaying data on after some delay

lcdWriteQ

Prototype	static int lcdWriteQ(uint8_t data, uint8_t type, uint32_t delay
Parameter	type and delay
Return Value	SUCCESS
Description	writing data on queue

lcdReadQ

Prototype	static int lcdReadQ(struct LcdEntry *le)
Parameter	pointer to structure LcdEntry
Return Value	SUCCESS
Description	Reading data from queue

// displlay.c//

Data Variable

unsigned char pswd[5]="1234";
unsigned char str[5];
unsigned char tmr[13];
int8_t pswdix;
int8_t timix;
int8_t curix;
int8_t pswdflag;
int8_t startTime=0;
int8_t result=0;

Data Definitions

struct modSel mods;
extern struct LcdDrvData lcdData;

angVolValOnLcd

Prototype	void angVolValOnLcd(uint16_t value,int row,int col)
Parameter	value ,row and column
Return Value	None
Description	Displaying Analog voltage on LCD

angCurrentValOnLcd

Prototype	void angCurrentValOnLcd(uint16_t value,int row,int col)
Parameter	value ,row and column
Return Value	None
Description	Displaying Analog current on LCD

homeScreen

Prototype	void homeScreen(void)
Parameter	None
Return Value	None
Description	Displaying home screen on LCD

DeFaultScn

Prototype	void DeFaultScn(void)
Parameter	None
Return Value	None
Description	Displaying the Default Message

menuSel

Prototype	void menuSel(struct modSel *pmod)
Parameter	pointer to structure of modSel
Return Value	None
Description	Displaying menu selection

procSubMenu

Prototype	void procSubMenu(struct modSel *pmod)
Parameter	pointer to structure of modSel
Return Value	None
Description	Displaying sub menu selection

angMenu

Prototype	void angMenu(struct modSel *pmod)
Parameter	pointer to structure of modSel
Return Value	None
Description	Displaying Analog menu

upDateScn

Prototype	void upDateScn(struct modSel *pmod)
Parameter	pointer to structure of modSel
Return Value	None
Description	Displaying current changed value

dispDigMsg1

Prototype	void dispDigMsg1(void)
Parameter	None
Return Value	None
Description	Displaying channels 1 t0 8

dispDigMsg2

Prototype	void dispDigMsg1(void)
Parameter	None
Return Value	None
Description	Displaying channels 9 to 15

dispDigMsg3

Prototype	void dispDigMsg1(void)
Parameter	None
Return Value	None
Description	Displaying channels 16 t0 24

dispDigMsg4

Prototype	void dispDigMsg1(void)
Parameter	None
Return Value	None
Description	Displaying channels 25 to 32

cursormov

Prototype	void cursormov(struct modSel *pmod)
Parameter	pointer to structure of modSel
Return Value	None
Description	used for cursor to move forward /backward

cursorIndex

Prototype	void cursorIndex(int8_t curix)
Parameter	cursor index
Return Value	None
Description	selects the channel to display

pswdChange

Prototype	uint32_t pswdChange(struct modSel *pmod)
Parameter	pointer to structure of modSel
Return Value	0 if success
Description	password change if required

pswdInit

Prototype	void dispDigMsg1(void)
Parameter	None
Return Value	None
Description	Read the Password from eeprom or writes

pswdUpdate

Prototype	uint32_t pswdUpdate(struct modSel *pmod)
Parameter	pointer to structure of modSel
Return Value	0 if success
Description	used to change or update the password

newPswd

Prototype	uint32_t newPswd(struct modSel *pmod
Parameter	pointer to structure of modSel
Return Value	0 if success
Description	used to change the password

TimeOnLcd

Prototype	uint32_t newPswd(struct modSel *pmod
Parameter	pointer to structure of modSel
Return Value	0 if success
Description	used to display time on lcd

setTimeOnLcd

Prototype	uint32_t newPswd(struct modSel *pmod
Parameter	pointer to structure of modSel
Return Value	None
Description	used to set time on lcd

printTimeOnLcd

Prototype	void printTimeOnLcd(void)
Parameter	None
Return Value	None
Description	print current time on lcd

CursorLeft

Prototype	void CursorLeft(struct modSel *pmod)
Parameter	pointer to structure of modSel
Return Value	None
Description	Used to move cursor left

5. Command Line Interface Module

This module is mainly responsible for giving the user interface with application in terminal by giving some commands we can get some statistics or read some important parameters and also user can modify those parameters.

This module is implemented in the file cli.c

Data Types

```
struct cliData
{
  int serHndl;          //this maintains the cli serial port fd
  char state;           //this maintains the cli state
  char cmdbuf[80]; //this maintains the user input
  char cix;             //this maintains the index of the input message
};
```

```
typedef struct cmdFun
 {
    char *cmd;                      //this maintains the command string
    int (*fun) (int, char **); //this maintains the corresponding function pointer
    char *helpstr;                 //this is an help string
 } cmdFun_t;
```

Data Definitions

```
static struct cliData cliObj;

static cmdFun_t cmdTab[MAX_CLI_CMDS] =
{
   {"h", dispCmds, "Print available commands"},
 };
```

Messages Format:

```
<Command string > <arg1> <arg2>.....<\n>
```

Global Functions:

cliInit

Prototype	int32_t cliInit (void)
Parameters	Void.
Return value	SUCCESS if initialization success and ERROR in initialization fail condition (integer type).
Description	In this function we are initializing the CLI serial port and update fd into global variable ‘cliObj serHndl’, here we use usartDrv functions.

cliProcChar

Prototype	int32_t cliProcChar(char ch)
Parameters	Character which is read from CLI serial port (character type).
Return value	SUCCESS if total message received and ERROR in middle of message receiving process.
Description	Whenever one character is received from the cli serial port we are appending that character to the command string ‘cliObj cmdbuf’ of global variable, and after receive of full message we are giving that total string to cliProcCmd function.

Local Functions:

cliProcCmd

Prototype	int32_t cliProcCmd(void)
Parameters	Void.
Return value	SUCCESS if command string matches in the table and ERROR if not matches.
Description	In this function we are breaking the command string into tokens and checking for the matching command string in the global variable ‘cmdTab’ and calling the corresponding function pointer for the process of remaining tokens.

6. Command Process Module

This module is mainly responsible for processes the command and respond with ACK and executes command and write to fd.

File : cmdproc.c

Global variable:

```
struct shared_struct_var *shared_var;
char                global_rly8_status;
char                global_rly16_status;
char                telectrl8_data;
char                telectrl16_data;
```

Global Functions:

processCmdPkt

Prototype	i32_t processCmdPkt(struct channelData *pcd)
Parameter	Pointer to ChannelData
Return Value	SUCCESS if successes
Description	processes the command and responds with ACK.

addHdrCksumAndSend

Prototype	static void addHdrCksumAndSend(uint32_t fd, uint8_t *ackmsg,uint16_t Len, uint8_t *rxmsg)
Parameter	channel fd,integer pointer,len and receive message pointer
Return Value	SUCCESS if successes
Description	create packet to send to respond to command packet

executeCmdPkt

Prototype	i32_t executeCmdPkt(ui32_t fd, ui8_t* cmdpkt)
Parameter	channel fd ,integer pointer
Return Value	0 if successes
Description	executes command and writes to fd.

logTimeChangeEvent

Prototype	int logTimeChangeEvent(ui8_t *pkdtm,ui8_t year)
Parameter	integer pointer ,integer year

Return Value SUCCESS if successed
Description create packet when time change event.

7. Local CMU module

This module is responsible for mainly communication between Event logger and PHMU.It receive data request get the data from Sd card and send back to event logger. In this process so many function is implemented.

Data Types:

```
union
{
  uint32_t u32val;
  uint8_t  u8val[4];
}pkdTmUn;
```

Data Definitions

```
struct lcmuData lcmu;
struct chnlStatCnts lcmuStat;
```

Global Functions:

calChksum16

Prototype uint16_t calChksum16(uint8_t *msg, int32_t len)
Parameter integer pointer ,integer length
Return Value Cheksum byte if successed
Description use to calculate cheksum

checksum

Prototype unsigned char checksum(unsigned char *buff,int len)
Parameter integer pointer ,integer length
Return Value Cheksum byte if successed
Description use to calculate cheksum

xbeeUartWrite

Prototype	void xbeeUartWrite(uint8_t data[],int len)
Parameter	integer data ,integer length
Return Value	None
Description	Make frame in API mode

addHdrCksumAndSend

Prototype	void addHdrCksumAndSend(uint8_t *msg, uint16_t len, uint8_t *rxmsg)
Parameter	integer nessage pointer ,integer length,integer receive pointer
Return Value	None
Description	Add DLid,seqNo,Cheksum in frame

extractDigCfgRecord

Prototype	void extractDigCfgRecord(int32_t dChnIx, uint8_t *msg)
Parameter	integer channel index ,integer length,integer message pointer
Return Value	None
Description	Extract Digital Configuration Records

extractAngCfgRecord

Prototype	void extractAngCfgRecord(int32_t aChnIx, uint8_t *msg)
Parameter	integer channel index ,integer length,integer message pointer
Return Value	None
Description	Extract Analog Configuration Records

lcmuProcessMsg()

Prototype	int lcmuProcessMsg()
Parameter	None
Return Value	None
Description	According to Data packet bit it works on if it is data request bit then it send Data from SD card otherwise it send command request

printbuf

Prototype	int32_t printbuf(struct lcmuData *pcd,int len)
Parameter	pointer to lcmData ,total length
Return Value	0 if success
Description	print the data

lcmuRxAndProc()

Prototype	void lcmuRxAndProc()
Parameter	None
Return Value	None
Description	Check data buffer if start delimeter is 7E or AA according to this it calls Function

rxSerialXbee()

Prototype	void rxSerialXbee()
Parameter	None
Return Value	None
Description	Receive communication protocol that makes sure that frame we get is valid And as well as Length too.

rxSerialProcess

Prototype	void rxSerialProcess()
Parameter	None
Return Value	None
Description	Receive communication protocol that makes sure that frame we get is valid And as well as Length too.

localPortStat

Prototype	void localPortStat(void)
Parameter	None
Return Value	None
Description	Print local port statistic

lcmuInit

Prototype	int32_t lcmuInit()
Parameter	None
Return Value	Port Fd
Description	Initialize UART4 with 115200 Baud Rate for local port

8. Main module

This module is mainly responsible for integrating all the modules into a single application.

main.c

This file is used to integrate all the other modules into a single module.

Global Variables

```
FATFS fatfs;
FILINFO *fileinfo;
DIR sdDir;
UINT BytesRead=0;
uint8_t events_Sign[8]={0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
uint8_t events_Signature[10];
uint8_t flag=0;

uint8_t sys_cfg_debug[15] =
{
  0x14,                                    //Data logger ID                (1 byte)
  0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,//Signature        (8 bytes)
  0x00,0x10,                          //No of digital inputs          (2 bytes)
  0x11,                                    //No of Analog inputs          (1 byte)
  0x00,0x00,                          //No of Alarms defined         (2 bytes)
  0x00                                  //Is RTU exist?                   (1 byte)
};

uint16_t i=0;
uint8_t res=0;

uint32_t
TIMFreq[8]={0},Capture[8]={0},IC3ReadValue[16]={0},CaptureNumber[8]={0},flgGetVal[8]
={0};
uint8_t ai_Buff[23];
uint8_t w_Buff[50]={0},r_Buff[50]={0};
int sysTickFlag,milsysTickFlag;
uint8_t Response_Ok_Flag=0,flagXbeeDataReady=0,flagTimerStart=0,flagTimerReady=0;
//,flagTimerStart;
uint16_t timerCount1=0;
bobDataType    bobData[MAX_DI_CHS];
chatDataType   chatData[MAX_DI_CHS];
struct channelData chnAdata;
struct channelData chnBdata;
struct dlsSysData  sysData;
struct DLConfigration DLsysConfig;
```

```
int32_t uart2HDL;;
struct cliData cliObj;
int allowReset,dbgcnt;
```

1. main

Prototype : int32_t main(void)
Parameters : None
Return Type : Returns 0 on Success.
Description : This function includes all initializations, an infinite loop in which all the modules calls.

Pseudo code:

```
/********************** Initializations *************************************/

CALL  delay to provide/give time delay
CALL  gpioInit to initializations of GPIO pins A,B,C,D,E,F,G,H,I
CALL  cliInit to initialize cli serial port returns address of  cli serial port control block as integer
CALL  SysTick_Config to genetates interrupt for every 1 misec
CALL  lcmuInit to initialize UART4 with baud rate 115200
CALL  rtcInit to initialize Real time clock
CALL  ADCInit to initialize Internal ADCs;
CALL  digital_Init to initialize some GPIO pin as Digital Input;
CALL  EXTILine15_10_Config to initialize some GPIO as External Interrupt Line ;
CALL  EXTILine9_5_Config to initialize some GPIO as External Interrupt Line ;
CALL  timer2_Init to initialize Timer2 to provide independent time
CALL  timer5Init to initialize Timer5 to provide independent time
CALL  initSysSeconds
CALL  initIoMod
CALL  watchdogInit to initialize watch dog timer
CALL  bbRamInit to initialize battery backup SRAM

IF check file system initialize or not

ZIGBEE PORT Initializatons
PORT A Initializatons using UART2
PORT B Initializatons using UART1

CALL  initIoConfig to initialize and check the Input/ouput configuration successfully from SD
       card
CALL  sessionRestore();
CALL  startTimer(&timers[HEALTH_TMR],HEALTH_TIMER_VAL)to initialize Health Time
CALL  copyCurTicks(&pkdTicks,&year) to initialize Gives the timestamp in timeticks format
(railway format) from system time;
```

```
CALL  getSysSeconds to initialize Returns time stamp of seconds
CALL  getSys10mSec to initialize Returns time stamp of milliseconds;
CALL  startTimer(&timers[DIG_TMR],DIG_TIMER_VAL) to initialize Digital Timer;
CALL  startTimer(&timers[ANG_TMR],ANG_TIMER_VAL) to initialize Analog Timer;
CALL  createAndLogBootEvt to initialize;
CALL  printLogRecQueStat to initialize print log records statistic;
CALL  printbksrval to initialize print backup SRAM ;
CALL  watchdogTrigger to initialize external watchdog timer
CALL  vtofscan to initialize voltage to frequency
/*********************************************************/
 WHILE 1

   WHILE sysTickFlag
     CALL tmrProcess to update timers in 100 milliseconds
     DECREMENT sysTickFlag
   ENDWHILE

   WHILE sysTickFlag
     CALL tmrProcess to update timers in 1 milliseconds
     DECREMENT sysTickFlag
   ENDWHILE

   CALL copyCurTicks to Gives the timestamp in timeticks format (railway format) from system
           time

   IF one character is read from cli port THEN
     CALL cliProcChar with read char for processing
   ENDIF
   /************************************/
         /*newly added kpdrv and lcddrv and display function          */
  CALL lcdProcessQue to lcd process
  CALL kpDrvscan()     to scan the pressed No

  IF Keypad Read Event is Equal to 0
    IF Keypad State is SET THEN
      Assign KeyNO to Key Val
      CALL    menuSel to select the menu
    ENDIF
   ENDIF
  CALL upDateScn to update Scanning

                    /**********************************/
 CALL lcmuRxAndProc to recieve request frame and process it

   IF Diag bit SET            //if 1 system in diagnosis mode
    CALL watchdogTrigger to intialize external watchdog
    continue
   ENDIF

   IF Check configuration flag
    CALL initIoConfig
   IF check configuration status bit SET
      PRINT SUCCESS
   CALL watchdogTrigger
     continue
   ENDIF

 //The following code is executed only on valid configuration
   IF check the pvtcomAcq bit
      IF Digital timer expired
      CALL setbksrval
      CALL readandprocDIdata
      CALL startTimer to start Digital timer again
      CALL setbksrval
      ENDIF
    IF Analog timer expired
    CALL setbksrval
    CALL setbksrval
    CALL readandprocAIdata
    CALL setbksrval;

    IF check vtofDataReady bit
      CALL readVTOFdata
      CALL setbksrval
      CALL copyCurTicks to Gives the timestamp in timeticks format (railway format)
      CALL processAIdata
      CALL detectAngAlarms
      CALL setbksrval
      assign 0 to flagRead
      assign 0 t0 vtofDataReady
    ENDIF
  CALL startTimer to start Analog timer again
 ENDIF
ENDIF

IF uart available at uart for channel A THEN
  CALL readSerialData to read data
IF uart available at uart for channel B THEN
```

```
  CALL readSerialData to read data
CALL rxProcessChannel
CALL rxProcessChannel
CALL setbksrval
CALL txProcessChannel with chnAdata
CALL setbksrval
CALL setbksrval
CALL txProcessChannel with chnBdata
CALL setbksrval

IF Channel A AckTimer expires THEN
  CALL processPendAckTmout with chnAdata
  CALL stopTimer with chnAdata.ackTmr
ENDIF
IF Channel A AckTimer expires THEN
  CALL processPendAckTmout with chnBdata
  CALL stopTimer with chnBdata.ackTmr
ENDIF

IF Health timer expired
  CALL creat_pack with (0x05 sysData.healthStat (ui8_t *)&pkdTicks,year) to Health record
  Generation
  CALL startTimer to start health timer
ENDIF

//////////////////Periodic Processings STARTS here /////////////////
IF seconds increase THEN
  CALL  watchdogTrigger
  CALL  getSysSeconds and store return value in cuSec
  Increase chnAdata.timeSinceLastRxPk
  Increase chnBdata.timeSinceLastRxPkt
IF After every 30 min THEN
  CALL logCommStatEvts          //Generate communication events at half-an-hour in wall time
ENDIF
Increase Local Second count     //localSec++
CALL  getSysSeconds and store return value in cuSec
IF After every 5 Seconds localy
  CALL resetChattering
  CALL sessionSave
ENDIF

IF after every 60s THEN
  assign 0 to localSec variable   //localSec=0
  Increse Local Minute variable   //localMin++;
IF after every 34 Minute THEN
  CALL watchdogTrigger
  CALL logAllDiStatEvts to Generate All Digital Status events
  CALL copyCurTicks to Gives the timestamp in timeticks format (railway format) from system
  CALL record_type to  generate system configuration event
ENDIF
ENDIF
ENDIF
////////////////////Periodic Processings ENDS here////////////////////

//Feed the watch dog, too keep it quite
IF allowReset bit Not SET THEN
  CALL watchdogTrigger
ENDWHILE
ENDMAIN
```