İçindekiler

ADC CONVERTER 3

1. int ADC_Init(void) 3
2. int ADC_Read_Pin(unsigned int pinNumber, unsigned int *readValue) 3

Extra Definitions 3

Its Usage 4

AES ENCRYPTION 4

1. void Aes_Init (AesType type) 4
2. void EncryptBlock (unsigned char data[16]) 4
3. void DecryptBlock (unsigned char data[16]) 4

Extra Definitions 4

Its Usage 5

FILE SYSTEM 5

1. FILE* FileOpen(const char *fileName, FILE_FLAGS flags) 5
2. int FileRead(FILE *hdl, char *buf, int len) 5
3. int FileWrite(FILE *hdl, const char *buf, int len) 6
4. int FileSeek(FILE *hdl, long off, int orig) 6
5. void FileClose(FILE *hdl) 6
6. int FileSize(const char *fileName) 6
7. int FileRemove(const char *fileName) 6
8. int FileExist(const char *file) 7
9. int FileRename(const char *oldname, const char *newname) 7
10. int FileCopy (const char *source, const char *target) 7

Extra Definitions 8

Its Usage 8

GPIO 9

1. int GPIO_Init(unsigned int pinNumber, GPIO_DIR direction) 9
2. GPIO_TYPE GPIO_Read_Pin(unsigned int pinNumber) 10
3. int GPIO_Write_Pin(unsigned int pinNumber, GPIO_TYPE value) 10

Extra Definitions 11

Its Usage 11

SERIAL PORT 11

1. int Uart_Init(const int portIdx, unsigned int baud) 11
2. void Uart_Close(int fd) 12
3. void Uart_Flush(int fd) 13
4. int Uart_Send(int fd, unsigned char *buf, unsigned int buf_size) 13
5. int Uart_Receive(int fd, unsigned char *buf, unsigned int buf_size, unsigned int vaitINms) 13
6. int CheckDataAvailability(int fileIndx, unsigned char blocking, long int time_sec, long int time_usec) 13
Extra Definitions 14
Its Usage 15
TCP/IP CLIENT CONNECTION 15
1. int TCP_Client_Connect(char *ip, int port) 15
2. int TCP_Client_Read(unsigned char *buffer, int len) 16
3. int TCP_Client_Write(unsigned char *buffer, int len) 16
4. void TCP_Client_Close(int socket) 16
Extra Definitions 16
Its Usage 17
USER LEDS 17
1. void LedON(unsigned char ledNumber) 17
2. void LedOFF(unsigned char ledNumber) 18
3. int UserLedsInit(void) 18
4. void UserLedsClose(void) 19
Extra Definitions 19
Its Usage 19
PIN CONFIGURATION 20

ADC CONVERTER

1. int ADC_Init(void)

//it is used for using ADC. It should be called before reading a pin's value. It should be called just one time (before while 1 for example)

```
{
        FILE *ain;
        ain = fopen("/sys/devices/bone_capemgr.8/slots", "w");
        if(ain == NULL)
                return RET_NOK;
        fseek(ain,0,SEEK_SET);
        fprintf(ain,"cape-bone-iio");
        fflush(ain);
        fclose(ain);
        return RET_OK;
}
```

2. int ADC_Read_Pin(unsigned int pinNumber, unsigned int *readValue)

//Its intensity is 12-bit up to 1800mV.

```
{
        FILE *aval;
        int value;
        char buf[50] = {0};
        sprintf(buf, "/sys/devices/ocp.2/helper.14/AIN%d", pinNumber);
        aval = fopen(buf, "r");
        if(aval == NULL)
                return RET_NOK;
        fseek(aval,0, SEEK_SET);
        fscanf(aval,"%d",&value);
        fflush(aval);
        fclose(aval);
        *readValue = value;
        return RET_OK;
}
```

Extra Definitions

```
#include <stdio.h>

#define MGRNUM          8
#define RET_OK  0
#define RET_NOK -1
```

Its Usage

- GND_ADC should be connected to mutual ground
- AIN0-6 can be used to ADC Input
- 1.8 volt is max connected voltage
- Code Example;

```
int main(void)
{
	ADC_Init();
	unsigned int value = 0;

	while(1)
	{
		ADC_Read_Pin(0, &value);
		printf("Value: %d", (value & 0x0FFF);
	}
}
```

AES ENCRYPTION

1. void Aes_Init (AesType type)

//Before its usage, this function should be called first for a just one time. It is not necessary to write its code because there are too much function that are not used by developer.

2. void EncryptBlock (unsigned char data[16])

//It is used to Encrypt the buffer. Buffer length must be 16 and its encrypted buffer is returned in the same parameter. No need to write the code.

3. void DecryptBlock (unsigned char data[16])

//It is used to Decrypt the buffer. Buffer length must be 16 and its decrypted buffer is returned in the same parameter. No need to write the code.

Extra Definitions

```
typedef enum
{
	AES_128,	/*!< AES-128 */
	AES_192,	/*!< AES-192 */
	AES_256		/*!< AES-256 */
}AesType;
const unsigned char AES_KEY[16] = { xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx, xx };
```

Its Usage

```
int main()
{
	Aes_Init(AES_128);
	char *buffer = (char *)calloc(16, sizeof(char));
	sprint(buffer, “selam”);
	EncryptBlock(buffer);
	DecryptBlock(buffer);
	printf(“output: %s”, buffer);
}
```

FILE SYSTEM

1. FILE* FileOpen(const char *fileName, FILE_FLAGS flags)

```
//It is used to create or write/write a file. Its return prm is the file index.
{
	char flg[5] = {0};

	if(flags==R_ONLY)
		strcpy(flg, "r");
	else if(flags==RW)
		strcpy(flg, "r+");
	else if(flags==RW_CREATE)
		strcpy(flg, "w+");
	else if(flags==A_ONLY)
		strcpy(flg, "a");
	else if(flags==RA)
		strcpy(flg, "a+");
	else
		return NULL;

	return fopen(fileName, flg);
}
```

2. int FileRead(FILE *hdl, char *buf, int len)

```
//Read a buffer that’s length is decided by developer
{
	int ret = fread(buf, 1, len, hdl);
	if(ret == len)
```

```
                return len;
        return RETURN_NOK;
}
```

3. int FileWrite(FILE *hdl, const char *buf, int len)

```
//Used to write a buffer to a file. Writing process was decided to FileOpen function before.

{
        int ret = fwrite(buf, 1, len, hdl);
        if(ret == len)
                return len;
        return RETURN_NOK;
}
```

4. int FileSeek(FILE *hdl, long off, int orig)

```
//It changes the cursor position in the file. “orig” prm coule be; SEEK_SET(0)/SEEK_CUR(1)/SEEK_END(2)
and “off” is the movement length of the cursor

{
        return fseek(hdl, off, orig);
}
```

5. void FileClose(FILE *hdl)

```
//A file must be closed if it was opened.

{
        fclose(hdl);
}
```

6. int FileSize(const char *fileName)

```
//return the size of the file in byte count.

{
        struct stat st;
        stat(fileName, &st);
        return st.st_size;
}
```

7. int FileRemove(const char *fileName)

```
//remove the file that was highlighted by its name

{
        return unlink(fileName);
}
```

8. int FileExist(const char *file)

```
//It returns a file exists or not

{
        FILE *fd = FileOpen(fileName, R_ONLY);

   if (fd == NULL)
        return RETURN_NOK;

   return RETURN_OK;
}
```

9. int FileRename(const char *oldname, const char *newname)

```
//change a file's name

{
        return rename(oldname, newname);
}
```

10. int FileCopy (const char *source, const char *target)

```
//create a copied-file with given name.

{
        if(FileExist(source) == RETURN_NOK)//there is no source then quit
        {
                return RETURN_NOK;
        }

        FILE  *sor = NULL;
        sor = FileOpen(source, RW);//open source for reading
        if(sor == NULL)
                return RETURN_NOK;

        int size = -100;
        size = FileSize(source);//find size of the source
        if(size<0)
                return RETURN_NOK;

        char *sourceReadBuf = (char*)malloc(size*sizeof(char));//this will be hold the rest of main string
        if(sourceReadBuf==NULL)//if there is not enough memory then quit
                return RETURN_NOK;

        FileRead(sor, sourceReadBuf, size);//read the file in "size" characters

        FileClose(sor);
```

```
	if(FileExist(target)==RETURN_OK)//there is target then remove it
	{
		if(FileRemove(target) == RETURN_NOK)//if cannot remove it, quit
			return RETURN_NOK;
	}

	//create a target
	FILE *tar = NULL;
	tar = FileOpen(target, RW_CREATE);
	if(tar == NULL)//cannot open then quit
		return RETURN_NOK;

	if(FileWrite(tar, sourceReadBuf, size) == RETURN_NOK)//copy data to new one
		return RETURN_NOK;

	FileClose(tar);

	free(sourceReadBuf);

	return RETURN_OK;//all is well
}
```

Extra Definitions

```
#define RETURN_OK   1
#define RETURN_NOK -1


typedef enum
{
	R_ONLY,//r
	RW, //r+
	RW_CREATE, //w+
	A_ONLY, //a
	RA //a+
}FILE_FLAGS;
```

Its Usage

```
if (FileExist((char *)MAIN_C_FILE_NAME) != -1)
	FileRemove((char *)MAIN_C_FILE_NAME);
```

```
FILE *fileEncrypted = FileOpen((char *)MAIN_C_FILE_NAME, RA);
if(fileEncrypted == NULL)
	printf("Cannot Created the new decrypted file\n");
```

GPIO

1. int GPIO_Init(unsigned int pinNumber, GPIO_DIR direction)

```
//It should be called every time for a pin that will be used.

{
	FILE *io,*iodir;
	io = fopen("/sys/class/gpio/export", "w");
	if(io==NULL)
		return RET_NOK;
	fseek(io,0,SEEK_SET);
	fprintf(io,"%d",pinNumber);
	fflush(io);

	char fileNameDir[40] = {0};
	sprintf(fileNameDir, "/sys/class/gpio/gpio%d/direction", pinNumber);

	iodir = fopen(fileNameDir, "w");
	if(iodir==NULL)
		return RET_NOK;
	fseek(iodir,0,SEEK_SET);

	if(direction == INPUT)
		fprintf(iodir,"in");
	else if(direction == OUTPUT)
		fprintf(iodir,"out");
	else
		return RET_NOK;

	fflush(iodir);

	fclose(iodir);
	fclose(io);

	return RET_OK;
}
```

2. GPIO_TYPE GPIO_Read_Pin(unsigned int pinNumber)

```
//Read a pin status (On or Off) if it is configured as GPIO
{
	FILE *inval;
	int value;

	char fileNameVal[40] = {0};
	sprintf(fileNameVal, "/sys/class/gpio/gpio%d/value", pinNumber);

	inval = fopen(fileNameVal, "r");
	if(inval == NULL)
		return RET_NOK;
	fseek(inval,0,SEEK_SET);
	fscanf(inval, "%d", &value);
	fclose(inval);

	if(value)	return HIGH;
	else		return LOW;
}
```

3. int GPIO_Write_Pin(unsigned int pinNumber, GPIO_TYPE value)

```
//Used to write a status over a GPIO pin.
{
	FILE *ioval;
	char fileNameVal[40] = {0};
	sprintf(fileNameVal, "/sys/class/gpio/gpio%d/value", pinNumber);

	ioval = fopen(fileNameVal, "w");
	if(ioval==NULL)
		return RET_NOK;
	fseek(ioval,0,SEEK_SET);

	if(value == HIGH)
		fprintf(ioval,"%d",1);
	else if(value==LOW)
		fprintf(ioval,"%d",0);
	else
		return RET_NOK;

	fflush(ioval);
	fclose(ioval);
	return RET_OK;
}
```

Extra Definitions

```
#define RET_OK  0
#define RET_NOK -1

typedef enum
{
	INPUT,
	OUTPUT
}GPIO_DIR;

typedef enum
{
	LOW,
	HIGH
}GPIO_TYPE;
```

Its Usage

```
GPIO_Init(48, INPUT);
GPIO_TYPE ret = GPIO_Read_Pin(48);
if(ret == HIGH)
	//sth
else
	//sth else
```

SERIAL PORT

1. int Uart_Init(const int portIdx, unsigned int baud)

//portIdx is the number of for example UART4. Port number can be seen in the pin config image. It returns a file index. It will be used in other functions. This should be used just once for a port.

```
{
	if(portIdx < 0)
		return RETURN_NOK;

	char portName[32] = {0};
	unsigned char BAUD = 0;

	if(baud == 4800)
```

```
        BAUD = B4800;
    else if(baud == 9600)
        BAUD = B9600;
    else if(baud == 19200)
        BAUD = B19200;
    else
        return RETURN_NOK;

    int file;

    if(portIdx<0 || portIdx>5)
        return RETURN_NOK;

    sprintf(portName, "/dev/ttyO%d", portIdx);

    if ((file = open(portName, O_RDWR | O_NOCTTY | O_NDELAY))<0){
        return RETURN_NOK;
    }

    struct termios options; // the termios structure is vital
    tcgetattr(file, &options); // sets the parameters associated with file
    // Set up the communications options:
    // 9600 baud, 8-bit, enable receiver, no modem control lines

    options.c_cflag = BAUD | CS8 | CREAD | CLOCAL;
    options.c_iflag = IGNPAR | ICRNL; // ignore partity errors, CR -> newline
    options.c_oflag = 0;
    options.c_lflag = 0;

    options.c_cc[VTIME] = 0;
    options.c_cc[VMIN] = 1;

    tcflush(file, TCIFLUSH); // discard file information not transmitted
    tcsetattr(file, TCSANOW, &options); // changes occur immmediately

    return file;
}
```

2. void Uart_Close(int fd)

```
//used for close the port, prm is the file idx.

{
    close(fd);
}
```

3. void Uart_Flush(int fd)

```
//It clear remaining data bytes in the comm line. Prm is the file idx

{
        tcflush(fd, TCIOFLUSH);
}
```

4. int Uart_Send(int fd, unsigned char *buf, unsigned int buf_size)

```
//Used to send a buffer via serial port

{
        if(write(fd,buf,buf_size)<0)
                return RETURN_NOK;
        return RETURN_OK;
}
```

5. int Uart_Receive(int fd, unsigned char *buf, unsigned int buf_size, unsigned int vaitINms)

```
//It receives a buffer over serial cannel. It has timeout also

{
        int i = 0;
        for(i=0;i<buf_size;i++)
        {
                if(Uart_ReadByte(fd, &buf[i], vaitINms) != 1)
                        return RETURN_NOK;
        }
        return buf_size;
}
```

6. int CheckDataAvailability(int fileIndx, unsigned char blocking, long int time_sec, long int time_usec)

```
//It controls the availability of the channel.

{
   fd_set fds;
   struct timeval tv;
   int sel_ret = 0;
   int ret = 0;

   FD_ZERO(&fds);

   if(fileIndx != -1)
      FD_SET(fileIndx, &fds);
```

```
    if(!blocking)
    {
       tv.tv_sec = time_sec;
       tv.tv_usec = time_usec;
       sel_ret = select((fileIndx + 1), &fds, NULL, NULL, &tv);
    }
    else
    {
       sel_ret = select((fileIndx + 1), &fds, NULL, NULL, NULL);
    }

    if(sel_ret > 0)
    {
       if((fileIndx != -1) && (FD_ISSET(fileIndx, &fds)))
          ret = 1;
      else
          ret = sel_ret;
    }
    else if (sel_ret == -1)
       ret =  -1;

    return ret;

}
```

Extra Definitions

```
#define RETURN_OK   1
#define RETURN_NOK -1

#define DATA_AVAILABILITY_TIMEOUT 100000 //100 ms

typedef struct
{
          unsigned int baud_rate;
          unsigned char data_bits;
          unsigned char stop_bits;
          char parity;
          unsigned char rts_cts;
          unsigned char dtr_mode;
}Uart_Comm_Param;
```

Its Usage

```
int PORT_USBDEV = -1;

PORT_USBDEV = Uart_Init(4, 9600);//ADDED

if(PORT_USBDEV<0)//ADDED

        printf("serial port open fail\n");

unsigned int messageLen = 0; //ADDED

Uart_Flush(PORT_USBDEV);

receiveReturn = Uart_Receive(PORT_USBDEV, &message[0], 2, generalTimeout);
if(receiveReturn == 2)
        //data come
```

TCP/IP CLIENT CONNECTION

1. int TCP_Client_Connect(char *ip, int port)

```
//Used to create a link to connect a device via internet

{
        server = gethostbyname(ip);
        if (server == NULL) {
                printf("Socket Client: error - unable to resolve host name.\n");
                return -1;
        }

        socketfd = socket(AF_INET, SOCK_STREAM, 0);
        if (socketfd < 0){
                printf("Socket Client: error opening TCP IP-based socket.\n");
                return -1;
        }

        // clear the data in the serverAddress sockaddr_in struct
        bzero((char *) &serverAddress, sizeof(serverAddress));
        int portNumber = port;
        serverAddress.sin_family = AF_INET; //set the address family to be IP
        serverAddress.sin_port = htons(portNumber); //set port number to 80
        bcopy((char *)server->h_addr,(char *)&serverAddress.sin_addr.s_addr,
        server->h_length); //set address to resolved hostname address
        // try to connect to the server
```

```
        if (connect(socketfd, (struct sockaddr *) &serverAddress,
                sizeof(serverAddress)) < 0){
                printf("Socket Client: error connecting to the server.\n");
                return -1;
        }

        return 0;
}
```

2. int TCP_Client_Read(unsigned char *buffer, int len)

```
//Read buffer via internet link

{
        if (read(socketfd, buffer, len) < 0){
                printf("Socket Client: error reading from socket");
                return -1;
        }
        return len;
}
```

3. int TCP_Client_Write(unsigned char *buffer, int len)

```
//Write buffer via internet link

{
        if (write(socketfd, buffer, len) < 0){
                printf("Socket Client: error writing to socket");
                return -1;
        }
        return 0;
}
```

4. void TCP_Client_Close(int socket)

```
//Close the link. If it is not closed, connection automatically killed in a time.

{
        close(socketfd);
        socketfd = -1;
}
```

Extra Definitions

```
struct sockaddr_in serverAddress;
struct hostent *server;
int socketfd = -1;
```

Its Usage

```
if(TCP_Client_Connect("192.168.1.80", 2503) != -1)
{
	if(TCP_Client_Write(dummy_buffer, 2) != -1)
	{
		if(TCP_Client_Read(buffer, 2) == 2)
		{
			//do sth
		}
	}
}
```

USER LEDS

1. void LedON(unsigned char ledNumber)

```
//Power on for specified led.
{
	if(isInitExecuted==0)
		UserLedsInit();

	if(ledNumber==1)
	{
		fprintf(led1,"%d",1);
		fflush(led1);
	}else if(ledNumber==2)
	{
		fprintf(led2,"%d",1);
		fflush(led2);
	}else if(ledNumber==3)
	{
		fprintf(led3,"%d",1);
		fflush(led3);
	}else if(ledNumber==4)
	{
		fprintf(led4,"%d",1);
		fflush(led4);
	}
}
```

2. void LedOFF(unsigned char ledNumber)

```
//Power of for specified led.

{
	if(isInitExecuted==0)
		UserLedsInit();

	if(ledNumber==1)
	{
		fprintf(led1,"%d",0);
		fflush(led1);
	}else if(ledNumber==2)
	{
		fprintf(led2,"%d",0);
		fflush(led2);
	}else if(ledNumber==3)
	{
		fprintf(led3,"%d",0);
		fflush(led3);
	}else if(ledNumber==4)
	{
		fprintf(led4,"%d",0);
		fflush(led4);
	}
}
```

3. int UserLedsInit(void)

```
//Init the user leds. Actually, open specified files.

{
	led1 = fopen("/sys/class/leds/beaglebone:green:usr0/brightness", "w");
	led2 = fopen("/sys/class/leds/beaglebone:green:usr1/brightness", "w");
	led3 = fopen("/sys/class/leds/beaglebone:green:usr2/brightness", "w");
	led4 = fopen("/sys/class/leds/beaglebone:green:usr3/brightness", "w");

	if((led1==NULL)||(led2==NULL)||(led3==NULL)||(led4==NULL))
		return -1;

	//Slience
	fprintf(led4,"%d",0);
	fflush(led4);
	fprintf(led3,"%d",0);
	fflush(led3);
	fprintf(led2,"%d",0);
```

```
        fflush(led2);
        fprintf(led1,"%d",0);
        fflush(led1);

        isInitExecuted = 1;
        return 1;
}
```

4. void UserLedsClose(void)

```
//Close specified led's file.
{
        fclose(led1);
        fclose(led2);
        fclose(led3);
        fclose(led4);
}
```

Extra Definitions

```
FILE *led1, *led2, *led3, *led4;
unsigned char isInitExecuted = 0;
```

Its Usage

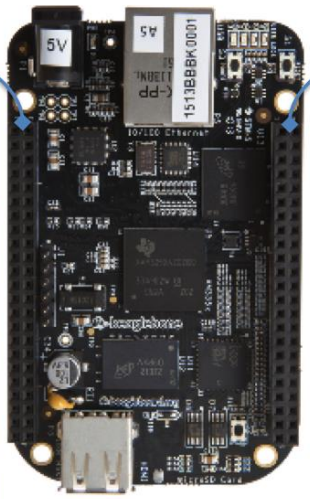
```
LedON(1);
LedON(2);
LedOFF(3);
```

PIN CONFIGURATION

Cape Expansion Headers

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETn
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPI0_CS0	17	18	SPI0_D1
I2C2_SCL	19	20	I2C2_SDA
SPI0_D0	21	22	SPI0_SCLK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SPI1_CS0
SPI1_D0	29	30	GPIO_112
SPI1_SCLK	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPPWM0
DGND	43	44	DGND
DGND	45	46	DGND



LEGEND
POWER/GROUND/RESET
AVAILABLE DIGITAL
AVAILABLE PWM
SHARED I2C BUS
RECONFIGURABLE DIGITAL
ANALOG INPUTS (1.8V)

P8

DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DAT0	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1