2016

Event Logger Design Document

Sirveen Control System

24-Feb-16

Revision History

Revision	Date	Author	Review	Review description
1.0	24-Feb-16	Md Shahid		Initial version

Definitions

Term	Definition
PHMU	Point Health Monitoring Unit

References

Following are the reference document used in preparing this document.

Ser No	Documents
1	Event Logger System Requirement
2	Event Logger Hardware Requirement
3	Driver Frame Work for STM32 based boards
4	Software Engineering A Practitioner's Approach by Roger S Pressman

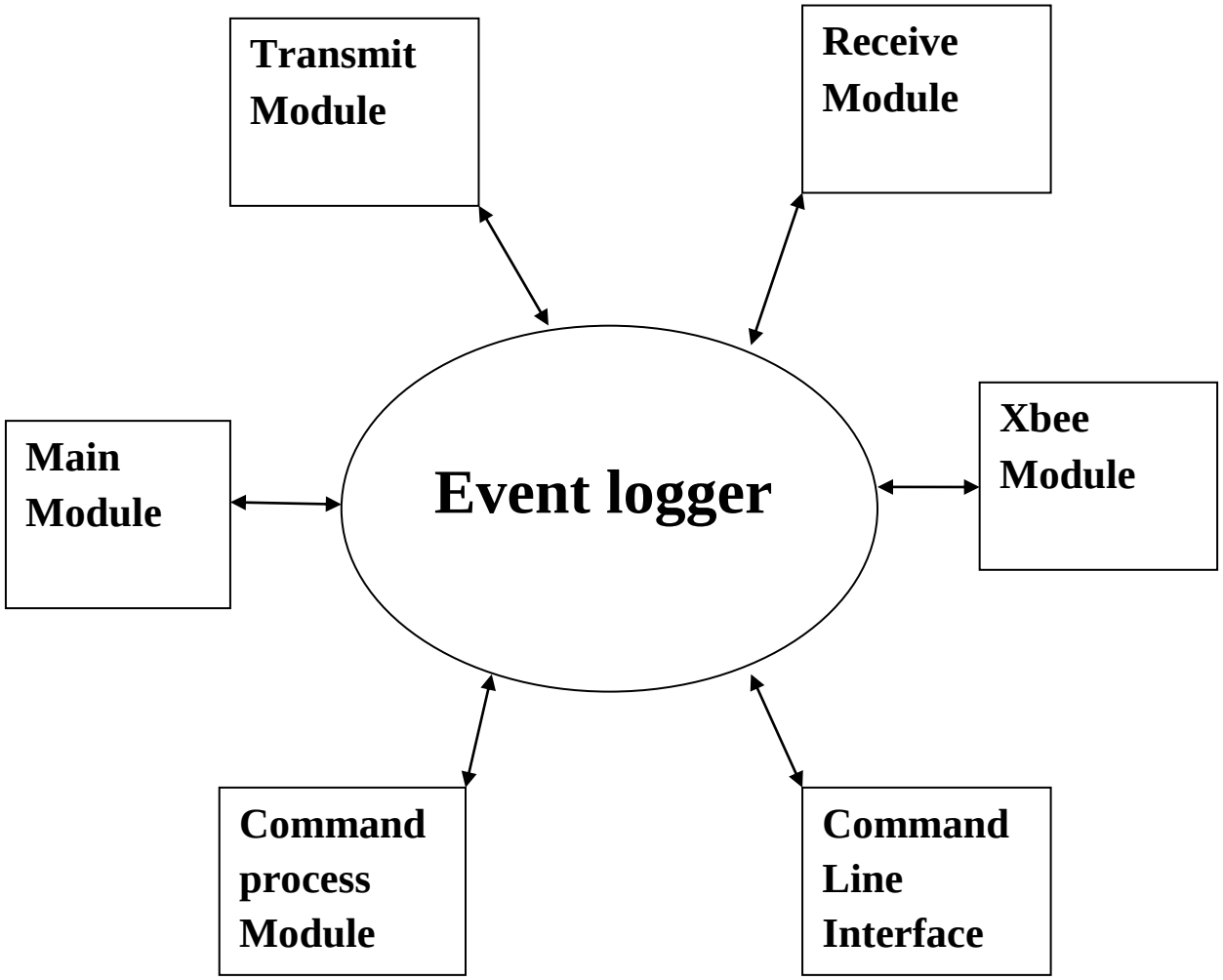
Introduction

This document describes the software design for Event Logger Unit.

Data Flow Diagram

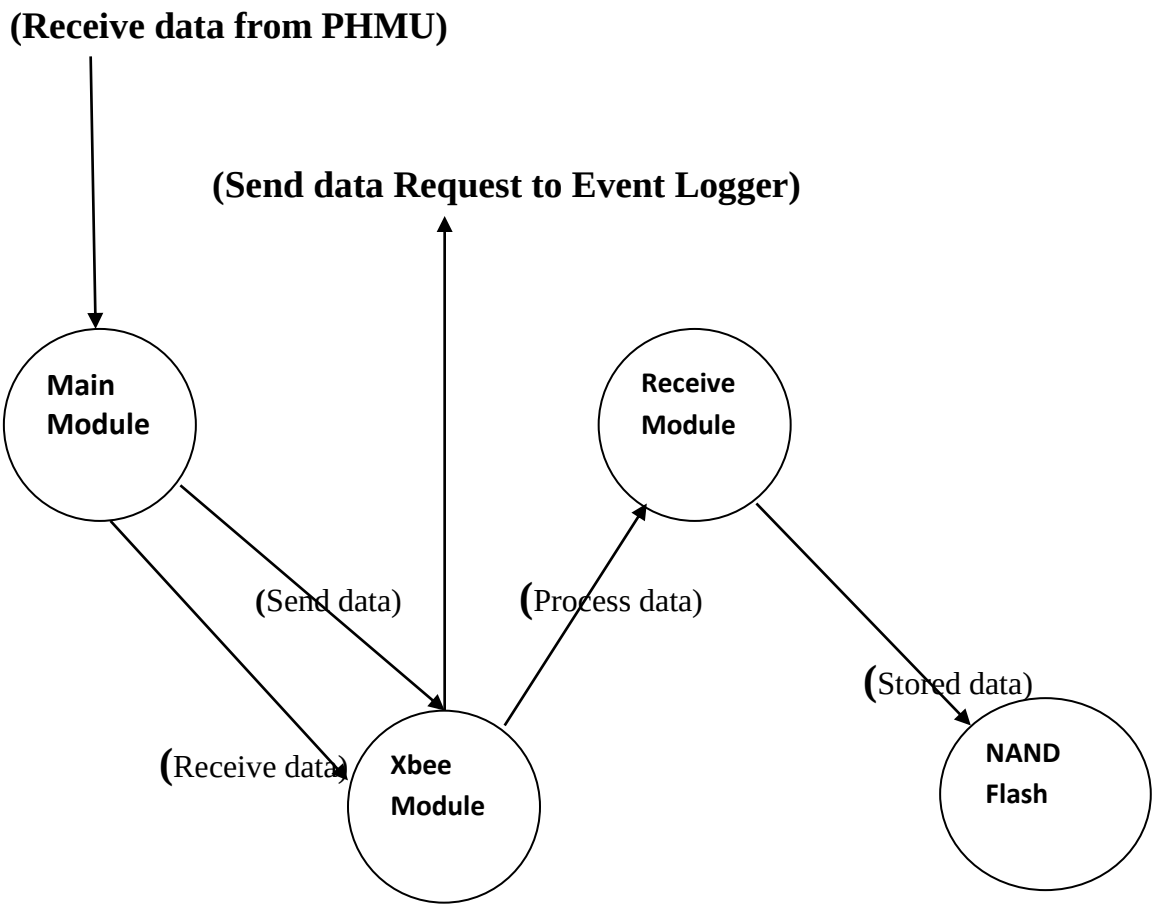
Level 0 data flow diagram (DFD)

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



Level 1 data flow diagram (DFD)

In the level 0, complete Event Logger 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 Event Logger Software

1. Transmit module

2. Receive module

3. Xbee module

4. Command Line interface module

5. Command process module

6. Main module

1. 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.

2. Receive module

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

3. Xbee module

This module is responsible for making xbee API frame to send request to PHMU as well as it receive the data from PHMU.

4. 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 Event Logger performance statistics, to log the event Logger data periodically for later analysis.

5. Command Process Module

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

6. Main module

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

1. 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

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

2. 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

processPhmuData(struct channelData *pcd)

Prototype	i32_t processPhmuData(struct channelData *pcd)
Parameter	pointer to struct channelData
Return Value	return Integer if successed
Description	Stored the Data into Flash of Event Logger

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

validateAndProcess

Prototype	int validateAndProcess(struct channelData *pcd)
Parameter	Pointer to ChannelData
Return Value	Total No of packet count if successes
Description	validates and processes the Data

rxProcessChannel

Prototype	int rxProcessChannel(struct channelData *pcd)
Parameter	Pointer to ChannelData
Return Value	Total No of packet count if successes
Description	validate and process the Data

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

3. Xbee module

This module is responsible for making xbee API frame to send request to PHMU as well as it receive the data from PHMU.

File xbee.c

Global Functions:

xbeeInit

Prototype	`int xbeeInit()`
Parameter	None
Return Value	0 if successes
Description	For Xbee initialization

uartReconfigure

Prototype	int uartReconfigure()
Parameter	None
Return Value	-1 if fail
Description	This function will reconfigure the uart for xbee

sendReqToPhmu

Prototype	i32_t sendReqToPhmu()
Parameter	None
Return Value	SUCCESS if success
Description	Requesting to PHMU

checksum

Prototype	unsigned char checksum(unsigned char *buff,int len)
Parameter	frame and legth of frame
Return Value	Result if success
Description	calculating the checksum

xbeeUartWrite

Prototype	int xbeeUartWrite(unsigned char *data,unsigned char *address
Parameter	frame and legth of frame
Return Value	Result if success
Description	Making API Frame and writing to Uart

readXbeeSerialData

Prototype	int32_t readXbeeSerialData(struct ComProto *pcd)
Parameter	pointer to structure of ComProto
Return Value	Result if success
Description	Reading the PHMU data

4. 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.

5. 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.

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

```
uint32_t getSysSeconds();
int32_t  serInit(char *name);
uint32_t SystemCoreClock = 120000000;  //crystal feequency

int allowReset,dbgcnt;

struct channelData chnDdata;
struct channelData chnAdata;
struct channelData chnBdata;
struct channelData chnCdata;
struct channelData cQueue;
struct ComProto xbeePort;
struct dlsSysData  sysData;

int sysTickFlag;
struct DLConfigration DLsysConfig;
void iwdg_rld();
int initwdg();
```

Pseudo code:

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.

Local variables:

```
int32_t n,stat;
char    ch;
int32_t cliHndl;
uint32_t pkdTicks,cuSecs,localSec=0,localMin=0;
uint8_t year;


/****** **************Initializatons********************/
CALL  stackTestFill()
CALL  bbRamInit to initialize battery backup SRAM
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  tmrInit() to initialize timer
CALL  rtcInit to initialize Real time clock
CALL  initSysSeconds();

IF Total phmuId are loaded THEN)
   print CRS
 ELSE
   print CRF

Initializatons of UART1 for PORT A
Initializatons of UART6 for PORT B
Initializatons of UART5 for PORT C
Initializatons of UART4 for PORT D as well as Zigbee

CALL initChannel with chnAdata to initialize channels
CALL initChannel with chnBdata to initialize channels
CALL initChannel with chnCdata to initialize channels
CALL initChannel with chnDdata to initialize channels

CALL initLogRecQueue();
CALL initwdg();
CALL cuSecs     = getSysSeconds();
CALL createAndLogBootEvt();
CALL uartReconfigure();
assign xbeePort.flag with 1          //To initialize the send and receive command
```

```
/*************************************************************/
 WHILE 1

   WHILE sysTickFlag
     CALL tmrProcess to update timers
     DECREMENT sysTickFlag
   ENDWHILE
   CALL copyCurTicks to Gives the timestamp in timeticks format

   IF one character is read from cli port THEN
     CALL cliProcChar with read char for processing
  ENDIF

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

  IF xbeePort flag bit is SET

    CALL sendReqToPhmu();
    CALL startTimer(&timers[PVTCOM_TMR],20)
    assign xbeePort.flag to 0
    Increase sysData.phmuVal          //sysData.phmuVal++
  ENDIF
  IF readXbeeSerialData(&xbeePort) equal to SUCCESS)

    SET xbeePort.flag to 1
    CALL startTimer(&timers[PVTCOM_TMR],20)
    IF sysData.phmuVal equal sysData.phmuCnt
      SET sysData.phmuVal to 0
      Increase sysData.phmu[sysData.phmuIx].rcvCnt
      Assign sysData.phmuIx with current Phmu Index
  ENDIF

  IF timer Expired
    SET xbeePort.flag to 1
    CALL stopTimer(&timers[PVTCOM_TMR]);
    Increase Timeout index
```

```
  Increase PHMU index
  IF phmuVal is Equal sysData.phmuCnt)
    assign sysData.phmuVal to 0
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

IF uart available at uart for channel c THEN
  CALL readSerialData to read data


CALL rxProcessChannel with chnAdata to process the receive data
CALL rxProcessChannel with chnBdata to process the receive data
CALL rxProcessChannel with chnCdata to process the receive data
CALL txProcessChannel with chnAdata to transmit and process
CALL txProcessChannel with chnBdata to transmit and process
CALL txProcessChannel with chnCdata to transmit and process

IF chnAdata ackTmr Expired THEN
  CALL processPendAckTmout with chnAdata
  CALL stopTimer with chnAdata.ackTmr
ENDIF

IF chnBdata ackTmr Expired THEN
  CALL processPendAckTmout with chnBdata
  CALL stopTimer with chnBdata.ackTmr
ENDIF


IF chnCdata ackTmr Expired THEN
  CALL processPendAckTmout with chnCdata
  CALL stopTimer with chnCdata.ackTmr
ENDIF

IF cuSecs Not equal getSysSeconds() THEN

  CALL  getSysSeconds and store return value in cuSec
```

```
    Increase chnAdata.timeSinceLastRxPk    //chnAdata.timeSinceLastRxPkt++;
    Increase chnBdata.timeSinceLastRxPk    //chnBdata.timeSinceLastRxPkt++;
    Increase chnCdata.timeSinceLastRxPk    //chnCdata.timeSinceLastRxPkt++;
    Increase localSec                      //localSec++
  ENDIF
  ////////////////////Periodic Processings ENDS here//////////////////////////


 //Feed the watch dog, too keep it quite
  IF allowReset bit Not SET THEN
    assign KR register with 0xaaaa //IWDG->KR = 0xaaaa;

 ENDWHILE
ENDMAIN

int initwdg
 assign KR register with 0x5555      //IWDG->KR  = 0x5555;
 assign PR register with 0x0006      //IWDG->PR  = 0x0006;
 assign KR register with 0xcccc      //IWDG->KR  = 0xcccc;
ENDinitwdg

void iwdg_rld
 assign KR register with 0xaaaa     //IWDG->KR = 0xaaaa;
ENDiwdg_rld
```