Title: New 8Fx Adapter IROS

Contents

1. PURPOSE/SCOPE

1.1 Purpose

This document describes the architecture and design of the new 8Fx adapter Firmware, Hardware.

1.2 Scope

This document describes all features, modes, and performance specifications that will be implemented in the product design for this project. All aspects of the design are covered, including System Hardware, Firmware.

The IROS is continually updated throughout the project. It begins as a design requirements document at PR2 and evolves into a design implementation document by PR3.

The main goal of this document is to capture the architecture and high-level design of new 8Fx adapter. This document also captures high-level use cases from the embedded module and demo tools. This document does NOT detail the required modules and interfaces as high-level modules of this application is documented in new 8Fx adapter EROS (*chma-52*) including a description of each module and interface.

1.3 IROS Document Format

The general layout of this document follows the Cypress document standard specified in 00-00021 CY Spec Format.

Sections 7, 8 and 9 are formatted as follows:

- Section 7: Overview of high-level product requirements, system architecture and project requirements.
- Section 8: Hardware, firmware and software design details.
- Section 9: System test implementation.

1.4 Definitions, Acronyms, and Abbreviations

2. RESPONSIBILITIES:

2.1 Defining the Process

The ICW MCU Apps group organization is responsible for defining the interfaces and modules this new 8Fx Adapter contains

2.2 Performing the Process

The ICW MCU Apps group organization is responsible for creating this document and ensuring through the review process that the tool is implemented as described in this document. The ICW MCU Apps group organization is then responsible for implementing the tools described in this document under the NPP process

2.3 Record Keeping

No records are required for this specification

3. REFERENCED DOCUMENTS:

3.1 00-00021 - Cypress Specification Requirements

3.2 00-00064 - Cypress Record Retention Policy

3.3 001-55157 - Firmware Design and Coding Rules and Best Practices

3.4 001-55229 - System New Product Development Methodology

3.5 001-56153 - System Hardware Development Methodology

4. MATERIALS: N/A

5. EQUIPMENT: N/A

6. SAFETY: N/A

7. CRITICAL REQUIREMENTS SUMMARY

7.1 System Description

A Solution Demonstration (SD) is a system-level implementation of key silicon product functionality, to be used by Sales and FAEs to facilitate the sales process when the silicon product reaches ES10. The SD development process must be simple and tightly focused to ensure that the SD is completed at silicon ES10 and will be used for early customer engagements, demonstration and initial customer evaluations.

This specification documents the requirements of new 8fx adapter. This SD will use new 8fx adapter board, being tracked through PM ID xxxxx.

7.2 Project objectives

7.2.1 Performance evaluation

Due to the MCU of 8Fx adapter MB2146-07-E is on EOL. A new adapter is required for customer. We need to re-design the hardware and firmware.

7.2.2 Hardware adjustment

4 layers boards.

7.3 System Technology Requirements

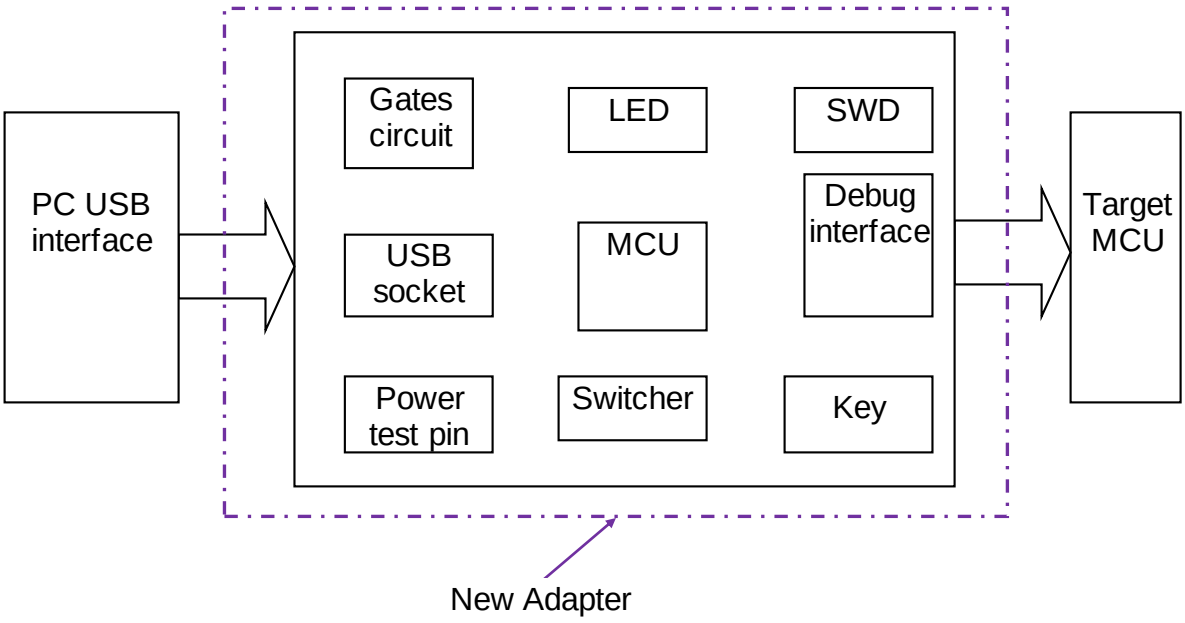
The detail please reference New 8Fx Adapter EROS CHMA-52.

7.3.1 Cypress devices required

FM0p chip: S6E1C32C0AGV20000.

7.4 System Architecture

7.4.1 Top-level System Block Diagram

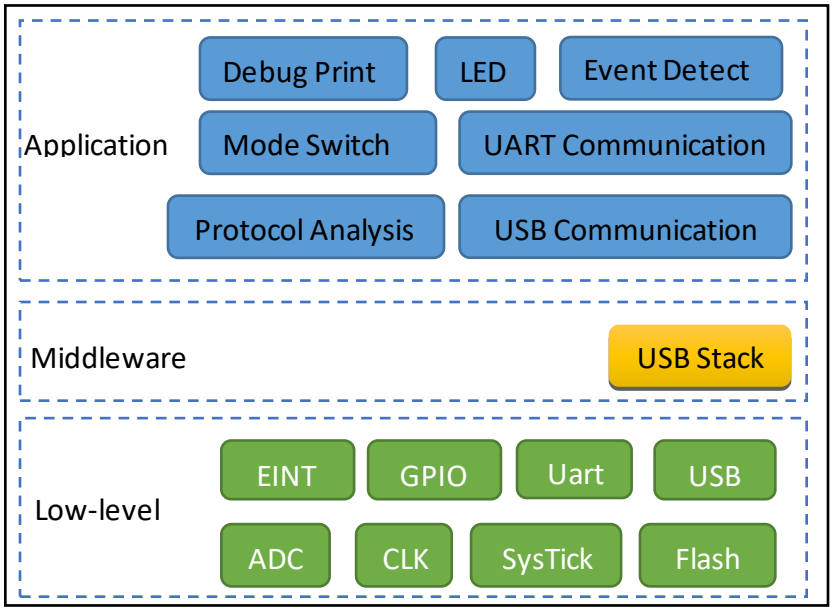


The features:

- adaptor supports USB2.0 full speed
- Monitor of target MCU power supply UVCC
- Mode control by single DBG line
- Reset circuit on new adaptor
- LED indicator on new adaptor
- Supports single line debug
- Power supply to target board is optional
- Detect of target MCU power supply UVCC
- Only 4 lines are needed between adaptor and target board
- A power switch is on adaptor
- A button is used to switch MCU mode
- A SWD interface uses to debug

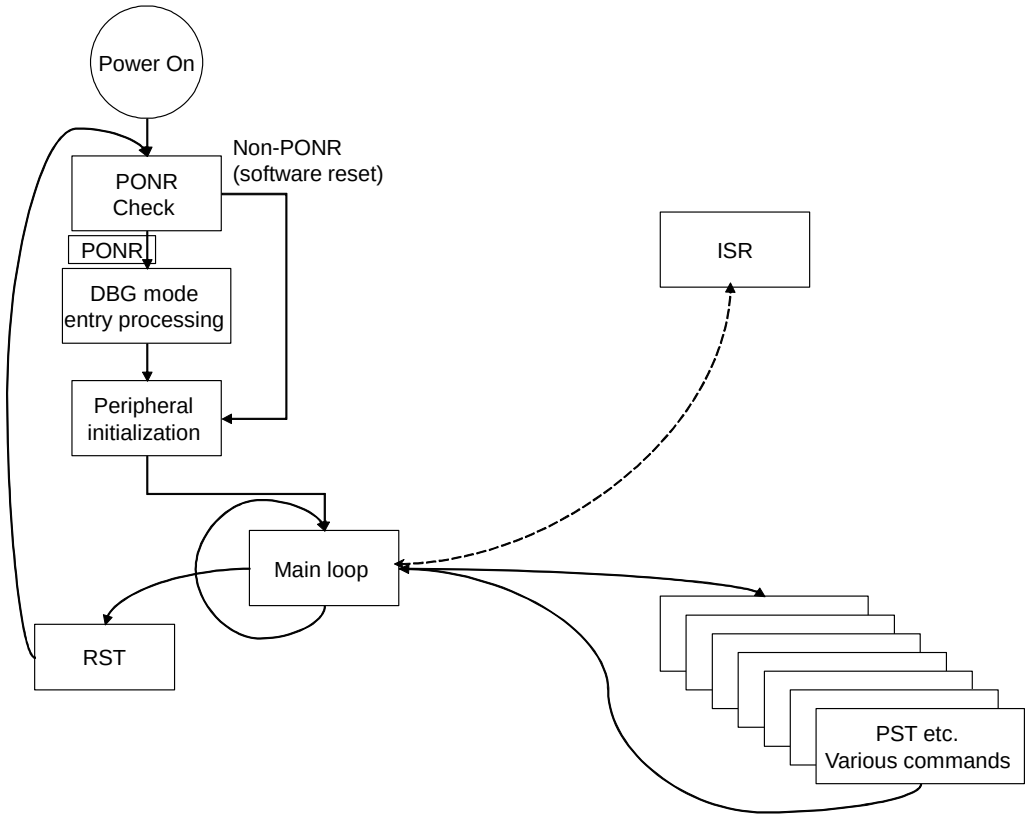
7.4.1.1 Firmware platform

7.4.1.1.1 High-level Firmware Block Diagram



In this project, USB and UART are used to communicate with PC tool and target board. Protocol Analysis is used to analysis the data from USB and then invoke related function to implement. ADC and external interrupt are used to monitor the event then control target MCU into different mode. SysTick is used timeout detection etc. Another UART uses to print the debug information (reserve function). LEDs use to indication the system status.

7.4.1.2 Firmware functional flow



7.4.1.3 Software platform

None

7.5 Traceability Matrix

The Product Traceability Matrix correlates the system requirements defined in the EROS to the design requirements in the IROS. All EROS requirements must have an associated IROS requirement.

7.5.1 Apportioning of Requirements

All requirements will be met by the first release.

7.6 Compliance requirements

No compliance testing is planned.

7.7 Developer-Related Requirements

7.8 Engineering Assumptions and Risks

8. DESIGN DETAILS

8.1 System Product Hardware Block Requirements

8.1.1 Main Board Details

The Main board is used to implement all the adapter functions, it consists of 2 interfaces, 2 buttons and LED indication.

8.1.1.1 Interface description

■ USB Interface

The mini-B type USB cable (CN2) is used to connect to PC host. Communication with PC tool is realized through USB interface.

■ Debug Interface

The debug interface (CN1, 2x5 pins, 2.54mm) connects the adapter and target board.

Connector pin number	Input / output	Target MCU connection pin name	Function	Remarks
1	Adapter ← MCU	UVCC	User power supply input	Connected to the MCU Vcc pin.
2	-	GND	Vss pin	Connected to the MCU Vss pin.
3	Adapter → MCU	RSTIN	Tool reset output	Not use at present
4	Adapter ← MCU	RSTOUT	User System reset output	Not use at present
5	-	RSV	-	-
6	Adapter → MCU	VCC	BGMA power output	BGMA supplies power to target board Vcc
7	-	RSV	-	-
8	Adapter ← MCU Adapter → MCU	DBG	Communication line	1-line UART
9	-	RSV	-	-
10	-	RSV	-	-

■ Switch

The switch (SW1) is used to turn on/off this adapter.

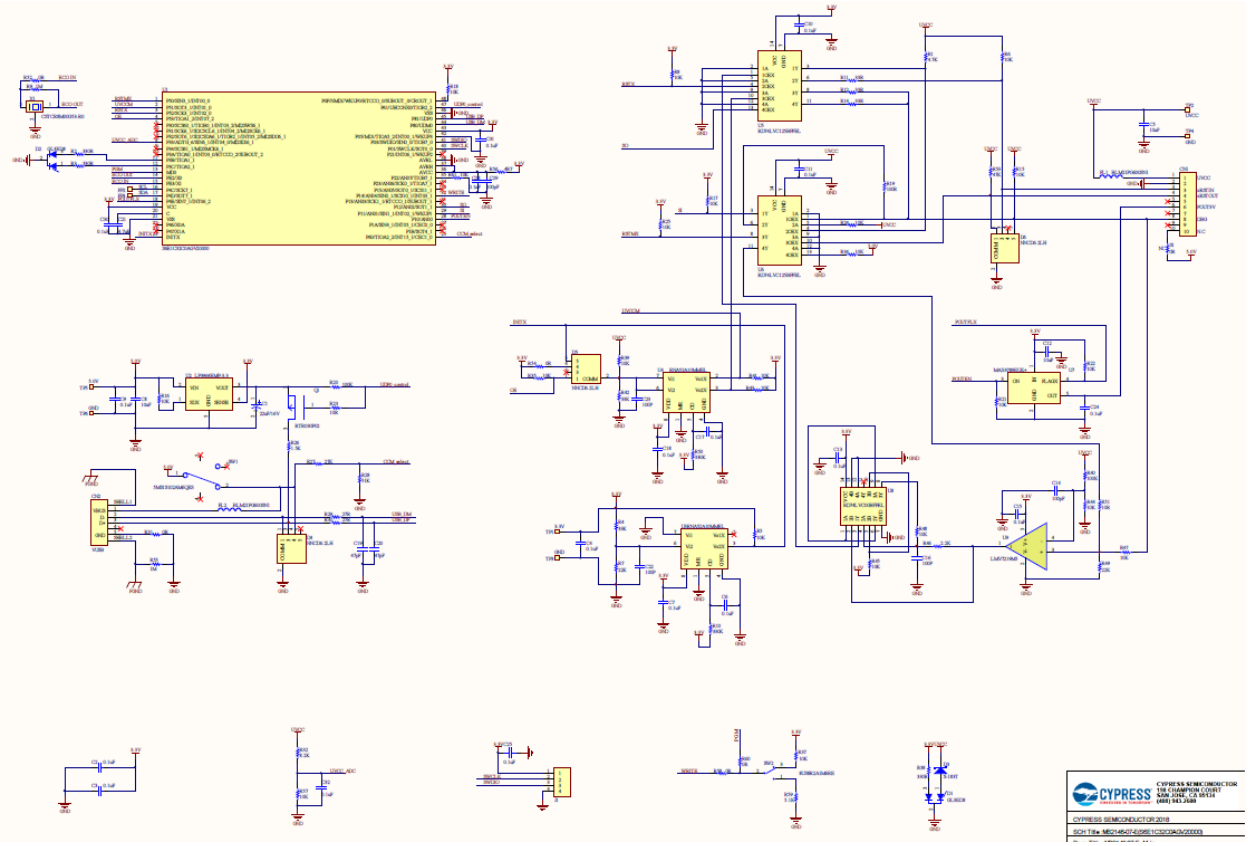
■ Button

The button (SW2) is used to switch MCU to user or programming mode.

■ LEDs

The LEDs are used to indicate system status.

8.1.1.2 Schematic



8.1.1.3 Clock and Block Timing requirements

No special timing requirements.

8.1.1.4 Environmental requirements

Design to 0-85 C.

Test at room temp only.

8.1.1.5 Design trade-offs

8.2 Firmware Design

For the firmware design, it's have been divide by functions.

- OCD (On-chip Debugger) Mode
- USB Communication
- UART Communication
- LED Indication

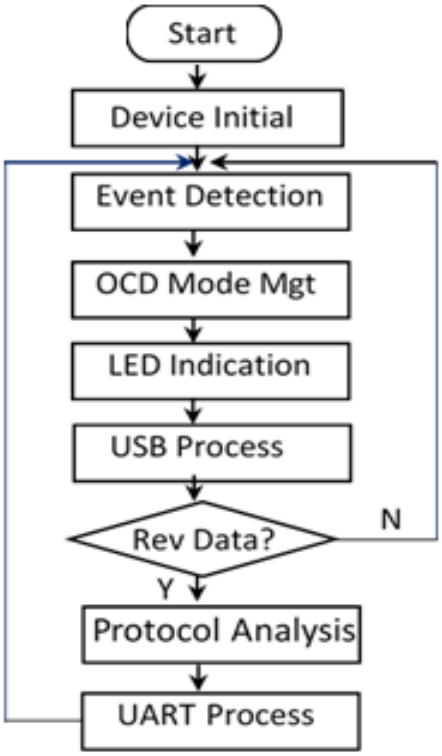
- Protocol Analysis
- Event Detection

8.2.1 Main Flow-chat

After power on, the adapter initializes all the peripheral devices includes system clock, USB, UART, Timer, external interrupt, GPIO etc.

When detected connected to the target, then control the target MCU to enter to debug mode via DBG pin.

Once USB received the vendor request from EP0, after relevant processing, reply directly. If the data came from EP1, check whether the packets are valid USB command. If yes, parses and invokes the related uart command to transmit to the target MCU.



8.2.1.1 USB Command List

Function name	FC	possible process					Function explanation
		8FX	LPC-0.18		LPC-0.35		
			*1	*2	*1	*2	
● Initialization /environment							
START_EML	0000H	○	○	×	○	×	Beginning of monitor
EXIT_EML	0100H	○	○	○	○	○	End of monitor

INIT_EML	0200H	○	○	×	○	×	Initialization of monitor
SET_CLK	1824H	○					Setting of clock improvement mode
	1826H		○	○	○	○	Notification of user system nuclear power plant clock (reservation)
BGMA_POW	1928H		○	○			Enable/disable power supply for target MCU
● MCU register access							
WRITE_REG	2000H	○	○	×	○	×	Writing of MCU register
READ_REG	2100H	○	○	×	○	×	Reading of MCU register
READ_REGLIST	2200H	○	○	×	○	×	Reading of MCU register row
● Memory access relation							
WRITE_MEM	3000H	○	○	×	○	×	Writing in memory
READ_MEM	3100H	○	○	×	○	×	Reading from memory
MON_RAM	3200H		○	×			Monitoring RAM area
(*2)							
ERASE_FLASH	3600H	○					Deletion of flash
● Execution							
READ_BRK_STAT	4000H	○	○	×	○	×	Reading of MCU break status
EXE_MCU	4100H	○	○	×	○	×	Continuous execution/step execution of target MCU
RESET_MCU	4200H	○	○	×	○	×	Target MCU reset
ABORT_MCU	4300H	○	○	×	○	×	Compulsion break of target MCU
● Break relation							
SET_BRK	5000H	○	○	×	○	×	Setting of breakpoint
DEL_BRK	5100H	○	○	×	○	×	Deletion of breakpoint
● RAM monitor relation							
SET_RCK_ADR	A128H		○	×			Set RAM monitor Address
READ_RCK_SMDAT	A228H		○	×			Read monitoring data
● LPC system command							
TRIM_CR	E026H		○	○	○	○	CR trim of target system
USER_CMD	E126H		○	×	○	×	Debugging program execution on RAM
● Commonness (basis) part							
CNG_MON	F000H	○					Switch of monitor program
LOAD_ENV	F400H	○	○	×	○	×	Reading of system environment
● BGM adaptor control part							

CNG_ADPT_MODE	FE00H	System	The operation mode of the BGM adaptor is changed.
LOAD_ADPT_ENV	FF00H	System	The operation mode which can be used with the BGM adaptor is acquired.

8.2.1.2 API Reference

8.2.1.2.1 Main function

```
void bgma_app_main(void)
{
    s_ReceivedSize = 0;

    /* initialize USB function */
    epcs_initialize();

    InitSystick();
    bgma_io_initial();
    uart_init(UART_BAUD_RATE_62500);
    enable_uvccm_int();
    enable_rstmx_int();
    enable_poutflx_int();

    while(epcs_clear_read() < 0);

    GpioIpin_Put( GPIO1PIN_P3B, 1);  // USB connection successful

    /* infinite loop */
    while(1) {
        usb_transfer_task();
    }
}
```

8.2.1.2.2 System function

The BGM Adapter system function is usb_transfer_task. This function includes idle process, receive USB data process and send USB data process three parts.

In idle process part, the system executes all of interrupt handle, records and deals system flag, system will jump out when system detect USB interrupt

```
static void usb_transfer_task(void)
{
    int32_t epcs_ret;
    uint16_t size;
    static bool GreenLed = 0;

    while(1) {
        bgma_idle_process();
        epcs_ret = epcs_check_read();
        if(epcs_ret > 0){
            break;
        } else if(epcs_ret < 0) {
            /* Source MCLK 1/2 */
            //CPU_MAP_IO_CSELR_CKS = 0;
            //while((CPU_MAP_IO_CMONR_CKM != 0) ||(IO
            /* Reset */
            NVIC_SystemReset();
            /* wait */
            while(1){};
        } else {
            ;
        }
    }
```

In receive process part, system gets USB data, deals USB and UART data, and controls Green LED for communicating indication.

```
    size = (epcs_ret < APP_TRANSFER_BUFFER_SIZE) ? epcs_ret : APP_TRANSFER_BUFFER_SIZE;
    epcs_ret = epcs_read_data(s_AppReceive.Buffer, 0, size);
    if (epcs_ret > 0) {
        s_AppReceive.Size = epcs_ret;
        receive_data_process();
        s_AppReceive.Size = 0;
        /* CHMA added */
        UsbCommFlag = 1;  // Indication USB is receiving/sending data
        ClrTimeOut(&UsbCommTimer);
        Gpiolpin_Put( GPIO1PIN_P3B, GreenLed);  // Blink the green led
        GreenLed = !GreenLed;
    }
```

In send process, system send data to PC.

```
    if (s_AppSend.Size > 0) {
        // epcs_ret = epcs_check_write();
        epcs_ret = epcs_clear_write();
        if (epcs_ret == 0) {
            epcs_ret = epcs_write_data(s_AppSend.Buffer, s_AppSend.Size);
            if (epcs_ret == s_AppSend.Size) {
                epcs_write_data(s_AppSend.Buffer, 0);
                s_AppSend.Size = 0;
            }
        }
    }
    return;
}
```

8.2.1.2.3 Sub System function

This function is for USB and UART data deal, it is from USB receive_data_process, it includes receiving USB data from PC, parsing USB data, communicating with target board with UART and sending USB data for responding PC.

```
static void hostif_command_process(uint8_t* data, uint16_t length, uint8_t fn)
{
    uint16_t response_size = 0;
    uint8_t* response_data = NULL;
    const CMDTBL *tbl;
    uint16_t command = (((uint16_t)data[1]) | (((uint16_t)data[0]) << 8));

    (void)length;
    set_usbbuf_pointer(data);
    tbl= commandtable;
    while (tbl->fc != 0xFFFFU) {                /* parsing FC */
        if (command == tbl->fc) break;
        tbl++;
    }
    if (command == EXIT_MON) monitor_state &= ~MON_START;

    /* send VCCERR if LOWPOWERIDT is true; send NOWTRIM if BGMA still in trim process */
    if (((monitor_state & TRIMMING) == TRIMMING) || ((power_state & LOWPOWERIDT) == LOWPOWERIDT)){
```

Data Structure CMDTBL storages USB command and 24 callback functions with UART communication.

```
if (((monitor_state & TRIMMING) == TRIMMING) || ((power_state & LOWPOWERIDT) == LOWPOWERIDT)){
    if ((command == LOAD_ADPT_ENV)||(command == CNG_ADPT_MODE)){
        (tbl->func)();
    }else{
        if ((power_state & LOWPOWERIDT) == LOWPOWERIDT){
            set_errcode(VCCERR);
            power_state &= ~LOWPOWERIDT;
        }else set_errcode(NOWTRIM);
    }
}else if ((power_state & LOW_POWER) == LOW_POWER || (power_state & OVER_CURRENT) == OVER_CURRENT){
    if ((command == LOAD_ADPT_ENV)||(command == CNG_ADPT_MODE)){
        (tbl->func)();
    } else if(command == BGMA_POW){
        if((power_state & LOW_POWER) == LOW_POWER){
            (tbl->func)();
        }
    }
    else set_errcode(VCCERR);
} else {

      if ((monitor_state & MCU_RUN) == MCU_RUN){
          switch (command){ /* some command is different in MCU_RUN mode */
              case READ_BRK_STAT:
                  (tbl->func)();
                  break;   /* break status: 00 00 ff ff ff ff 00 00 00 00 */

              case ABORT_MCU:
                  (tbl->func)();
                  /* mcu_state = MCU_BREAK; */
                  break;   /* force to stop the free run mode */

              case LOAD_ADPT_ENV:
                  (tbl->func)();
                  break;

              case CNG_ADPT_MODE:
                  (tbl->func)();
                  break;
```

8.2.1.2.4 24 USB Commands Process

```
CMDTBL const commandtable[] = {
    { 0x0000U, proc_start_mon,},
    { 0x0100U, proc_exit_mon,},
    { 0x0200U, proc_init_mon,},
    { 0x1826U, proc_set_clk,},
    { 0x1928U, proc_bgma_pow},
    { 0x2000U, proc_write_reg,},
    { 0x2100U, proc_read_reg,},
    { 0x2200U, proc_read_reglist,},
    { 0x3000U, proc_write_mem,},
    { 0x3100U, proc_read_mem,},
    { 0x3600U, proc_erase_flash,},
    { 0x4000U, proc_read_brk_stat,},
    { 0x4100U, proc_exe_mcu,},
    { 0x4200U, proc_reset_mcu,},
    { 0x4300U, proc_abort_mcu,},
    { 0x5000U, proc_set_brk,},
    { 0x5100U, proc_del_brk,},
    { 0xA128U, proc_set_rck_adr,},
    { 0xA228U, proc_read_rck_smdat,},
    { 0xE026U, proc_trim_cr,},
    { 0xE126U, proc_user_cmd,},
    { 0xF400U, proc_load_env,},
    { 0xFE00U, proc_cng_adpt_mode,},
    { 0xFF00U, proc_load_adpt_env,},
    { 0xFFFFU, proc_other,},
};
```

8.2.1.2.4.1 proc_start_mon

Identify the monitor debug begins and return monitor program version.

Command frame:
00, 00

Return frame
00, XX, XX, XX ---- Normal end
| | ∟ Release number of monitor program
| ∟ Level number of monitor program
∟ Version number of monitor program
Please refer “18um LPC OCD System External Spec. pdf” chapter 8 for monitor program version details.
01, XX, 00, 00 ----NG end
∟ Command error type

```
static void proc_start_mon(void)
{
    UART_COMMAND cmd;
    uint16_t offset;

    /**leak UVCC power*/
    //bFM_GPIO_DDR1_P1 = 0;  //SI->P11
    //bFM_GPIO_DDR5_P0 = 0;  //RSTMX->P50
    bFM0P_GPIO_ADE_AN01 = 0;  // Disable pin ADC function
    Gpiolpin_InitIn ( GPIO1PIN_P11, Gpiolpin_InitPullup( 0u ) )
    Gpiolpin_InitIn ( GPIO1PIN_P50, Gpiolpin_InitPullup( 0u ) )
    /**end************/

    power_state &= ~USER_RST;  /* clear user reset flag at the
    monitor_state &= ~CLKUP_ON;  /* set clock up mode off at th
    monitor_state |= MON_START;  /* monitor start, monitor will
    if(version_num == VER_35) {
        tar_SYCC = DEF_SYCC;
        tar_SYCC2 = DEF_SYCC2;   /* set clock mode as default
        tar_PLLC = DEF_PLLC;
    } else if(version_num == VER_1_18) {
        tar_SYCC = DEF_SYCC_1_18;
        tar_SYCC2 = DEF_SYCC2_1_18;   /* set clock mode as defa
        tar_PLLC = DEF_PLLC;
    }

    put_normal();            /* USBbuf[FRMOUT_ENUM] = 0 */
    RegBuff_Init(&cmd);
```

8.2.1.2.4.2 proc_exit_mon

Identify the monitor debug end

Command frame:
01, 00

Return frame
00 ---- Normal end
01, XX, 00, 00 ----NG end
└ Command error type

```
static void proc_exit_mon(void)
{
    monitor_state &= ~MON_START; /* EMU stop */
    put_normal();
    if(IO_UVCCM_IN){
       if(IO_POUTEN_IN) IO_POUTEN_OUT = 0; /* if pow
    }
    if(!VER_2_18(version_num)) FlagForGoTime = 3; /*

    /**leak UVCC power*/
    bFM_GPIO_PDOR1_P1 = 0; //SI->P11
    bFM_GPIO_DDR1_P1 = 1;  //SI->P11

    IO_RSTMX_OUT = 0;      //RSTMX->P50
    bFM_GPIO_DDR5_P0 = 1;  //RSTMX->P50
    /**end************/
    return ;
}
```

8.2.1.2.4.3 proc_init_mon

Initialize the BDSU register.

Command frame:
02, 00

Return frame
00 ---- Normal end
01, XX, 00, 00 ----NG end
∟ Command error type

```
static void proc_init_mon(void)
{
    UART_COMMAND cmd;
    uint16_t InitData;
    uint8_t i;

    put_normal();
    /* set initial value */
    InitData = 0x0000U;
    /* clear BPR0 (Break Point Register) */
    SET_UARTCMD_ADDR(cmd,InitData);
    if (write_BDSU_reg(&cmd,BDSU_BPR0) !=NORMAL){
        set_errcode(cmd.RC);
        return ;
    }
    /* clear BPR1 */
    SET_UARTCMD_ADDR(cmd,InitData);
    if (write_BDSU_reg(&cmd,BDSU_BPR1) !=NORMAL){
        set_errcode(cmd.RC);
        return ;
    }
    /* clear BCR,BSTR,BFR 3bytes */
    for (i=0; i<3; i++){
        SET_UARTCMD_ADDR(cmd,InitData);
        if (write_BDSU_reg(&cmd,BDSU_BCR+i) != NORMAL){
            set_errcode(cmd.RC);
            return ;
        }
    }
```

8.2.1.2.4.4 proc_set_clk

The command is to select clock-up or not. Whenever the clock up mode is turned on, whenever the user program breaks, BGMA adjusts target's clock. Thus, delay is not needed in UART communication.

Command frame:
18, 26, XX, XX
| ∟ external clock frequency (e.g. 10MHz: 0A)
∟ clock optimization flag (00: disable; 01: enable)

Return frame
00 ---- Normal end
01, XX, 00, 00, ---- NG end
∟ Command error type

```
static void proc_set_clk(void)
{
    uint16_t offset;
    uint8_t ClkUpMode;
    UART_COMMAND cmd;

    put_normal();
    offset = 0;
    ClkUpMode = get_paramb(offset++);  /*
    mainOSC_freq = get_paramb(offset);
    if (ClkUpMode & 0x01) {
        monitor_state |= CLKUP_ON; /* ClkU
        if (clock_up(&cmd)){
            set_errcode(cmd.RC);
            return;
        }
        delay_time = 0x0000U;           /*
        if(version_num == VER_35){
            tar_SYCC = DEF_SYCC;
            tar_SYCC2 = DEF_SYCC2;    /* se
            tar_PLLC = DEF_PLLC;
        }
        else if(version_num == VER_1_18){
            tar_SYCC = DEF_SYCC_1_18;
            tar_SYCC2 = DEF_SYCC2_1_18;
            tar_PLLC = DEF_PLLC;
        }
    }
    else monitor_state &= ~CLKUP_ON; /* Cl
```

8.2.1.2.4.5 proc_bgma_pow

Enable/disable BGMA power supply

Command frame:
19, 28, XX, XX
  | └ Power type.00: Disable power supply; 01: Enable power supply
  └ Reserved

Return frame
00, XX --- Normal end
  └ 00: power operation OK; 01: power operation NG.
01, XX, 00, 00 ---- NG end
  └ Command error type

```
static void proc_bgma_pow(void)
{
    uint16_t offset;
    uint8_t PowerType,OperationFlag;

    put_normal();
    offset = 0;
    offset++;  /* ignore reserve byte */
    PowerType = get_paramb(offset);

    if(PowerType == 0x00)
    {
        if(IO_UVCCM_IN)/* if UVCC is high, disable
        {
            IO_POUTEN_OUT = 0;
        }
        else
        {
            if(IO_POUTEN_IN)
                IO_POUTEN_OUT = 0;
        }
        OperationFlag = 0;
    }
    else if(PowerType == 0x01)
    {
        if(!IO_UVCCM_IN) /* if UVCC is low, enable
        {
            IO_POUTEN_OUT = 1;
            OperationFlag = 0;
```

8.2.1.2.4.6 proc_write_reg

Write user register which is defined by user.
Command frame:
20, 00, XX, XX, XX, XX, 00, 01, XX, XX, XX, XX
| | ∟ write data
| ∟ Write length (Fixed to 0x01)
∟ Written register ID (see table 4.5.5-1)

Register number	Name
0H	A
1H	T
2H	IX
3H	SP
4H	EP
5H	PS
6H	PC
10H	R0
11H	R1
12H	R2
13H	R3

14H	R4
15H	R5
16H	R6
17H	R7

Return frame
00, 00, 01---- Normal end
∟ Write length (fixed to 0x01)
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_write_reg(void)
{                                           /* need modified later */
    uint8_t WriteState;
    uint16_t offset;
    uint32_t WriteRegID;
    uint16_t WriteNumber;
    uint32_t WriteData;

    offset = 0;
    put_normal();
    WriteRegID = get_paramw(offset);        /* get register ID */
    offset += sizeof(uint32_t);

    WriteNumber = get_paramh(offset);
    if (WriteNumber != 0x01U){              /* only one register should be writte
        set_errcode(PARAER);                /* parameter error */
        return ;
    }
    offset += sizeof(uint16_t);

    WriteData = get_paramw(offset);         /* get write data */
    WriteState = write_user_reg((uint16_t)WriteData,(uint16_t)WriteRegID);
    if (WriteState != NORMAL){
        set_errcode(WriteState);
        return ;
    } else {
        offset = 0;
```

8.2.1.2.4.7 proc_read_reg

Read user register command

Command frame:
21, 00, XX, XX, XX, XX, 00, 01
| ∟ Read length (fixed to 0x01)
∟ Read register ID (see table 4.5.5-1)

Return frame
00, 00, 01, XX, XX, XX, XX ---- Normal end
| ∟ Read data

∟ Read length (fixed to 0x01)
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_read_reg(void)
{
    uint8_t ReadState;
    uint16_t offset;
    uint32_t ReadRegID;
    uint16_t ReadNumber;
    uint16_t ReadRegData;/* should be unsigned long type in return frame */

    offset = 0;
    put_normal();
    ReadRegID = get_paramw(offset);        /* get register ID */
    offset += sizeof(uint32_t);

    ReadNumber = get_paramh(offset);
    if (ReadNumber != 0x01U){          /* only one register should be read */
        set_errcode(PARAER);               /* parameter error */
        return ;
    }
    ReadState = read_user_reg(&ReadRegData,(uint16_t)ReadRegID);
    if (ReadState != NORMAL){
        set_errcode(ReadState);
        return ;
    } else {
        offset = 0;
        put_paramh(offset,ReadNumber);        /* return read number */
        offset += sizeof(uint16_t);
```

8.2.1.2.4.8 proc_read_reglist

Command frame:
22, 00, XX, XX, XX, XX, XX, …… XX, XX, XX, XX
| | ∟ Register ID (4*15 bytes)
| ∟ Read Number (Fixed to 0x0F)
∟ Reserved

Return frame
00, 00, 0F, XX, XX, XX, XX, …… XX, XX, XX, XX ---- Normal end
| ∟ Read data (The total number is 0x0F)
∟ Read length (Fixed to 0x0F)
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_read_reglist(void)
{
    uint8_t ReadState;
    uint16_t offset;            /* for read data from USB
    uint16_t ReadRegID;         /* number we should read
    uint16_t ReadRegData;       /* temp readed data */
    uint16_t TempPS;            /* temp PS value for R0~R

    //wait60us(1);
    offset = 0;
    put_normal();
    offset++;                           /* ignore flag */
    if (get_paramh(offset) != REG_LIST_NUM){    /* read reg
        set_errcode(PARAER);
        return ;
    }

    offset = sizeof(uint16_t);
    for (ReadRegID=A_ID; ReadRegID<=PC_ID; ReadRegID++){
        ReadState = read_user_reg(&ReadRegData,ReadRegID);
        if (ReadState != NORMAL){
            set_errcode(ReadState);
            return ;
        }
        put_paramw(offset,(uint32_t)ReadRegData);
        offset += sizeof(uint32_t);
        if (ReadRegID == PS_ID) TempPS = ReadRegData; /* re
    }
    ReadState = read general reglist(TempPS,&offset);
```

8.2.1.2.4.9 proc_write_mem

Write data into memory.

Command frame:
30, 00, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX…… XX, XX, XX
| | | ∟ Write number ∟ Write data ∟ Type
| | ∟ Start address
| ∟ Reserved

Type:
0x00: RAM
0x01: DSU
0xFF: Flash

Write number:
The USB data buffer length is 1024
The USB command length is (1024-4-7)/2=506
Max number of data = 506 – 2(FC) – 2(Reserved) – 4 (Start address) – 4 (Write number) – 1 (type) = 493

Return frame
00, XX, XX, XX, XX ---- Normal end
|
└ Write length
01, XX, XX, XX, XX, XX, XX ---- NG end (write error)
| | └ Error address
| └ Write error flag (bit0 = 1: error occur, bit0=0: no error)
└ Command error type
01, XX, 00, 00 ---- NG end (the other errors)
└ Command error type

```
static void proc_write_mem(void)
{
    UART_COMMAND cmd,cmd1;
    UART_COMMAND_8 cmd8;
    uint16_t offset;
    uint32_t StartAddress;
    uint32_t WriteNumber;
    uint8_t TempDCR;
    uint16_t TempCounter,i;
    uint8_t j;
    uint8_t ExtendedCommand;
    uint8_t WriteState;
    uint8_t WR8TemperaPara;
    uint8_t *ReadDToCMD8;
    uint32_t StartAddrTrack=0;
    uint8_t BDSWrit;

    offset = 0;
    put_normal();
    TempDCR = 0;
    WriteState = NORMAL;
    /* ignore memory data size + attribution */
    offset += sizeof(uint8_t) + sizeof(uint8_t);
    StartAddress = get_paramw(offset);  /* get start address */
    offset += sizeof(uint32_t);
```

8.2.1.2.4.10 proc_read_mem

Read data from memory.

Command frame:
31, 00, XX, XX, XX, XX, XX, XX, XX, XX, XX
| | | | └ Type
| | | └ Read number
| | └ Start address
| └ Reserved

Type:
0x00: RAM or Flash
0x01: BDSU

Read number;
Max number of data = 506 – 1(TYPE) – 4(Read length) = 501

Return frame
00, XX, XX, XX, XX, XX, XX, …., XX, XX---- Normal end
| ∟ Read data
∟ Read length

01, XX, 00, 00 ---- NG end (the other errors)
∟ Command error type

```
static void proc_read_mem(void)
{
    UART_COMMAND cmd;
    UART_COMMAND_16 cmd16;
    uint16_t offset;
    uint32_t StartAddress;
    uint32_t ReadNumber;
    uint16_t TempCounter;
    uint8_t i;
    uint8_t ExtendedCommand;
    uint8_t WR16TemperaPara;
    uint16_t end_address = 0xFFFF;
    uint32_t Readaddress;

    offset = 0;
    put_normal();
    /* ignore mem data size + attribution */
    offset += sizeof(uint8_t) + sizeof(uint8_t);
    StartAddress = get_paramw(offset);      /* get the start address */
    offset +=sizeof(uint32_t);
    ReadNumber = get_paramw(offset);
    if((ReadNumber > MAX_READ_LEN)||((StartAddress + ReadNumber) > 0x00010000UL
    {
        set_errcode(OVRFLW);
        return ;
    }
```

8.2.1.2.4.11 proc_erase_flash

Erase Flash.

Command frame:
36, 00, XX, XX, XX, XX, XX
| ∟ Erase address
∟ Erase type

Erase type:
0x00: Chip erase
Else: Sector erase
Erase address:
When chip erase, the address should be inside whole Flash memory; when sector erase the address should be inside sector memory.

Return frame
00 ---- Normal end
01, XX, 00, 00---- NG end
└ Command error type

```
static void proc_erase_flash(void)
{
    UART_COMMAND cmd;
    uint32_t SectorAddress;
    uint16_t offset;
    uint8_t TempDCR;
    uint8_t cnt;
    uint8_t EraseState;
    uint8_t UartState;

    offset = 0;
    put_normal();
    TempDCR = 0;
    EraseState = NORMAL;

    if ( read_BDSU_reg(&cmd,BDSU_DCR) != NORMAL )  /* read DC
    {
        set_errcode(cmd.RC);
        return ;
    }

    if ( cmd.AddrL & 0x01U )
    {
        TempDCR = 0x01U;    /* wild register has been set */
        cmd.AddrH = 0;
        cmd.AddrL = 0;
```

8.2.1.2.4.12 proc_read_brk_stat

Read the break state of target MCU.

Command frame:
40, 00

Return frame
00, XX, XX, XX, XX, XX, XX, XX, XX, XX---- Normal end
| | └ Break type
| └ Break address (Only valid when program stops)
└ Break status (Bit7 = 0: Run, Bit7 = 1: Stop)

Break type:
bit30: Force break
bit29: Step break
bit28: SW break
bit25: HW break 1
bit24: HW break 0
bit14: Execute error
else: Reserved
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_read_brk_stat(void)
{
    UART_COMMAND cmd;
    uint16_t offset;
    uint32_t BrkFactor;
    uint16_t Read_PC;
    uint8_t Read_BFR;

    offset = 0;
    Read_PC = 0;
    put_normal();
    if ((monitor_state & MCU_RUN) == MCU_RUN)
    {
        /* in free run mode, return 00 00 ff ff ff ff
        put_paramb(offset++,NO_BREAK);            /* s
        put_paramw(offset,0xFFFFFFFFUL);
        offset += sizeof(uint32_t);
        put_paramw(offset,0);
        offset += sizeof(uint32_t);
        put_frmlen(offset);
        }
    else
    {
        if (read_BDSU_reg(&cmd,BDSU_PCER) != NORMAL)
        {
            set_errcode(cmd.RC);
            return ;
```

8.2.1.2.4.13 proc_exe_mcu

Execute program in target MCU

Command frame:
41, 00, XX, XX, XX
| | ∟ Reserved (Fixed to 0x00)
| ∟ Watch dog enable
∟ Execute type
Execute type:
bit0-3 = 0000: Run, bit0-3 = 0001: Step in
bit4-7: Reserved

Watch dog enable:
bit1 = 0: enable watch dog, bit1 = 1: disable watch dog

Return frame
00 ---- Normal end
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_exe_mcu(void)
{
    UART_COMMAND cmd;
    uint8_t ReadBCR;
    uint8_t ExecuteType;
    uint8_t WatchDogFlag;
    uint16_t offset;
    uint8_t ClockSetState;

    /*reserve user clock*/
    if( VER_2_18(version_num) )
    {
        ResetFlag = 0;
    }
    if( !VER_2_18(version_num) )
    {
        FlagForGoTime++;
        if( FlagForGoTime != 0x02 )
        {
            FlagForGoTime=3;    /*avoid first go*/
        }
        else if(FlagForGoTime == 2)  /*the first time ente
        {
            if ( (monitor_state & CLKUP_ON) != CLKUP_ON )
            {
                if ( clock_setting(&cmd,WRITE) )  /* recov
                {
                    set_errcode(cmd.RC);
```

8.2.1.2.4.14 proc_reset_mcu

Reset target MCU, and initial general register list.

Command frame:
42, 00

Return frame
00 ---- Normal end
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_reset_mcu(void)
{
    UART_COMMAND cmd;
    uint8_t WriteState;
    put_normal();

    if(VER_2_18(version_num))    ResetFlag = 1;
    if( (version_num == VER_1_18 && level_num == 2)||(VER_2_18(version_num)) )
    {
        if( uart_baudrate_up(&cmd, UART_BRT_RCV) )
        {
            set_errcode(cmd.RC);
            return ;
        }

    }
```

8.2.1.2.4.15 proc_abort_mcu

Function:
Abort target MCU, after implementing this command, DBG pin will be pulled down for a period time to force target MCU enter break status.

Command frame:
43, 00

Return frame
00 ---- Normal end
01, XX, 00, 00 ---- NG end
└ Command error type

```
static void proc_abort_mcu(void)
{
    put_normal();
    /* DBG pin keep low about 683us */
    send_break();
}
```

8.2.1.2.4.16 proc_set_brk

Set hardware break point.

Command frame:
50, 00, XX, XX, XX, XX, XX, XX, XX
| | | | └ Setting Flag (Fixed to 0x00)
| | | └ break index (0x00 or 0x01)
| | └ Code break flag (Fixed to 0x00)
| └ Break address

Return frame

00 ---- Normal end
01, XX, 00, 00 ---- NG end
└ Command error type

```
static void proc_set_brk(void)
{
    UART_COMMAND cmd;
    uint8_t ReadBCR, BreakPointNum;
    uint32_t BreakAddress;
    uint16_t offset, BreakPointID;    /* point

    put_normal();
    offset = 0;
    BreakAddress = get_paramw(offset);
    offset += sizeof(uint32_t);
    if (get_paramb(offset))
    {
        set_errcode(PARAER);   /* break type !
        return ;
    }

    BreakPointNum = get_paramb(++offset);  /*
    switch (BreakPointNum)
    {
        case BP0:        /* break point 0 */
        BreakPointID = BDSU_BPR0;
        break;

        case BP1:        /* break point 1 */
        BreakPointID = BDSU_BPR1;
        break;
```

8.2.1.2.4.17 proc_del_brk

Delete hardware break point.

Command frame:
51, 00, XX, XX, XX, XX, XX, XX, XX
| | | | └ Setting Flag (Fixed to 0x00)
| | | └ break index (0x00 or 0x01)
| | └ Code break flag (Fixed to 0x00)
| └ Break address

Return frame
00 ---- Normal end
01, XX, 00, 00 ---- NG end
└ Command error type

```
static void proc_del_brk(void)
{
    UART_COMMAND cmd;
    uint8_t ReadBCR, BreakPointNum;
    uint16_t offset=0, BreakPointID;
    put_normal();
     /* point to break point type flag */
    offset += sizeof(uint32_t);
    if (get_paramb(offset))
    {
        set_errcode(PARAER);        /* break
        return ;
    }
    BreakPointNum = get_paramb(++offset);  /*
    switch (BreakPointNum)
    {
        case BP0:        /* break point 0 */
        BreakPointID = BDSU_BPR0;
        break;

        case BP1:        /* break point 1 */
        BreakPointID = BDSU_BPR1;
        break;

        default:
        set_errcode(PARAER);
        return;
    }
```

8.2.1.2.4.18 proc_set_rck_adr

Set RAM monitor address

Command frame:
A1, 28, XX, XX, XX, XX, …XX, XX, XX
| | | ∟ W/B + address (0~15) * 3 bytes
| | ∟address (2bytes)
| ∟W/B 00: read 1 byte; 01: read 1 word
∟monitor number (0~16 bytes)

Return frame:
00 ---- Normal end
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_set_rck_adr(void)
{
    uint16_t offset = 0;
    uint8_t cnt;
    put_normal();
    rck_length = get_paramb(offset);
    if(rck_length > RCK_MAX_SIZE)
    {
        set_errcode(PARAER);  /* the length exceed max size */
        rck_length = 0;
        return;
    }
    offset += sizeof(uint8_t);
    for(cnt=0; cnt<RCK_MAX_SIZE; cnt++)
    {
        rck_read_type[cnt] = get_paramb(offset); /* store data type */
        offset++;
        rck_adr[cnt] = get_paramh(offset); /* store data address */
        offset +=  sizeof(uint16_t);
    }
    return;
}
```

8.2.1.2.4.19 proc_read_rck_smdat

Set RAM monitor address

Command frame:
A2, 28

Return frame:
00, XX, XX, XX, XX,……XX, XX ---- Normal end
| | ∟ monitoring data (0~16)* 2 bytes
| ∟ Number (0~16 bytes)
∟ execute status. 00: executing; 80: break
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_read_rck_smdat(void)
{
    UART_COMMAND cmd;
    uint8_t mcu_state, ReadBCR, ClockSetState, cnt;
    uint16_t read_data, offset,wait_cnt = 0;
    if(rck_length == 0 || rck_length > RCK_MAX_SIZE)
    {
        rck_length = 0;
        offset = 0;
        put_normal();
        if((monitor_state & MCU_RUN) == MCU_RUN)
          mcu_state = EXECUTE_STATE;
        else
          mcu_state = BREAK_STATE;
        put_paramb(offset++, mcu_state);
                put_paramb(offset++, 0);
        put_frmlen(offset);
        return;
    }
```

8.2.1.2.4.20 proc_trim_cr

Get the CR trimming status, the trimming process should be done once target power is on.

Command frame:
E0, 24

Return frame
00, XX ---- Normal end
- ∟ Return status
  - 0x00: No need to trim
  - 0x01: need trimming, trim success
  - 0xFF: need trimming, trim failure

01, XX, 00, 00 ---- NG end
- ∟ Command error type

```
static void proc_trim_cr(void)/* not confirmed, just for mon
{
    uint32_t offset;
    offset = 0;
    put_normal();
    if ((monitor_state & CR_FINE) == CR_FINE)/* CR is ok*/
    {
        put_paramb(offset, CR_state);
        put_frmlen(++offset);
    }else{
        monitor_state |= TRIMMING;
        retry_counter =0;
        set_errcode(NOWTRIM);
    }
    return;
}
```

8.2.1.2.4.21 proc_user_cmd

Jump to RAM and execute RAM code. The specific RAM address is 0x0090.

Command frame:
E1, 24

Return frame
00 ---- normal end
01, XX, 00, 00 ---- NG end
└ Command error type

```
static void proc_user_cmd(void)
{
    UART_COMMAND cmd;
    uint8_t CommandType;
    uint16_t offset;
    uint16_t IxValue;

    offset = 0;
    put_normal();
    CommandType = get_paramb(offset);
    IxValue = get_paramh(++offset);
    switch (CommandType)
    {
        case CMD_TYPE_A:
        cmd.FC = BGMA_CMD_A;
        SET_UARTCMD_ADDR(cmd,IxValue);
        break;

        case CMD_TYPE_B:
        cmd.FC = BGMA_CMD_B;
        break;

        default:
        set_errcode(PARAER);
        return;
    }
    /* 0A 00 90 -------- jump ram command */
```

8.2.1.2.4.22 proc_load_env

Function:
Command to read system environment
Command frame:
F4, 00

Return frame
00, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX ---- Normal end
- HWID (0xFFFF)
- BID (0xFF)
- LDR_ID (0x0210)
- LVER (0x00)
- LLVL (0x01)
- LREL (0x02)
- CMN ID (0xFFFF)
- CVER (0xFF)
- CLVL (0xFF)
- CREL (0xFF)

01, XX ---- NG end
- Command error type

```
static void proc_load_env(void)
{
    UART_COMMAND cmd;
    uint8_t i;
    uint16_t offset = 0;

    put_normal();
    put_paramh(offset,0xFFFFU);
    offset += sizeof(uint16_t);
    put_paramb(offset++,0xFFU);
    put_paramh(offset,AID);
    offset += sizeof(uint16_t);
    cmd.FC = BGMA_PST;
    SET_UARTCMD_ADDR(cmd,MONITOR_INFO_ADDR);
    /* PST FC 00 -------- set ENV address */
    if(uart_communication(&cmd) != NORMAL)
    {
        set_errcode(cmd.RC);
        return ;
    }
    cmd.FC = BGMA_DRD;
    for(i=0;i<MONITOR_INFO_NUM;i++)
    {
        /* DRD xx xx -------- read ENV infomation */
        if(uart_communication(&cmd) != NORMAL)
        {
              set_errcode(cmd.RC);
              return ;
        }
```

8.2.1.2.4.23 proc_cng_adpt_mode

Function:
Configure Adapter ID. The new Adapter ID is 0x0220.

Command frame:
FE, 00, XX, XX
∟Adaptor ID

Return frame
00
01, XX, 00, 00 ---- NG end
∟ Command error type

```
static void proc_cng_adpt_mode(void)
{
    uint16_t offset= 0;
    put_normal();
    if (get_paramh(offset) != AID)      /* adapt ID != 0220 */
    {
        set_errcode(PARAER);
        return ;
    }
    return ;
}
```

8.2.1.2.4.24 proc_load_adpt_env

Load adaptor system environment
Command frame:
FF, 00

Return frame
00, XX, XX, XX, XX, XX, XX ---- Normal end
| | | | | └AREL
| | | | └ALVL
| | | └AVER
| | └ Adaptor ID
| └Number

Adaptor ID:

Hardware	Function	Adaptor ID
BDSU/EVA	Old BGM	0000H
	Enhancing BGM	0010H
BDSU/LPC	BGM	0210H
BDSU/18LPC	BGM	0220H
Mass production	FGM	8000H

01, XX, 00, 00 ---- NG end
└ Command error type

```
static void proc_load_adpt_env(void)
{
    uint16_t offset;

    offset = 0;
    put_normal();
    put_paramb(offset++,NUM);          /* NUM */
    put_paramh(offset,AID);             /* ADPT ID */
    offset += sizeof(uint16_t);
    put_paramb(offset++,VER);           /* ADPT version */
    put_paramb(offset++,LVL);           /* ADPT level */
    put_paramb(offset++,REL);           /* ADPT release */
    put_frmlen(offset);
    return ;
}
```

8.2.2 Common Data Structures

8.2.2.1 Macros

Following the Spec "*18um LPC MCU BGMA FW External Specification 1V6-20100622.doc*"

```
/* USB error code */

#define EC_USB_ NORCOD          0x00

#define EC_USB_ MCUBUSY         0x02

#define EC_USB_ NOWTRIM         0x20

…

/* UART error code */

#define EC_UART_ NORCOD          0x00

#define EC_UART_ COMERR          0x80

#define EC_UART_ ERSERR          0x85

…
```

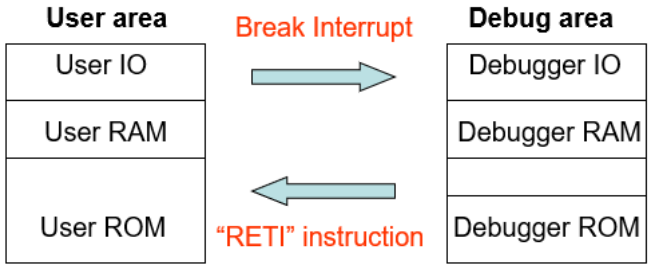
Other macros are defined as much as possible in the previous project.

8.2.2.2 Data Structures

```
typedef enum {CR_OK=0u, TRIM_OK=1u, TRIM_FAIL=0xFF,
CR_OK_ACK=0x51, TRIM_OK_ACK=0xA1} enCrState_t;
typedef struct SysStatus {
        uint8_t monitor_state;
        uint8_t power_state;
        enCRState_t encr_state;
        uint8_t version_num;
        uint8_t monitor_state;
        uint8_t release_num;
        …
} stcSysStatus;
```

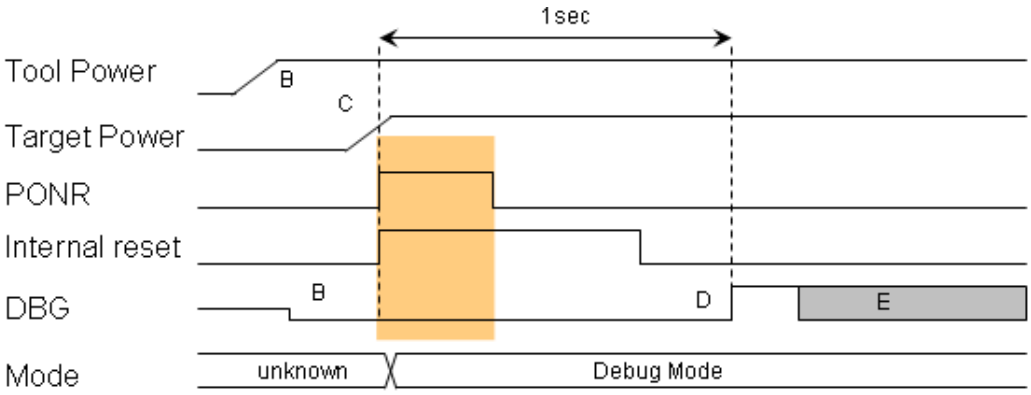
8.2.3 OCD Mode Management

The target OCD (On-Chip Debugger) MCU has two memory area, user area and debug area.



Correspondingly, it has two modes free-run mode and debug mode (consists of user mode and emulation mode).

When connecting the target MCU, adapter controls target MCU to enter to debug mode via the DBG pin.



8.2.4 LEDs Indication

This is two sets of RGB LEDs, but each set only uses Green and Red LEDs. The different combinations of lights represent the various states of the adapter.

Power LED, it controls by hardware directly.

LED	Status	Information
Green & Red	OFF	Both BGMA and target board power off
Green	ON	BGMA Power on only
Red	ON	Target board power on only
Orange	ON	Both BGMA and target board power on

Status LED:

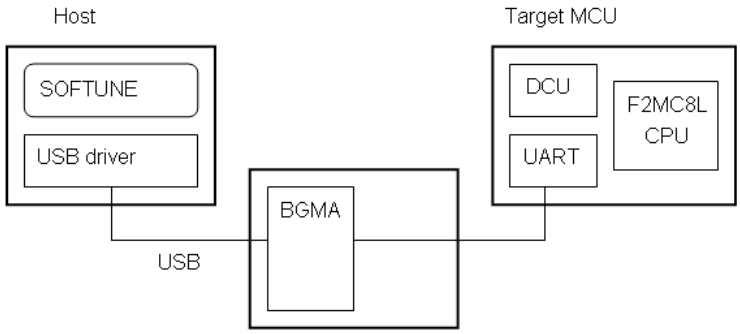
LED	Status	Information
Green & Red	OFF	Idle mode
Red	ON	Disconnect to target MCU
	Twinkling	
Green	ON	Connect to target MCU
	Twinkling	Operation
Orange	ON	

8.2.5 Protocol Analysis

After receiving the transmitted data from USB EP1, judge whether it is a valid command then do the corresponding processing (includes UART command send/receive, and GPIO pin configuration etc.)

8.2.6 USB Communication

USB interface use to communicate with PC host, the PC host contains SOFTUNE for target MCU debug and other software for BGMA Field Update and Standalone Programming. Note that this version will not support the BGMA field update function.



The features of USB I/F are as follows:

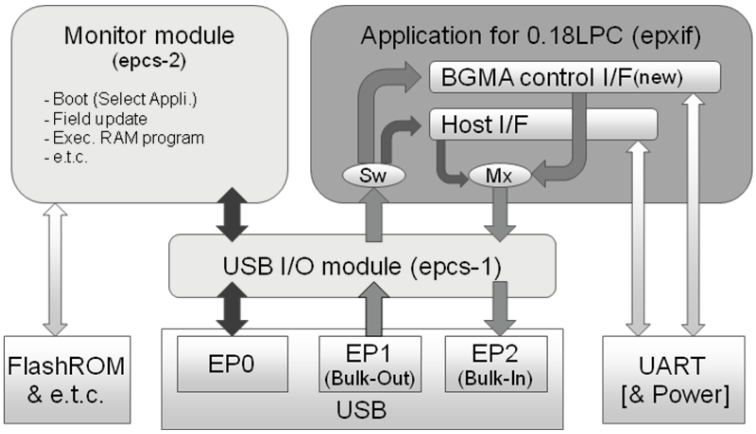
- Corresponding to Full Speed: BGMA is an USB device corresponding to the Full Speed(12Mbps) of USB 2.0 SPEC.
- Bus-powered: The USB I/F is a bus-powered device, whose power is supplied by USB bus.

- Vendor Define Class: No communication Device Class, Vendors can Define Device Class independently.
- Three Endpoints:

  EP0 Default (0x00: I/O): receive BGMA basic command such as Vendor command. Response to connection and configuration handling.

  EP1 (0x01 / OUT): Receive communication packet from Host PC.

  EP2 (0x82 / IN): send communication packet to Host PC.



8.2.6.1 USB Descriptor

The product string of BGMA is MB2146-07-E, the vendor ID is 0x04C5, the product ID is 0x2024.

```
/*------------ USB descriptor configuration-------------------*/
#define USB_VENDOR_ID       (0x04C5)   /* Vendor ID(fujitsu) */
#define USB_PRODUCT_ID      (0x2024)   /* Product ID */
#define USB_BCD_DEVICE      (0x0200)   /* USB Specification Release Number in Bi
nary-Coded Decimal */
```

8.2.6.1.1 Device Descriptor

```
///DEVICE DESCRIPTOR
0x12,    ///bLength: Length of this descriptor
0x01,    ///bDescriptorType: Device Descriptor Type
0x00,    ///bcdUSB: USB Version
0x02,    ///bcdUSB: USB Version
0xFF,    ///bDeviceClass: Class Code: VENDOR_SPECIFIC
0xFF,    ///bDeviceSubClass: Sub Class Code
0xFF,    ///bDeviceProtocol: Protocol Code
0x40,    ///bMaxPacketSize0: Maximum size of endpoint 0
LOBYTE(USB_VENDOR_ID),    ///idVendor: Vendor ID
```

```
HIBYTE(USB_VENDOR_ID),    ///idVendor: Vendor ID
LOBYTE(USB_PRODUCT_ID),    ///idProduct: Product ID
HIBYTE(USB_PRODUCT_ID),    ///idProduct: Product ID
LOBYTE(USB_BCD_DEVICE),    ///bcdDevice: Release Number
HIBYTE(USB_BCD_DEVICE),    ///bcdDevice: Release Number
0x01,    ///iManufacture: String-Index of Manufacture
0x02,    ///iProduct: String-Index of Product
0x03,    ///iSerialNumber: String-Index of Serial Number
0x01     ///bNumConfigurations: Number of possible configurations
```

8.2.6.1.2 Configuration Descriptor

```
///CONFIG DESCRIPTOR(1)
0x09,    ///bLength: Length of this descriptor
0x02,    ///bDescriptorType: Config Descriptor Type
32,      ///wTotalLength: Total Length with all interface- and endpoint
descriptors
0x00,    ///wTotalLength: Total Length with all interface- and endpoint
descriptors
0x01,    ///bNumInterfaces: Number of interfaces
0x01,    ///iConfigurationValue: Number of this configuration
0x04,    ///iConfiguration: String index of this configuration
0x80,    ///bmAttributes: Remote-Wakeup not supported
0xFA,    ///MaxPower: (in 2mA)
    ///INTERFACE DESCRIPTOR(0)
    0x09,    ///bLength: Length of this descriptor
    0x04,    ///bDescriptorType: Interface Descriptor Type
    0x00,    ///bInterfaceNumber: Interface Number
    0x00,    ///bAlternateSetting: Alternate setting for this interface
    0x02,    ///bNumEndpoints: Number of endpoints in this interface excluding
endpoint 0
    0xFF,    ///iInterfaceClass: Class Code: VENDOR_SPECIFIC
    0xFF,    ///iInterfaceSubClass: SubClass Code
    0xFF,    ///bInterfaceProtocol: Protocol Code
    0x00,    ///iInterface: String index
        ///ENDPOINT DESCRIPTOR(1)
        0x07,    ///bLength: Length of this descriptor
        0x05,    ///bDescriptorType: Endpoint Descriptor Type
        0x82,    ///bEndpointAddress: Endpoint address (IN,EP2)
        0x02,    ///bmAttributes: Transfer Type: BULK_TRANSFER
        0x40,    ///wMaxPacketSize: Endpoint Size
        0x00,    ///wMaxPacketSize: Endpoint Size
        0x00,    ///bIntervall: Polling Intervall
        ///ENDPOINT DESCRIPTOR(0)
        0x07,    ///bLength: Length of this descriptor
```

```
0x05,    ///bDescriptorType: Endpoint Descriptor Type
0x01,    ///bEndpointAddress: Endpoint address (OUT,EP1)
0x02,    ///bmAttributes: Transfer Type: BULK_TRANSFER
0x40,    ///wMaxPacketSize: Endpoint Size
0x00,    ///wMaxPacketSize: Endpoint Size
0x00     ///bIntervall: Polling Intervall
```

8.2.6.2 Endpoint Details

The BGMA Endpoint has the following function listed below:

Endpoint No.	Functions	Remarks
0 (Default)	The standard requirements and the vendor requirements are accepted as default pipe.	Control I/O Max. 64Bytes x 4
1 (0x01)	Receive Max.1024 Bytes of Epx I/F Data Packet (Include ID cod) and send to Application.	Bulk OUT Max. (1024 + 64) Bytes
2 (0x82)	Send Max.1024 Byte of Epx I/F Data Packet (Include ID code) to host.	Bulk IN Max. (1024 + 64) Bytes

Note:

The Endian of Data takes 8bit as the basic unit: According to the target SPEC, it's difficult to convert the endian of communication Data. Therefore, data is basically processed in units of 8 bits. In other words, according to the target SPEC, it is necessary to adjust the sending and receiving data on the Host side.

Data on USB is little Endian: The data stream passing on the USB is based on little Endian. Therefore, when standard requirements and Vendor command use data of more than 16bit, it becomes little Endian.

8.2.6.2.1 Default Endpoint (EP0)

Accept Standard Requests and Vendor Requests. Control transmission.

The requirements for setting BGMA are as follows. For details, please refer to "[MB2146-07] EPCS_IF_Specification_for_BGMA-07" document.

The following is Vendor Requests:

No.	bRequest	Description	Remarks
0x00	VSET_BMGA_PTR	Set the address of ROM, RAM and I/O in BGMA.	with OUT data (32bit)
0x01	VGET_BGMA_DATA	Returns the values of ROM, RAM, and I/O in the specified BGMA. The data number is specified on wLength.	with IN data (Max64Byte)
0x02	VSET_BGMA_DATA	Set the specified value on the RAM	with OUT data

		and I / O in the specified BGMA. The data number is specified on wLength.	(Max64Byte)
0x03	VSET_BGMA_FER	Erases the Flash ROM.	with OUT data
0x04	VSET_BGMA_FWR	Write Flash ROM.	with OUT data
0x10	[VSET_LED]	The LED on the BGMA turns on / off.	only wValue
0x11	VGET_BGMA_INFO	Returns the version information of BGMA.	with OUT data (Max64Byte)
0x12	VSET_BGMA_RESET	BGMA resets.	only wValue
0x13	VGET_BGMA_CALL	Execute RAM code on BGMA and return the result.	with OUT data

8.2.6.2.2 Endpoint 1,2

With 64Bytes as the unit, the bulk transmission of (1024 + 64) Bytes is maximum. OUT at Endpoint1, IN at Endpoint2.

Corresponding to Data Packets above 64 Bytes, short Packet (Packet less than 64 Bytes) is detected at the end of Packet. (When the Data Packet is an integer multiple of 64 Bytes, it is necessary for the sending side to continuously send 0Byte of Data.)

Application uses the API of the epcs-1 module to obtain the packet from EndPoint1. Use the same API to send Packet to EndPoint2. For short packets that cannot be detected, the epcs-1 module cannot distinguish data. Therefore, no matter what kind of Packet format, there is no problem.

The USB data frame form:

Offset +0[Byte]	+1	+2	+3	+4	...	+N
Header (0xBA)	ID (0x00~0xFF)	Packet Length-L(*) (0x00~0xFF)	Packet Length-H(*) (0x00~0x3F)	Packet Data[0]	...	Packet Data[N-4]

Big endian format is used, so if data 1234H is filled to USB command, the data should be 00, 00, 12, 34.

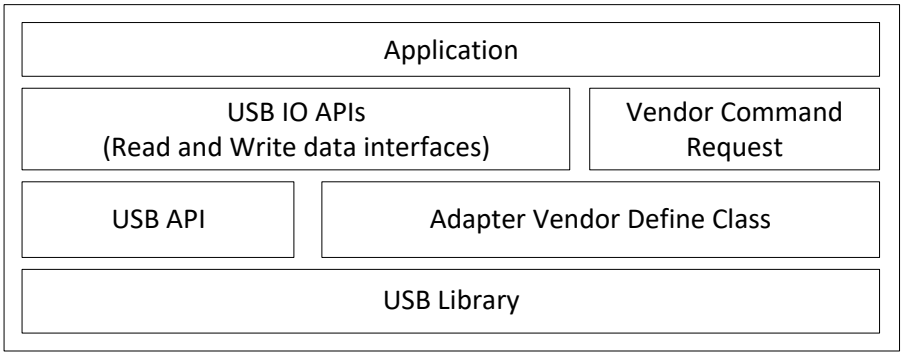
The maximum size of USB frame is 512 bytes for fitting the buffer size.

ID description:

ID	Description
0x00	Packet Data is HostI/F
0x01	Packet Data is UART control command.
0x02 .. 0xFF	(Reserved)

8.2.6.3 Architecture

The USB communication includes the USB Library, BGMA vendor request, data receiving and sending APIs, and application. The Main Application is executing the user defined program. The USB IO APIs are the interfaces between the USB Library and the Main Application. Only a simple USB API is the direct interface between Main Application and USB Library, which handles USB initialization and getting the current status of the USB function.



USB Library: The USB function library handles all configuration and data transfers with the hardware abstraction layer. A small USB API is provided to the main application to open or close the USB features. The vendor class handles the main communication.

USB IO APIs: All initiated USB transfers or received USB transfers can be accessed from the main application via the vendor class API.

This USB interface is vendor define device class, although we can use USB Wizard tool to generate USB related code, but this tool does not support to generate the vendor define class, so we need to implement this device class in the new 8Fx adapter project.

8.2.6.3.1 Vendor Class Initialization

The class initialization function need to set the interface class, sub class and protocol to 0xFF, and vendor request callback function in the USB config struct, then call the UsbDevice_RegisterVendorClass() function to register this class. If the configuration descriptor includes this class interface, register this class to USB device internal, and set the class OUT/IN endpoint callback function.

```
void UsbDeviceVendor_Init(stc_usbn_t* pstcUsb)
{
    stc_usbdevice_class_config_t stcUsbClassConfig;
    stc_usbdevice_endpoint_config_t stcEndpointConfig;
    uint8_t* pu8Interface = NULL;
    uint8_t u8InterfaceLength = 0;

    pstcUsbHandle = pstcUsb;
    stcUsbClassConfig.u8InterfaceNumber         = 0xFF; // Do not use fix interface number, choose by class, subclas
    stcUsbClassConfig.u8InterfaceClass          = 0xFF;
    stcUsbClassConfig.u8InterfaceSubClass       = 0xFF;
    stcUsbClassConfig.u8InterfaceProtocoll      = 0xFF;
    stcUsbClassConfig.pfnSetupRequestCallback   = NULL; // No setup request callback handled
    stcUsbClassConfig.pfnConnectionCallback     = NULL; // No connection callback handled
    stcUsbClassConfig.pfnDisconnectionCallback  = NULL; // No disconnection callback handled
    stcUsbClassConfig.pfnConfCallback           = ConfCallback; // Callback for configuration set
    stcUsbClassConfig.pfnVendorRequestCallback  = VendorRequestCallback; // Callback for vendor request
    UsbDevice_RegisterVendorClass(pstcUsb, &stcUsbClassConfig, &pu8Interface, &u8InterfaceLength);

    for(uint8_t i = 0;i < u8InterfaceLength;) {
        if (USBDESCR_ENDPOINT == pu8Interface[i + 1]) {
            stcEndpointConfig.u8EndpointAddress = pu8Interface[i + 2];
            // OUT endpoint
            if(((stcEndpointConfig.u8EndpointAddress) & 0x80) == 0) {
                stcEndpointConfig.pfnRxTxCallback = OutEpCallback;
                stcEndpointConfig.pstcEndpointBuffer = &stcEndpointBufferOUT;
                pstcEndpointOUT = UsbDevice_SetupEndpoint(pstcUsb, &stcEndpointConfig);
            } else { // IN endpoint
                stcEndpointConfig.pfnRxTxCallback = InEpCallback;
                stcEndpointConfig.pstcEndpointBuffer = &stcEndpointBufferIN;
                pstcEndpointIN = UsbDevice_SetupEndpoint(pstcUsb, &stcEndpointConfig);
                pstcEndpointIN->bAutomaticNullTermination = TRUE;
            }
        }
        i += pu8Interface[i];
    }
}
```

The ConfCallback() callback use to see if configuration is set or cleared. When the USB SET_CONFIGURATION request received and set the vendor class flag to true.

8.2.6.3.2 Vendor Request

The host PC software will use this vendor request to get adapter version and adapter information, and other functions. In the USB initialization, use the following function to set the vendor request callback in the UsbDeviceVendor.c file:

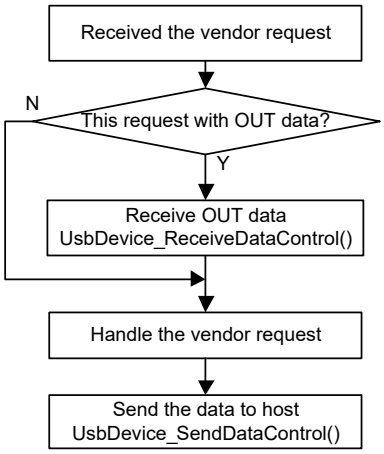
```
void UsbDeviceVendor_SetVendorRequestCallback(usbdevice_setuprequest_func_ptr_t cb)
{
    pfnVendorRequestCallback = cb;
}
```

In the vendor class initialization function, set the vendor request callback to VendorRequestCallback(), when the USB receive the vendor request, the library will call this callback function.

```
static void VendorRequestCallback(stc_usbn_t* pstcUsb, stc_usb_request_t* pstcSetup)
{
    if(NULL != pfnVendorRequestCallback) {
        pfnVendorRequestCallback(pstcUsb, pstcSetup);
    }
}
```

When the vendor request need to send or received data, use the UsbDevice_SendDataControl() or UsbDevice_ReceiveDataControl() function to handle request data.

When receive the vendor command request, check the command type, if the command needs to receive data and initiate endpoint buffer to receive vendor command data, then handle this vendor request and return data to host.



8.2.6.3.3 Data Receiving Implementation

After the USB initialization, call the following function to set OUT endpoint receiving buffer and enable endpoint interrupt to receive data, meanwhile this function will set the data receiving complete callback.

```
en_result_t UsbDeviceVendor_ReceiveData(uint8_t* pu8Buf, uint32_t u32Len, trans_com_func_ptr_t cb)
{
    if(bVendorReady == FALSE) {
        return Error;
    }
    pstnEndpointOutTransComFunc = cb;
    return UsbDevice_ReceiveData(pstcUsbHandle, pstcEndpointOUT, pu8Buf, u32Len, UsbIRQ);
}
```

When the data transfer complete, the data receiving complete callback function will be called in the OUT-endpoint data transferred callback of vendor class.

```
static void OutEpCallback(stc_usbn_t* pstcUsb, struct stc_usbn_endpoint_data* pstcEndpoint)
{
    if((NULL == pstcUsb) || (NULL == pstcEndpoint)) {
        return;
    }
    if(NULL != pstnEndpointOutTransComFunc) {
        pstnEndpointOutTransComFunc(pstcUsb, pstcEndpoint->pstcEndpointBuffer->pu8Buffer, pstcEndpoint->pstcEndpointBuffe
        pstnEndpointOutTransComFunc = NULL;
    }
}
```

8.2.6.3.4 Data Sending Implementation

The device need to send data to host PC, use the UsbDevice_SendData() function to initiate sending data by IN endpoint, the vendor class implement the following function to send data and set the data sending complete callback:

```
en_result_t UsbDeviceVendor_SendData(uint8_t* pu8Buf, uint32_t u32Len, trans_com_func_ptr_t cb)
{
    if(bVendorReady == FALSE) {
        return Error;
    }
    pstnEndpointInTransComFunc = cb;
    return UsbDevice_SendData(pstcUsbHandle, pstcEndpointIN, pu8Buf, u32Len, UsbIRQ);
}
```

When the data sending complete, the transfer complete callback will be called in the IN-endpoint data transferred callback of vendor class.

```
static void InEpCallback(stc_usbn_t* pstcUsb, struct stc_usbn_endpoint_data* pstcEndpoint)
{
    if((NULL == pstcUsb) || (NULL == pstcEndpoint)) {
        return;
    }
    if(NULL != pstnEndpointInTransComFunc) {
        pstnEndpointInTransComFunc(pstcUsb, pstcEndpoint->pstcEndpointBuffer->pu8Buffer, pstcEndpoint->pstcEndpointBuffer
        pstnEndpointInTransComFunc = NULL;
    }
}
```

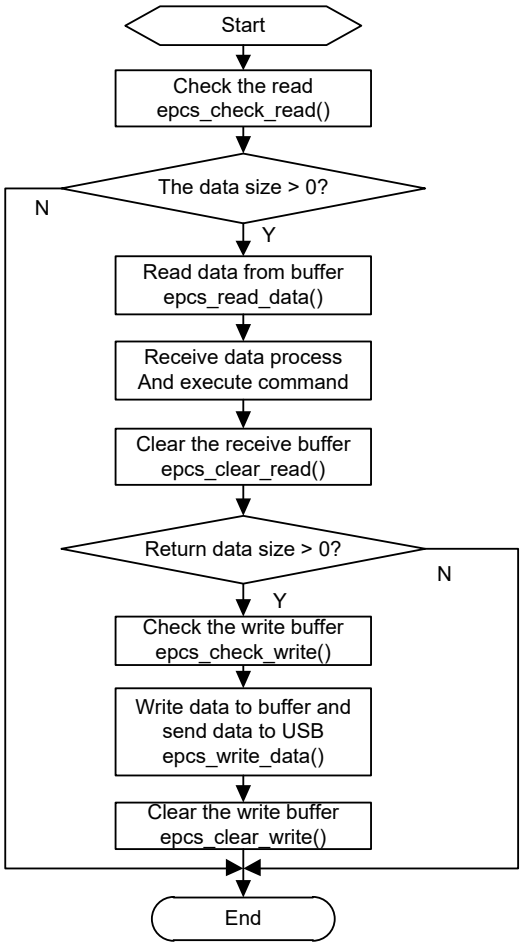
8.2.6.3.5 USB IO APIs

The USB IO APIs use to receive data from host and send the application data to host. The USB IO APIs function:

Function	Description
epcs_initialize	Initialize USB device UsbConfig_UsbInit() UsbDeviceVendor_SetVendorRequestCallback()
epcs_check_read	Check if USB data is received
epcs_read_data	Read received data
epcs_clear_read	Clear received data and initiate the receiving data UsbDeviceVendor_ReceiveData()
epcs_check_write	Check the send buffer status

epcs_write_data	Write send data to buffer and request the USB to send data UsbDeviceVendor_SendData()
epcs_clear_write	Clear the send data buffer and status

The following is the USB data receiving and sending flow: the task checks the receive buffer, if received data from host, and verify the command format and then handle the HOSTIF command, after command handle completely and return the command result to host:



8.2.7 UART Communication

8.2.7.1 Overview

The system sends command to target MCU when it receives USB data, and the UART handle mode includes A, B, C, D, E.

8.2.7.2 Internal CR Clock Trimming

The trimming process is the same as synchronize process for BGMA; BGMA sends “0x55AA” at 62500bps continuously and wait for ACK. Target MCU tunes its internal CR clock frequency by modifying the CR tuning register after it receives an error data from 1-line UART. This

process continues until target MCU receives “0x55AA” correctly. Then target MCU returns ACK (0xA1) to BGMA.
By receiving ACK 0xA1, BGMA finishes the internal CR clock trimming process.
If BGMA cannot receive ACK 0x51 or 0xA1 after “0x55AA” sent over 2000 times, it means the internal CR trimming failure.
Please refer “0.18um LPC OCD System External spec. PDF” chapter 7.2 for more information.

8.2.7.3 Protocol

There are Type A with the output parameter, Type B which includes the input parameter, Type C for the GO command and Type D for multi-RW.
The input output parameter is expressed by 16bit by two bytes of H/L.
Identify by MSB of FC.

- MSB:0 Type A
- MSB:1 Type B

SUM is calculated with FC+H+L=SUM (Type A, B and C).
SUM is calculated with FC+D0+..+D15=SUM (Type D).
The sequence of FC-H-L-SUM-RC of the target MCU is reset by making an error of the communication of data from the adaptor or monitor.
The packet sequence can be intentionally reset by transmitting the BRK signal from the adaptor or monitor.

Type A

Adaptor	FC	H	L	SUM	
Direction	↓	↓	↓	↓	↑
Monitor					RC

Type B

Adaptor	FC				
Direction	↓	↑	↑	↑	↑
Monitor		H	L	SUM	RC

Type C

Adaptor	FC	H	L	SUM		
Direction	↓	↓	↓	↓	↑	↑
Monitor					RC	RC2

Type D

Adaptor	FC	D0	…	D15	SUM	
Direction	↓	↓	↓	↓	↓	↑
Monitor						RC

Or

Adaptor	FC					
Direction	↓	↑	↑	↑	↑	↑
Monitor		D0	…	D15	SUM	RC

8.2.7.3.1 1-Line UART Command Error Type

RC	Symbol	Description
00	NORCOD	Normal termination
80	COMERR	FC type is not supported
85	ERSERR	Fails in the memory deletion.
86	WRTERR	Fails in the memory writing.
C0	TARERR	Check sum is not correct
FD	SECERR	The command cannot be executed because security error/is protected.

8.2.7.3.2 1-Line UART Command List

FC	Type	Symbol	Description	Sec	Parameter Type	Parameter introduction
00	A	PST	Parameter set	○	In	I: Parameter
81	B	RD	Read memory	×	P+1	P: adrs O:read data
91	B	RD_2	Read memory by word	×	P+2	P: adrs O:read data
B1	D	RD16	Read 16byte	×	P+16	P: adrs O:read data
82	B	DRD	Dsu read	×	P+1	P: adrs O:read data
03	A	WR	Write memory	×	P+1	P: adrs I:write data
05	A	DWR	Dsu write	○	P+1	P: adrs I:write data
06	A	FER	Flash erase	×	Broken	P: sector I:Type
07	A	FWR	Flash write	×	P+1	P: adrs I:write data
17	A	FWR_2	Flash write by word	×	P+2	P: adrs I:write data
27	D	FWR16	Write 16bytes Flash	×	P+16	P: adrs I:write data
08	A	RST	user reset	×	Broken	N/A
09	A	GO	user execute	×	Broken	N/A
0A	A&B	CMD	Command execute	×		

8.2.7.3.3 1-Line UART Command Specification

8.2.7.3.4 PST

The input data is set in the parameter register.
The set parameter register is used as address by other commands.
00, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ Address L
└ Address H

8.2.7.3.5 RD

The memory of the user space is led by the byte.
The address is a parameter register set by the PST command.
As for the parameter register, +1 is done when succeeding in reading.
81, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ Data L
└ Data H

8.2.7.3.6 RD2

The memory of the user space is led by the word.
The address is a parameter register set by the PST command.
As for the parameter register, +2 is done when succeeding in reading.
91, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ Data L
└ Data H

8.2.7.3.7 RD16

The memory of the user space is led by 16 bytes.
The address is a parameter register set by the PST command.
As for the parameter register, +16 is done when succeeding in reading.
B1, XX, XX, ….XX, XX, XX
| | └ RC. 00: normal; others: ERROR
| └ Check sum
└ Data 0~15

8.2.7.3.8 DRD

The memory of the emulation space is led by the byte.
The address is a parameter register set by the PST command.
As for the parameter register, +1 is done when succeeding in reading.
82, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ Data L
└ Data H

8.2.7.3.9 WR

The memory of the user space is written.
The address is a parameter register set by the PST command.
As for the parameter register, +1 is done when succeeding in writing.
03, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ DATA L
└ DATA H

8.2.7.3.10 DWR

The memory of the emulation space is written.
The address is a parameter set by the PST command.
As for the parameter, +1 is done when succeeding in writing.
05, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ DATA L
└ DATA H

8.2.7.3.11 FER

Chip erase is implemented if Address = 0; and Sector erase is implemented if Address = sector start address
06, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ Address L
└ Address H

8.2.7.3.12 FWR

Write 1 byte data into a built-in flash memory.
The address is a parameter register set by the PST command.
As for the parameter register, +1 is done when succeeding in writing.
07, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ Data L
└ Data H

8.2.7.3.13 FWR2

Write 1 word data into a built-in flash memory.
The address is a parameter register set by the PST command.
As for the parameter register, +2 is done when succeeding in writing.
07, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ Data L
└ Data H

8.2.7.3.14 FWR16

Write 16 bytes data into a built-in flash memory.
The address is a parameter register set by the PST command.
As for the parameter register, +16 is done when succeeding in writing.
07, XX, … XX, XX, XX
| | └ RC. 00: normal; others: ERROR
| └ Check sum
└ Data 0~15

8.2.7.3.15 RST

Reset is specified.
The parameter register does not change.
08, XX, XX, XX, XX
| | | └ RC. 00: normal; others: ERROR
| | └ Check sum
| └ No use
└ No use

8.2.7.3.16 GO

Monitor code changes to the user mode.
When the break is done after a usual packet is issued and Monitor code moves to the emulation mode, the break notification data is sent. The parameter register becomes irregular.
The compulsion break is done by issuing the break signal to DBG.

09, XX, XX, XX, XX, XX
- XX: RC2. 01: normal; others: ERROR
- XX: RC1. 00: normal; others: ERROR
- XX: Check sum
- XX: No use
- XX: No use

Monitor program returns RC1 to indicate the GO command is received; target MCU go to user program after RC1. RC2 means target MCU quits from user program and enters into monitor program.

8.2.7.3.17 CMD

The program on RAM is executed.
When this command is issued, 0x0090 RAM is called.
It is necessary to write the program in the WR command since 0x0090. Be the instruction which uses the general register and the stack cannot used and careful. The program ends by the RET instruction.
As for the value of the parameter register, when called by the Type A protocol, the input parameter is set in EP and read to IX. The value of A register when ending is returned as RC, and the value of the IX register is set in the parameter register.
When read by the Type B protocol, the value of the parameter register is set in IX and called. The content of the EP register when ending is returned as an output parameter, and the value of A register is returned as RC. The value of the IX register is set in the parameter register.

0A, XX, XX, XX, XX
- XX: RC. 00: normal; others: ERROR
- XX: Check sum
- XX: No use
- XX: No use

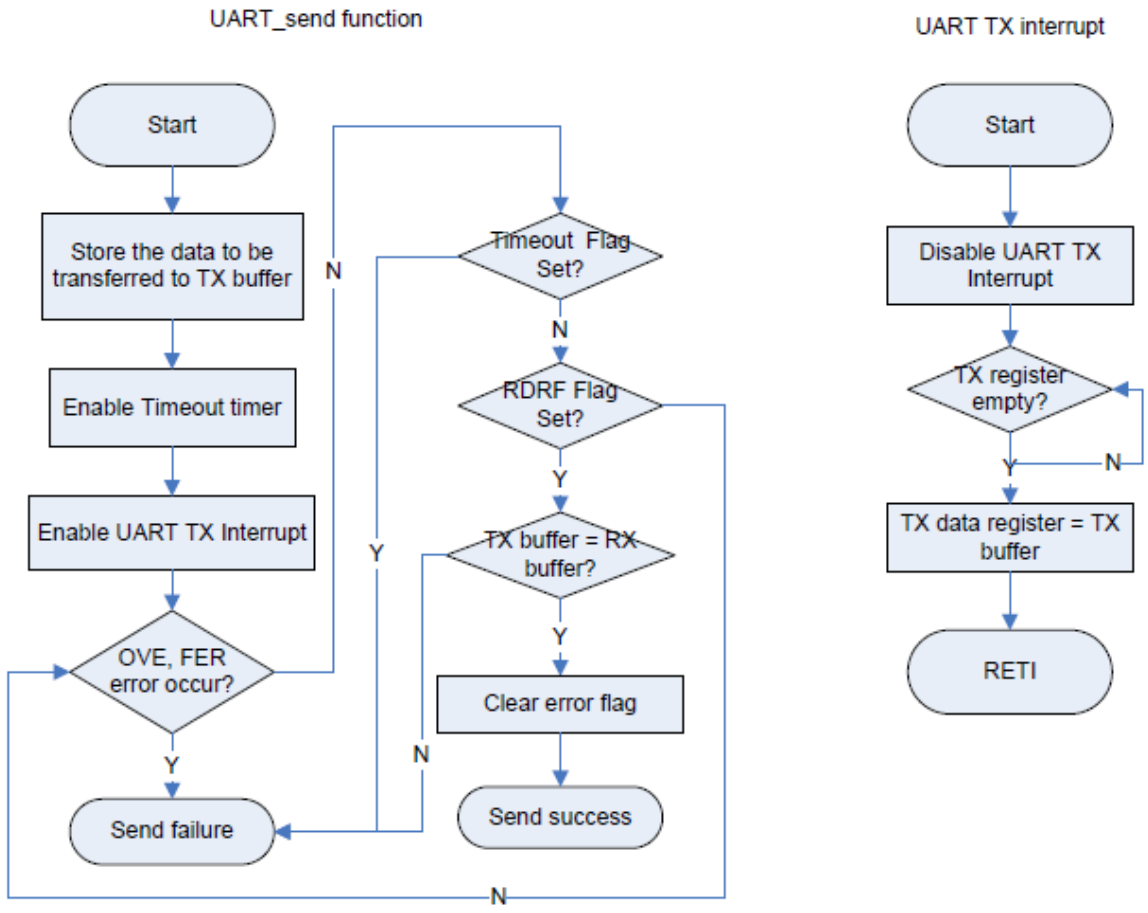
8A, XX, XX, XX, XX
- XX: RC. 00: normal; others: ERROR
- XX: Check sum
- XX: Data L
- XX: Data H

8.2.7.3.18 Flowchart

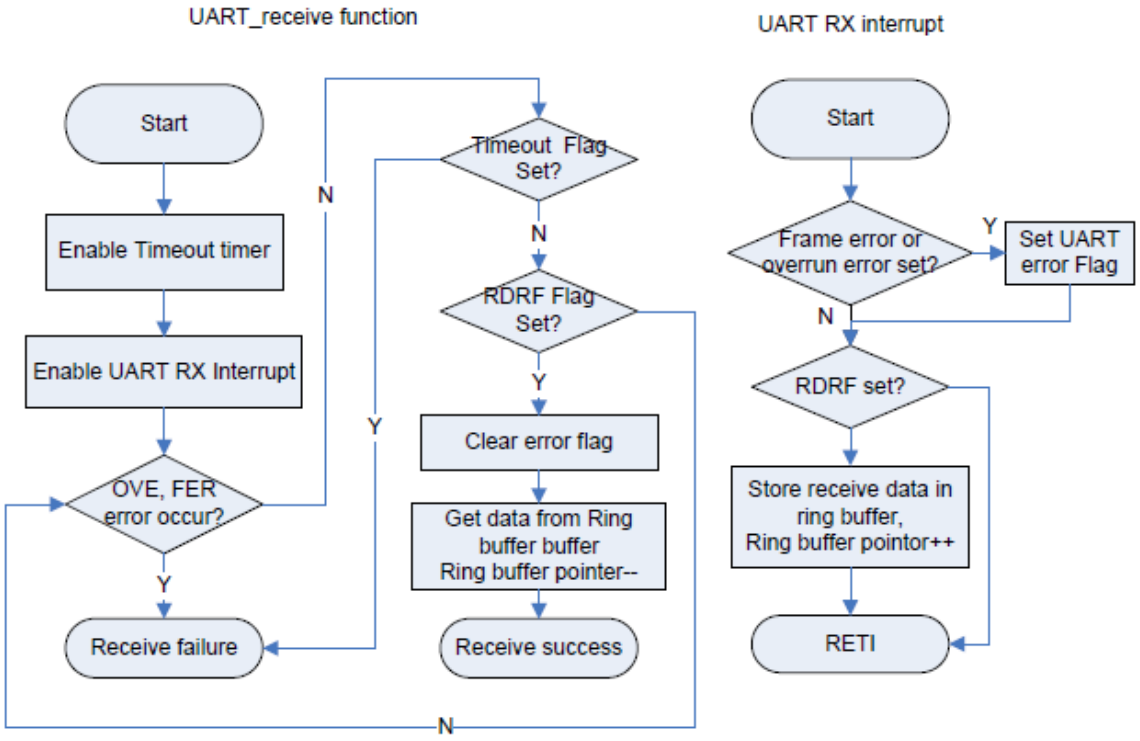
Communication process:

- UART initialize setting
- Fill send data in UART OUT buffer
- Check receive flag (if OK)
- Read receive buffer (if same as send data)
- Data has been sent by BGM adaptor
- Prepare receive data from LPC MCU
- Check receive flag
- Read receive buffer (this will be the data sent from monitor)
- Data has been received by BGM adaptor

The UART send byte flow is shown as below.

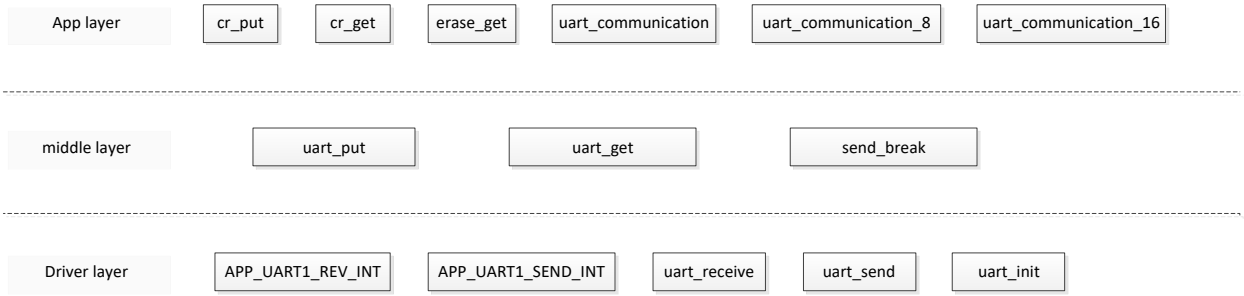


The UART get byte flow is shown as below.



If UART send/receive fail, the debug pin will be pull down for 683us to reset UART communication.

8.2.7.3.19 Architecture



8.2.7.3.19.1 Data introduction:

The UART communication data structure includes UART_COMMAND, UART_COMMAND_16 and UART_COMMAND_8.

```
typedef struct __uart_command__ {   /* UART communication data structure */
    uint8_t FC;                 /* command type */
    uint8_t AddrH;              /* higher 8bits address */
    uint8_t AddrL;              /* lower 8bits address */
    uint8_t RC;                 /* response command from target */
} UART_COMMAND;

typedef struct __uart_command_16__ {   /* UART communication data structure */
    uint8_t FC;                 /* command type */
    uint8_t Data0;
    uint8_t Data1;
    uint8_t Data2;
    uint8_t Data3;
    uint8_t Data4;
    uint8_t Data5;
    uint8_t Data6;
    uint8_t Data7;
    uint8_t Data8;
    uint8_t Data9;
    uint8_t Data10;
    uint8_t Data11;
    uint8_t Data12;
    uint8_t Data13;
    uint8_t Data14;
    uint8_t Data15;
    uint8_t RC;                 /* response command from target */
} UART_COMMAND_16;
typedef struct __uart_command_8__ {   /* UART communication data structure */
    uint8_t FC;                 /* command type */
    uint8_t Data0;
    uint8_t Data1;
    uint8_t Data2;
    uint8_t Data3;
    uint8_t Data4;
    uint8_t Data5;
    uint8_t Data6;
    uint8_t Data7;
    uint8_t RC;                 /* response command from target */
} UART_COMMAND_8;
```

Function introduction:

8.2.7.3.19.2 App layer:

- CR_put:

This function is that BGM Adapter send 0x55AA to target board MCU for CR trimming.

```
void cr_put(UART_COMMAND* cmd, uint32_t timer)
{
    p_wr_buf = p_rd_buf = &rev_buf[0];
    UART_DBG_CH->SCR_f.RIE = 1;/* enable RX enable */
    SET_PUARTCMD_ADDR(cmd,CR_TRIM_CMD);
    uart_send(&(cmd->AddrH), UART_SEND_TIMEOUT_TIMER);
    waitus(timer);
    uart_send(&(cmd->AddrL), UART_SEND_TIMEOUT_TIMER);
}
```

- CR_get:

This function is that BGM Adapter get CR trimming result of target board MCU. If the value is 0x51 or 0xA1, the result is pass.

```
uint8_t cr_get(void* DataPtr)
{
    uint8_t ret;
    ret = cr_receive((uint8_t*)DataPtr);
    UART_DBG_CH->SCR_f.RIE = 0;/* disable
    return ret;
}
```

- Erase_get:

This function is that BGM Adapter get target MCU state after target board MCU execute Erase operation.

```
uint8_t erase_get(void* DataPtr)
{
    uint8_t ret;
    ret = erase_receive((uint8_t*)
    UART_DBG_CH->SCR_f.RIE = 0;/*
    return ret;
}
```

- Uart_communication:

This function is that BGM Adapter communicate with target board MCU, it includes data communication, resend handle, timeout handle and communication break handle. BGM Adapter will send 4 bytes data and receive one byte data in this function, BGM Adapter open BT timer for waiting for get one byte data after BGM Adapter send 4 bytes data. If receive data is time out, BGM Adapter will stop communication and resend the send data. If resending counter is over, BGM Adapter will stop communication and record time out state.

it includes two command types (A and B) as below.

Type A: send FC, AddrH, AddrL and SUM, receiving RC

Type B: send FC, receiving AddrH, AddrL, SUM and RC

```
uint8_t uart_communication(UART_COMMAND *cmd)
{
    uint32_t cnt;
    cnt = CMD_RETRY;    /* retry times */
    cmd->RC = NORMAL;
    if (!(cmd->FC & 0x80))
    {
        /* type A */
        while (1)
        {
    }
    else  /* type B */
    {
    return cmd->RC;    /* 'cmd->RC' indicate the co
}
```

- uart_communication_16:

This function is that BGM Adapter communicate with target board MCU, it includes data communication, resend handle, timeout handle and communication break handle. BGM Adapter will send 1 byte data and receive 18 bytes data in this function, BGM Adapter open BT timer for waiting for get one byte data after BGM Adapter send 1 byte data. If receive data is time out, BGM Adapter will stop communication and resend the send data. If resending counter is over, BGM Adapter will stop communication and record time out state.

its command type is D1 as below.

Type D: send FC, receiving Data0.....Data15, SUM and RC

```
uint8_t uart_communication_16(UART_COMMAND_16 *cmd16)
{
    uint32_t cnt;
    cnt = CMD_RETRY;    /* retry times */
    cmd16->RC = NORMAL;
    if ((cmd16->FC & 0x20))  /* type monitor 2-18 whi
    {
        /* type D */
        if ((cmd16->FC & 0x80))
        {
    }
    return cmd16->RC;    /* 'cmd->RC' indicate the com
}
```

- uart_communication_8:

This function is that BGM Adapter communicate with target board MCU, it includes data communication, resend handle, timeout handle and communication break handle. BGM Adapter will send 10 bytes data and receive 1 byte data in this function, BGM Adapter open BT timer for waiting for get one byte data after BGM Adapter send 10 bytes data. If receive data is time out, BGM Adapter will stop communication and resend the send data. If resending counter is over, BGM Adapter will stop communication and record time out state.

Its command type is E as below.

Type E: send FC, Data0.....Data7 and SUM, receiving RC

```
uint8_t uart_communication_8(UART_COMMAND_8 *cmd8)
{
    uint32_t cnt;
    cnt = CMD_RETRY;    /* retry times */
    cmd8->RC = NORMAL;
    if ((cmd8->FC & 0x20))  /* tyge monitor 2-18 w
    {
        /* type E */
        if (!(cmd8->FC & 0x80))
        {
    }
    /* ohma added */
    return ERROR;
}
```

8.2.7.3.19.3 middle layer:

- uart_put:

This function is that BGM Adapter send data to target board MCU, it includes send data and set check sum functions,

```
uint8_t uart_put(uint8_t *cmd, uint32_t timer, uint8_t put_cnt)
{
    uint8_t i;
    uint8_t sum = 0;
    uint8_t ret = 0;
    Secu16BytesRec = 0;/*use for 16bytes security error*/

    if(put_cnt == 1)
    {
    else if(put_cnt == 4)
    {
    else/*8 bytes*/
    {

}
```

- uart_get:

This function is that BGM Adapter receive data from target board MCU, it includes receive data and check sum verification functions,

```
uint8_t uart_get(uint8_t *cmd, uint32_t timer, uint8_t get_cnt)
{
    uint8_t fc,sum,ret=0,i;
    uint8_t tempDataPtr[18];
    wait60us(timer);      /* delay 50us*timer for target operati

    if(ret = uart_receive(&tempDataPtr[0], UART_REV_TIMEOUT_TIME
    {
    UART_DBG_CH->SCR_f.RIE = 0;/* disable RX enable */
    if(get_cnt == 1)
    {
    else if(get_cnt == 4)
    {
    else if(get_cnt == 18)  /* type D*/
    {
    return ret;
}
```

- send_break:

This function is that BMGA send break communication to target board MCU, when some error occurred such as check sum error, time out, the BGM Adapter MCU ask target board to reset UART data, and it will resend data to target board MCU.

In this function, UART port will be set to GPIO type, set it to low level. Set it to high level after time out. When the 'break' info has sent finish. Switch the port to UART mode and set UART baud to 62500.

```
void sendbrk(uint32_t time_out)
{
    UART_DBG_CH->SCR_f.TXE = 0;/* Disable send */
    UART_DBG_CH->SCR_f.RXE = 0;/* Disable Receive */
    UART_DBG_CH->SCR_f.RIE = 0;/* disable RX enable */
    bt0_enable(1, time_out);

    /* Shift UART to GPIO */
    MFS_ChangeUARTPinToGPIO();
    /* Pull L SO pin */
    Gpiolpin_Put( GPIO1PIN_P12, 0u);

    while(!check_bt0_overflow());

    /* Pull H SO pin */
    Gpiolpin_Put( GPIO1PIN_P12, 1u);
   // PORT1_DDR0_DDR04 = 1;
    /* Shift GPIO to UART */
    SetPinFunc_SIN1_1();
    SetPinFunc_SOT1_1();

    if((version_num == VER_35)||((version_num == VER_1_18) && (level_num == 1)))
    {

    UART_DBG_CH->SCR_f.TXE = 1;/* enable send */
    UART_DBG_CH->SCR_f.RXE = 1;/* enable Receive */
    UART_DBG_CH->SCR_f.RIE = 1;/* enable RX enable */

    return;
}
```

8.2.7.3.19.4 Driver layer:

- MFS1_RX_TX_IRQHandler:

This is UART interrupt handle function. This function includes receive interrupt and send interrupt. Because this is a same interrupt number in receive interrupt and send interrupt.

```
typedef enum IRQn
{
    NMI_IRQn                                =  -14, /* Non Maskable Interrupt NMI */
    HardFault_IRQn                          =  -13, /* HardFault HardFault */
    SVC_IRQn                                =   -5, /* SV Call SVC */
    PendSV_IRQn                             =   -2, /* Pend SV PendSV */
    SysTick_IRQn                            =   -1, /* System Tick SysTick */

    CSV_SWDT_LVD_IRQn                       =    0, /* CSV_SWDT_LVD_IRQ */
    MFS0_RX_TX_IRQn                         =    1, /* MFS0 RX TX IRQ */
    MFS1_RX_TX_IRQn                         =    2, /* MFS1 RX TX IRQ */
    MFS3_RX_TX_IRQn                         =    4, /* MFS3_RX_TX_IRQ */
    MFS4_RX_TX_IRQn                         =    5, /* MFS4_RX_TX_IRQ */
```

It includes communication error verify, data number check function in receive handle.

```
void MFS1_RX_TX_IRQHandler(void)
{
    if ( UART_DBG_CH->SSR_f.RDRF) //RX interrupt
    {
        UART_DBG_CH->SCR_f.TIE = 0;  /* disable uart interrupt */
        UART_DBG_CH->SCR_f.TIE = 0;  /* dummy, for time delay */

        if ( UART_DBG_CH->SSR_f.FRE || UART_DBG_CH->SSR_f.ORE )  /* f
        {
        else
        {
    }

    else  //TX interrupt
    {
        UART_DBG_CH->SCR_f.TIE = 0;  /* disable uart interrupt */
        UART_DBG_CH->RDR = send_buf;
    }
    return;
}
```

- uart_receive:

This function is UART low level receive function. This function is waiting for result of receive interrupt handle till the receive buffer full in this function, this function includes data receive handle, error handle, time out handle and buffer check handle.

```
uint8_t uart_receive(uint8_t *DataPtr, uint32_t time_out, uint8_t cnt)
{
    uint8_t  ret = 0;
    uint8_t  timecnt=0;
    if(uart_error) {uart_error = 0; return 1;}  /* if error has happene
    uart_error = 0;                             /* clear uart status */
    bt0_enable(0, time_out);                     /* enable dt timer wit
    while(1)
    {
    bt0_disable();
    return ret;
}
```

- uart_send:

This function is UART low level send function, when data sent, BGM Adapter MCU will occurred Tx and Rx interrupt, the Tx interrupt is for transmit data to UART send/receive register, the Rx interrupt for verify send data, because the UART is one line UART, SI and SO is a same line.

it includes send data, receive data for verify send data, time out handle, and data verification error handle.

```
uint8_t uart_send(uint8_t *DataPtr, uint32_t time_out)
{
    uint8_t  ret = 0;
    if(uart_error) {uart_error = 0; return 1;} /* if err
    send_buf = *DataPtr;                        /* store
    uart_error = 0;                             /* clear
    p_wr_buf = p_rd_buf = &rev_buf[0];
    UART_DBG_CH->SCR_f.TIE = 1;                 /* enabl
    bt0_enable(1, time_out);                     /* enab
    while(1)
    {
    bt0_disable();
    return ret;
}
```

- uart_init:

this function is UART initialization.

```
void uart_init(uint8_t Baudrate)
{
    uint32_t u32baudrate;
    if(Baudrate == UART_BAUD_RATE_62500)
      u32baudrate = UART_BAUDRATE(62500);
    else if(Baudrate == UART_BAUD_RATE_500k)
      u32baudrate = UART_BAUDRATE(500000);
    else if(Baudrate == UART_BAUD_RATE_125000)
      u32baudrate = UART_BAUDRATE(125000);

    SetPinFunc_SIN1_1();
    SetPinFunc_SOT1_1();

    UART_DBG_CH->SCR =  0x80;
    UART_DBG_CH->SMR =  0x01;
    UART_DBG_CH->ESCR = 0x00;
    UART_DBG_CH->BGR = (uint16_t)u32baudrate;
    UART_DBG_CH->SCR = 0x03;
    UART_DBG_CH->SSR_f.REC = 0x01;

    NVIC_ClearPendingIRQ(MFS1_RX_TX_IRQn);
    NVIC_SetPriority(MFS1_RX_TX_IRQn, 1);
    NVIC_EnableIRQ(MFS1_RX_TX_IRQn);
    return;
}
```

- bt0 functions

the base timer 0 is used for counting UART communication time. The bt0 functions include bt0_enable, bt0_disable, check_bt0_overflow, and clear_bt0_flag.

There have two modes for selection in bt0_enable function, the modes include one-shot timer and reload timer.

```
void bt0_enable(uint8_t one_shot, uint32_t timer)
{
    FM_BT0->TMCR_f.FMD = 0;
    FM_BT0->TMCR_f.FMD = 0x03;                  /
    FM_BT0->TMCR_f.CKS = 0x02;                  /
    FM_BT0->TMCR_f.T32 = 1;                     /
    FM_BT0->TMCR_f.EGS = 0;                     /
    FM_BT0->TMCR_f.OSEL = 0;                    /
    FM_BT0->STC_f.TGIE = 0u;
    FM_BT0->STC_f.UDIE = 0u;
    if(one_shot)
    {
        FM_BT0->TMCR_f.MDSE = 1;
    }
    else
    {
        FM_BT0->TMCR_f.MDSE = 0;
    }
    FM_BT0->STC_f.TGIR = 0u;
    FM_BT0->STC_f.UDIR = 0u;
    FM_BT1->PCSR = ((timer*2u) >> 16) & 0xffff;
    FM_BT0->PCSR = ((timer*2u)) & 0xffff;
    FM_BT0->TMCR_f.CTEN = 1u;
    FM_BT0->TMCR_f.STRG = 1u;

}
```

8.2.8 Misc

Many functions include initialization the GPIO, external interrupt and so on in this section.

8.2.8.1 Port function instruction

GPIO:

- P10: Controls power for target board MCU
- P14: Detects mode key, this function is reserved.
- P3B: Red LED for indication CR state.
- P3C: Green LED for indication USB communication state.
- P52: Reset control for target board MCU.
- P53: Controls the voltage of 1-line UART. When power on, we will control this pin to pull down DBG line

External interrupt:

- P4E: Detects UVCC over current.
- P50: Detects reset signal of target board MCU.
- P51: Detects power fail from target board MCU.

8.2.8.2 Function

- bgma_io_initial

This function initializes all GPIO ports when power on.

```
void bgma_io_initial(void)
{
    /* Initialize Red/Green LEDs */
    Gpiolpin_InitOut( GPIO1PIN_P3B, Gpiolpin_InitVal( 0u ) );    // Red LED
    Gpiolpin_InitOut( GPIO1PIN_P3C, Gpiolpin_InitVal( 0u ) );    // Green LED

    /* Set POUTFLX (over-current detection) as input port*/
    Gpiolpin_InitIn ( GPIO1PIN_P4E, Gpiolpin_InitPullup( 0u ) );  // P4E
    // SetPinFunc_INT06_2(0u);                      // P4E
    // Gpio1pin_InitIn ( GPIO1PIN_P4E, Gpio1pin_InitPullup(1u));  // Pull-up

    /* Set RSTX (reset MCU) as output port*/
    Gpiolpin_InitOut( GPIO1PIN_P52, Gpiolpin_InitVal( 1u ) );     // P52
    Gpiolpin_Put( GPIO1PIN_P52, 1);   // Output 'H'

    /* Set RSTMX as input port*/
    /* Monitor reset accessed by reset button in user board */
    SetPinFunc_INT00_0(0u);                         // P50
    // Gpio1pin_InitIn ( GPIO1PIN_P50, Gpio1pin_InitPullup(1u));  // Pull-up

    /* Set UVCCM as input port */
    /* Monitor UVCC of LPC, if BGM find power fall, an interrupt will be happene
```

- enable_uvccm_int

This function is initialized and enabled external interrupt for detecting UVCC level.

This interrupt edge is falling.

```
void enable_uvccm_int(void)
{
    FM0P_GPIO_PFR5 |= 0x2;
    bFM0P_EXTI_ENIR_EN1  = 0;  // Disable external interrupt
    bFM0P_EXTI_EICL_ECL1 = 0;  // Clear external interrupt factor
    bFM0P_EXTI_ELVR_LA1  = 1;
    bFM0P_EXTI_ELVR_LB1  = 1;  // Detects the falling edge
    bFM0P_EXTI_ENIR_EN1  = 1;  // Enable external interrupt

    NVIC_ClearPendingIRQ(EXINT0_1_IRQn);
    NVIC_SetPriority(EXINT0_1_IRQn, 3);
    NVIC_EnableIRQ(EXINT0_1_IRQn);
}
```

- enable_rstmx_int

This function is detected and enable external interrupt for detecting Reset signal of target board MCU.

This interrupt edge is falling.

```
void enable_rstmx_int(void)
{
    FM0P_GPIO_PFR5 |= 0x1;
    bFM0P_EXTI_ENIR_EN0  = 0;  // Disable external interrupt
    bFM0P_EXTI_EICL_ECL0 = 0;  // Clear external interrupt factor
    bFM0P_EXTI_ELVR_LA0  = 1;
    bFM0P_EXTI_ELVR_LB0  = 1;  // Detects the falling edge
    bFM0P_EXTI_ENIR_EN0  = 1;  // Enable external interrupt

    NVIC_ClearPendingIRQ(EXINT0_1_IRQn);
    NVIC_SetPriority(EXINT0_1_IRQn, 3);
    NVIC_EnableIRQ(EXINT0_1_IRQn);
}
```

- enable_poutflx_int

This function is detected and enable external interrupt for detecting UVCC over current of target board MCU.

This interrupt edge is falling.

```
void enable_poutflx_int(void)
{
    FM0P_GPIO_PFR6 |= 0x4000;
    bFM0P_EXTI_ENIR_EN6  = 0;  // Disable external interrupt
    bFM0P_EXTI_EICL_ECL6 = 0;  // Clear external interrupt factor
    bFM0P_EXTI_ELVR_LA6  = 1;
    bFM0P_EXTI_ELVR_LB6  = 1;  // Detects the falling edge
    bFM0P_EXTI_ENIR_EN6  = 1;  // Enable external interrupt

    NVIC_ClearPendingIRQ(EXINT6_7_IRQn);
    NVIC_SetPriority(EXINT6_7_IRQn, 3);
    NVIC_EnableIRQ(EXINT6_7_IRQn);
}
```

- power_on_init

This function is initialized system parameters, UART function and interrupt function when target board MCU power on.

```
void power_on_init(void)
{
    power_state &= ~LOW_POWER;      /* clear LOW_POWER flag */
    power_state &= ~OVER_CURRENT;   /* clear OVER_CURRENT flag */
    monitor_state &= ~MCU_RUN;      /* clear MCU_RUN flag,set MCU_BREAK flag */
    Delay_ms(1000);                 /* 1000ms, wait 1500*700us */
    uart_init(UART_BAUD_RATE_62500); /* initial UART */
    enable_uvccm_int();
    enable_rstmx_int();
    enable_poutflx_int();
    monitor_state |= TRIMMING;          /* doing trim after power on */
    retry_counter = 0;
    if ((monitor_state & MON_START) == MON_START){
        monitor_state &= ~MON_START; /* exit monitor mode */
        power_state |= LOWPOWERIDT;  /* need indicate power off during in monitor mode */
        power_state &= ~RST_ERROR;   /* clear RST error flag after exit monitor mode */
    }
}
```

- EXINT0_1_IRQHandler

This function is interrupt handle for detecting Ext0(Reset signal) and Ext1(UVCC level).

```
void EXINT0_1_IRQHandler(void)
{
    /* RSTMX interrupt, target MCU reset */
    if ( (1u==bFM0P_EXTI_EIRR_ER0) && (0u==IO_RSTMX) )
    {
        power_state |= USER_RST;
        bFM0P_EXTI_ENIR_EN0 = 0u;  // Disable this external interrupt
    }
    /* UVCCM interrupt, target MCU power off */
    if ( (1u==bFM0P_EXTI_EIRR_ER1) && (0u==IO_UVCCM) )
    {
    bFM0P_EXTI_EICL_ECL0 = 0u;  // Clear interrupt flag
    bFM0P_EXTI_EICL_ECL1 = 0u;  // Clear interrupt flag
    /* Clear the interrupt flag */
    NVIC_ClearPendingIRQ(EXINT0_1_IRQn);
}
```

- EXINT6_7_IRQHandler

This function is interrupt handle for detecting Ext6(UVCC Over current).

```
void EXINT6_7_IRQHandler(void)
{
    if ( 0u == IO_POUTFLX )
    {
        power_state |= OVER_CURRENT;
        /* Control to power off LPC, P10 [POUTEN] */
        Gpiolpin_Put( GPIO1PIN_P10, 0);   // Output 'L'
        bFM0P_EXTI_ENIR_EN6  = 0;  // Disable external interrupt
    }

    bFM0P_EXTI_EICL_ECL6 = 0u;  // Clear interrupt flag
    NVIC_ClearPendingIRQ(EXINT6_7_IRQn);
}
```

8.2.8.3 System tick

Provide a software timer for system delay and timeout which is used to the system tick timer.

This interrupt interval is 1ms.

```
uint8_t TimerOverTimeMs(uint32_t *pTimer, uint32_t TimeOutVal)
{
    uint32_t CurrentTime;
    uint32_t ElapseTime;

    /* Store current system time */
    CurrentTime = SystemTime;
    /* Reset setting */
    if ( 0 == TimeOutVal )
    {
    /* Over a round */
    if ( (*pTimer) > CurrentTime )
    {
        ElapseTime = 0xFFFFFFFF - (*pTimer) + CurrentTime + 1;
    }
    else
    {
        ElapseTime = CurrentTime - (*pTimer);
    }

    /* Check if timeout or not */
    if ( ElapseTime >= TimeOutVal )
    {
        *pTimer += TimeOutVal;
        return 1u;
    }
    return 0u;
}
```

8.3 Software Design – N/A

8.4 Mechanical – N/A

8.5 Manufacturing

There is no specific design for manufacturing or manufacturing test items in this project.

9. QUALITY REQUIREMENTS

9.1 System Hardware Design for Test

9.2 System Firmware Design for Test

9.2.1 Unit Testing

9.2.1.1 Protocol Analysis

Test the 24 USB command handler functions, check the UART return data from target MCU.

9.3 System Software Design for Test: N/A

10. RECORDS

10.1 Storage location and retention period for records is specified in procedure 00-00064 Cypress Record Retention Policy.

11. PREVENTIVE MAINTENANCE: N/A

12. POSTING SHEETS/FORMS/APPENDIX

Appendix

Document History Page

Document Title: New 8Fx Adapter
Document Number: 001-xxxxx

Rev.	ECN No.	Orig. of Change	Description of Change
**	xxxxxxx	CHMA, KOZU, HAXI	New release

Distribution: WHAO,

Posting: None