乐升半导体
LEVETOP

LT168

串口屏控制芯片

脚位对应软件配置说明

V1.1

www.levetop.cn

Levetop Semiconductor Co.，Ltd.

版本记录

版 本	日 期	说 明
V1.0	2024/4/24	初版
V1.1	2024/5/28	修改 PWM 输入和 EBI 配置函数错误

版权说明

目　　录

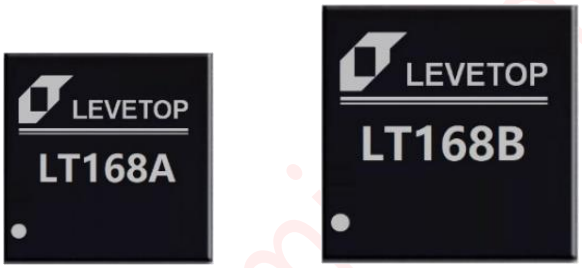
1. 前言

LT168 是一款 TFT 串口屏显示控制芯片。它包含一个 32 位 MCU 和图形显示模块，提供串口通信协议，其中 LT168A 支持 8 位并口的 8080 MCU 或是 SPI 接口的 TFT 屏,而 LT168B 支持 8 位/16 位并口的 8080 MCU 接口或是 RGB/SPI/QSPI 串口接口的 TFT 屏。RGB/QSPI 接口的 TFT 屏分辨率可以支持到 480*480；8 位/16 位 8080 MCU 接口的 TFT 屏分辨率可以支持到 800*480。

由于含有高容量的 Flash、SRAM 及众多 IO 接口，LT168 可以将部分接口资源拿来使用，或是将 LT168 作为主控的 MCU，将主控及 TFT 显示功能由一颗 LT168A/B 来完成，本说明书介绍在使用 LT168 的脚位时所对应的软件配置。

LT168 有两种封装及型号，分别如下：

- QFN48 (6*6 mm^2) – LT168A
- QFN68 (8*8 mm^2) – LT168B



QFN48 (6*6 mm^2)　　QFN68 (8*8 mm^2)

图 1-1：LT168A 和 LT168B 外观图

表 1-1：LT168 型号

型号	封装	内建 Flash	内建 SRAM	适用的 TFT 屏
LT168A	QFN48 (6*6 mm^2)	512K Bytes	256K Bytes + 512K Bytes	● 8-bits 8080 MCU 接口 TFT 屏
LT168B	QFN68 (8*8 mm^2)	2M Bytes	256K Bytes + 512K Bytes	● 8/16-bits 8080 MCU 接口 TFT 屏 (Max. 800*480) ● RGB 565 接口 TFT 屏 (Max. 480*480) ● QSPI 接口 TFT 屏

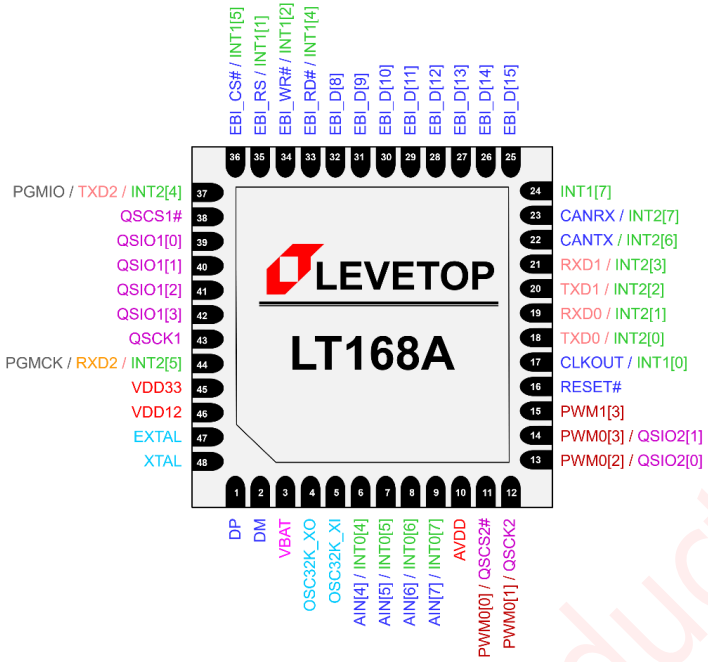


图 1-2：LT168A (QFN48) 脚位图

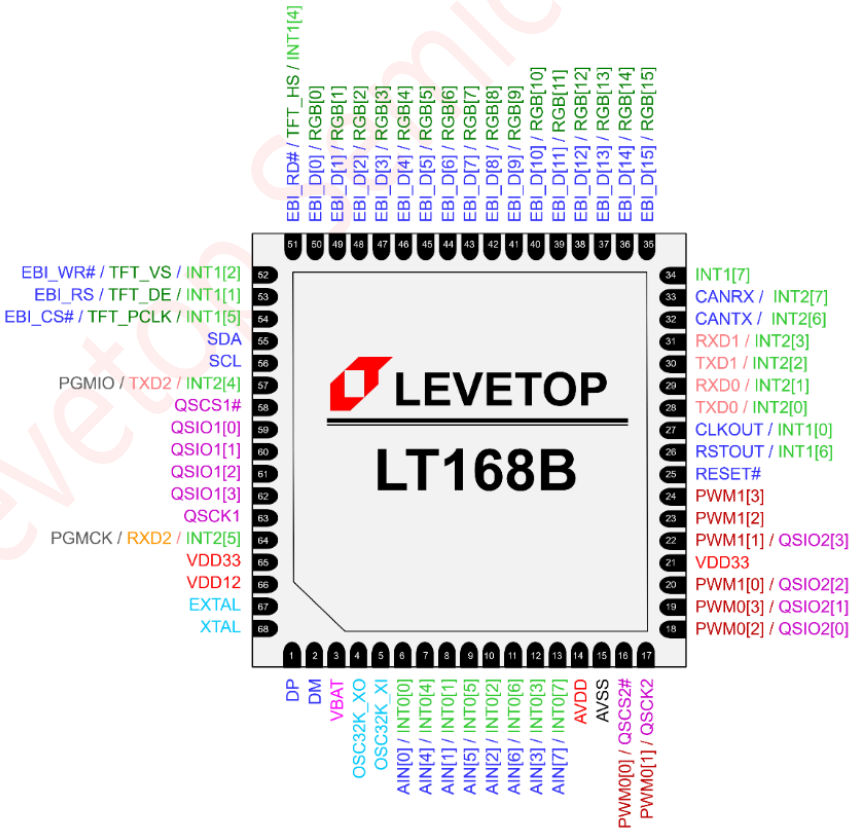


图 1-3：LT168B (QFN68) 脚位图

2. AIN[X]/INT0[X] 端口

对应芯片脚位 LT168A: Pin6-Pin9；LT168B: Pin6-Pin13

此组端口默认是 ADC 功能，ADC 程序配置请参考 SDK 的 ADC demo 程序:

```
void ADC_Init(void)                          //初始化函数
unsigned short Get_ADC_Val(ADC_CH ch)        //读取对应 ADC 脚位的值。
```

此组端口配置为 EPORT 功能，程序参考如下:

```
//ADC[0-7] 配置为 Eport0[0-7]---------------
OPTION->ADCCDISR = 0x0000;
OPTION->ADCCDISR = 0x4000;
OPTION->ADCCDISR = 0x8000;
OPTION->ADCCDISR = 0xC000;
OPTION->ADCCDISR |= 0xC000|(1<<7)|(1<<6)|(1<<5)|(1<<4)|(1<<3)|(1<<2)|(1<<1)|(1<<0);
//EPORT0[X]端口需要以上配置，EPORT1[X]/EPORT2[X] 端口不需要。
```

配置端口输入输出寄存器：EPORT0->EPDDR
配置端口输出高低电平寄存器：EPORT0->EPDR
读取端口高低电平寄存器：EPORT0->EPPDR
配置端口上拉使能寄存器：EPORT0->EPPUE

2.1. 例程 1 配置 AIN[0]/INT0[0] 输出高低电平

```
OPTION->ADCCDISR = 0x0000;
OPTION->ADCCDISR = 0x4000;
OPTION->ADCCDISR = 0x8000;
OPTION->ADCCDISR = 0xC000;
OPTION->ADCCDISR |= 0xC000|(1<<0); //配置 AIN[0]/INT0[0] 为 EPORT
//EPORT0[X] 端口需要以上配置，EPORT1[X]/EPORT2[X] 端口不需要。

EPORT0->EPDDR |= (1<<0);             //配置 EPORT0[0] 为输出
EPORT0->EPDR |= (1<<0);              //配置 EPORT0[0] 输出高电平
EPORT0->EPDR &= ~(1<<0);             //配置 EPORT0[0] 输出低电平
```

2.2. 例程 2 配置 AIN[3]/INT0[3] 输入并读取电平状态

```
unsigned char temp;
OPTION->ADCCDISR = 0x0000;
OPTION->ADCCDISR = 0x4000;
OPTION->ADCCDISR = 0x8000;
OPTION->ADCCDISR = 0xC000;
OPTION->ADCCDISR |= 0xC000|(1<<3);        //配置 AIN[3]/INT0[3] 为 EPORT
//EPORT0[X] 端口需要以上配置，EPORT1[X]/EPORT2[X] 端口不需要。

EPORT0->EPDDR &= ~(1<<3);                 //配置 EPORT0[3] 为输入
EPORT0->EPPUE|= (1<<3) ;                  //配置 EPORT0[3] 内部上拉使能
Temp =(EPORT0->EPPDR)& (1<<3) ;           //读取 EPORT0[3] 状态
```

2.3. 例程 3 配置 AIN[5]/INT0[5] 为 EPORT 外部中断

```
OPTION->ADCCDISR = 0x0000;
OPTION->ADCCDISR = 0x4000;
OPTION->ADCCDISR = 0x8000;
OPTION->ADCCDISR = 0xC000;
OPTION->ADCCDISR |= 0xC000|(1<<5);        //配置 AIN[5]/INT0[5] 为 EPORT
//EPORT0[X] 端口需要以上配置，EPORT1[X]/EPORT2[X] 端口不需要。

//上升沿/下降沿中断--------------------------------
EPORT0->EPDDR &= ~(1<<5);                 //配置 EPORT0[5] 为输入
EPORT0->EPIER |= (1<<5);                  //配置 EPORT0[5] 中断使能
EPORT0->EPPAR = 0x0400;                   //1: Rising edge triggered
//EPORT0->EPPAR = 0x0800;                 //2: Falling edge triggered
//EPORT0->EPPAR = 0x0C00;                 //3: Rising and Falling edge triggered
EPORT0->EPFR = (1<<5);                    //清除 EPORT0[5] 中断标志
EIC->IER |= IRQ(30);                      //配置 EPORT0[5] 全局中断使能

void EPORT0_Handler(void)                 //中断函数
{
  unsigned char temp;
  temp = EPORT0->EPFR;
//User'S code...
  EPORT0->EPFR = temp;
}
```

```
//高/低电平中断--------------------------------
//EPORT0->EPPUE|= (1<<5);          //配置 EPORT0[5] 内部上拉使能，低电平中断时打开
//EPORT0->EPPUE &= ~(1<<5);        //配置 EPORT0[5] 关闭内部上拉，高电平中断时打开
EPORT0->EPDDR &= ~(1<<5);          //配置 EPORT0[5] 为输入
EPORT0->EPPAR &= 0xF3FF;           //0:HIGH/LOW level-sensitive,
//EPORT0->EPLPR |= (1<<5);         //HIGH level-sensitive if EPPAR = 0；高电平中断时打开
//EPORT0->EPLPR &= ~(1<<5);        //LOW level-sensitive if EPPAR = 0；低电平中断时打开
EPORT0->EPIER |= (1<<5);           //配置 EPORT0[5] 中断使能
EIC->IER |= IRQ(30);               //配置 EPORT0[5] 全局中断使能

void EPORT0_Handler(void)
{
    EPORT0->EPIER &= ~(1<<5);    //关闭 EPORT0[5] 中断
    //User'S code...
}
```

Note：Eport 的高低电平中断不能一直打开，进中断后关闭中断，需要时再打开。

	15	14	13	12	11	10	9	8
Read: Write:	EPPA7[1:0]		EPPA6[1:0]		EPPA5[1:0]		EPPA4[1:0]	
Reset:	0	0	0	0	0	0	0	0
	7	6	5	4	3	2	1	0
Read: Write:	EPPA3[1:0]		EPPA2[1:0]		EPPA1[1:0]		EPPA0[1:0]	
Reset:	0	0	0	0	0	0	0	0

图 2-1：EPPAR 寄存器

3. (PWM0[0-3], PWM1[0-1])/QSPI2, PWM1[2-3] 端口

对应芯片脚位 LT168A: Pin1-Pin15；LT168B: Pin16-Pin24

此组端口默认是 PWM 功能，PWM 功能配置如下，以 PWM1[3] 为例：

```
void PWM_OutputInit(uint16_t PWM_Width)
{
    // prescale: PWM_Prescaler+1
    // PWM_Csr: 0=1/2, 1=1/4, 2=1/8, 3=1/16, others=1
    // period: WM_Period+1
    // Actual PWM output: 75Mhz/(PWM_Prescaler+1)/(PWM_Csr)/(PWM_Period+1)
    // prescale: PWM_Prescaler+1
    // CP: 0=stop, others=CP+1
    // CSR: 0=2, 1=4, 2=8, 3=16, others=1
    // PCNR: 0=stop,
    // PCMR: CMR+1
    // period: PCNR+1
    // Actual PWM output: 168Mhz/CSR/CP/PCNR

    PWM1->U32_PCSR.CSR3 = 2;        //时钟分频(设置 2=8 分频,即时钟为 sys_clk:168/8=21Mhz)
    PWM1->U32_PPR.CP1 = 0;          //时钟预分频(0=未分频，最终输出时钟还是 21Mhz)
    PWM1->U32_PCR.CH3MOD = 1;       //定时器输出反转开关(1：反转开，0：反转关)
    PWM1->PCNR3 = 1050;             //周期时间 (21Mhz/1050=20Khz，即输出为 20Khz 的频率)
    PWM1->PCMR3 = 525;              //设置占空比，即最后的占空比为（525/1050）*100% = 50%
    PWM1->U32_PPCR.PDDR_3 = 1;      //设置 PWM13 为输出
    PWM1->U32_PCR.CH3EN = 1;        //使能 PWM13
}

void PWM_OutputClose(void)
{
    PWM1->U32_PCR.CH3EN = 0;
}

void PWM_OutputOpen(void)
{
    PWM1->U32_PCR.CH3EN = 1;
}
```

```
void PWM_Output_DutySet(uint16_t duty)
{
    if (duty == 0)
        LT_BacklightClose();
    else
    {
        PWM1->PCMR3 = duty;        //high leve = duty+1
        LT_BacklightOpen();
    }
}
```

以 PWM 配置为 GPIO 输出/输入,PWM1[3] 为例:

```
unsigned char temp;
PWM1->U32_PPCR.PDDR_3 = 1;        //PWM1[3] 配置为输出
PWM1->U32_PPCR.PDR_3 = 1;         //PWM1[3] 输出高电平
PWM1->U32_PPCR.PDR_3 = 0;         //PWM1[3] 输出低电平
PWM1->U32_PPCR.PDDR_3 = 0;        //PWM1[3] 配置为输入
//PWM 默认是内部下拉，如果硬件默认是高电平，需要程序关闭内部下拉使能，外接上拉电阻。
CCM->U32_CPPDC.pwm1_3 = 0;        //PWM1[3] Disable Pull down function
Temp = PWM1->U32_PPCR.PDR_3;      //读取 PWM1[3] 电平状态
```

此组端口配置为 QSPI 功能，对应端口(PWM0[0-3], PWM1[0-1])/QSPI2

```
void PIN_QSPI2(void)                    //配置端口复用为 QSPI2
{
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x01);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x02);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x03);
    OPTION->CCR |= (1 << 8);
    OPTION->CCR &= 0xFFFC;
}
```

QSPI 功能请参考串口屏工程中驱动 QSPI 屏部分代码:

```
void LCD_QSPI_STD_Init(void);
uint16_t LCD_QSPI_ReadWriteByte(uint32_t TxData);
```

此组端口配置为普通 4 线 SPI 功能，对应端口 PWM0[0-3]，SPI 功能请参考串口屏工程中驱动 SPI 屏部分代码:

```
void LCD_SPI_Port_Init(void);
uint16_t LCD_SPI_ReadWriteByte(uint32_t TxData);
```

4. RSTOUT/INT1[6], CLKOUT/INT1[0] 端口

对应芯片脚位 LT168A: Pin17；LT168B: Pin26, Pin27

此组端口默认配置为 RSTOUT/CLKOUT 功能，需通过程序配置为 EPORT1[6]/EPORT1[0]：

```
void PIN_EPORT1_6(void)  //RSTOUT 端口配置为 EPORT1[6]
{
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x01);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x02);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x03);
    OPTION->CCR |= (1 << 11);
    OPTION->CCR &= 0xFFFC;
}

void PIN_EPORT1_0(void)  //CLKOUT 端口配置为 EPORT1[0]
{
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x01);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x02);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x03);
    OPTION->CCR |= (1 << 10);
    OPTION->CCR &= 0xFFFC;
}
```

EPORT1[X] 程序可以参考第 2 章 Eport 配置程序。

5. TXD0/INT2[0], RXD0/INT2[1], TXD1/INT2[2], RXD1/INT2[3] 端口

对应芯片脚位 LT168A: Pin18-Pin21；LT168B: Pin28-Pin31

此 2 组端口默认为串口功能，以 Uart1 为例，程序如下：

```
#define UART_RECV_MAX_LEN   0x100
#define UART_PARITY_EVEN        0
#define UART_PARITY_ODD         1
#define UART_PARITY_NONE        2
typedef struct _uart_recv_buf
{
    UINT16 rp;                              //read pointer
    UINT16 wp;                              //write pointer
    UINT8   data[UART_RECV_MAX_LEN];    //buffer
}UartRecvBufStruct;
static UartRecvBufStruct g_Uart0RecvBufStruct = {0, 0};   //uart1 buffer
static UartRecvBufStruct g_Uart1RecvBufStruct = {0, 0};   //uart2 buffer
static UartRecvBufStruct g_Uart2RecvBufStruct = {0, 0};   //uart3 buffer

void UART_Init(sci *UARTx,uint32_t baudrate,uint8_t parity)
{
    uint16_t SBR = 0;
    uint32_t tempReg = 0;
    SBR = Find_SCI_Optimal_SBR(baudrate);
    Bit_Clear(UARTx->BAUD,SCI_BAUD_SBR_MASK);  //Baud rate = IPS/((OSR+1)*SBR) ,1<SBR<8191
    Bit_Set(UARTx->BAUD,SCI_BAUD_SBR(SBR));
    UARTx->OSR = SCI_OSR( SYS_CLOCK / ((SBR) * baudrate) - 1);       //3<=OSR<=255,如果小于3则会被设为 15
    Bit_Set(UARTx->FIFO,SCI_FIFO_TXFE_MASK);                //发送机的 FIFO 使能
    tempReg |= SCI_WATER_TXWATER(7);                        //发送机的水印寄存器
    UARTx->WATER = tempReg;
    //Bit_Set(UARTx->CTRL,);                                //发送使能
    UARTx->CTRL=0;
    UARTx->CTRL=(SCI_CTRL_TE_MASK|SCI_CTRL_RE_MASK|SCI_CTRL_RIE_MASK);
    if (parity != UART_PARITY_NONE)
    {
      UARTx->CTRL |= SCI_CTRL_PE_MASK;                      //Parity enable
      UARTx->CTRL |= parity;                                //odd parity
    }
```

```
    else UARTx->CTRL &= ~SCI_CTRL_PE_MASK;
    if (SCI0 == UARTx)
    {
      g_Uart0RecvBufStruct.wp = 0;
      g_Uart0RecvBufStruct.rp = 0;
      CCM->EPORT2FCR &= (~0x03);
      EIC->U32_IER.IE_SCI0 = 1;
    }
    else if (SCI1 == UARTx)
    {
      g_Uart1RecvBufStruct.wp = 0;
      g_Uart1RecvBufStruct.rp = 0;
      CCM->EPORT2FCR &= (~0x0C);
      EIC->U32_IER.IE_SCI1 = 1;
    }
    else
    {
      g_Uart2RecvBufStruct.wp = 0;
      g_Uart2RecvBufStruct.rp = 0;
      CCM->EPORT2FCR &= (~0x30);
      EIC->U32_IER.IE_SCI2 = 1;
    }
}

//Uart1 发送函数
void UART1_SendByte( UINT8 data)
{
    while((SCI1->STAT&SCI_STAT_TDRE_MASK)==0);
    SCI1->DATA = data&0xff;
}

void UART1_SendBytes(uint8_t *buffer, uint16_t length)
{
    UINT16 i = 0;
    for ( ; i < length; i++)
    {
      while((SCI1->STAT&SCI_STAT_TDRE_MASK)==0);
      SCI1->DATA = buffer[i]&0xff;
    }
}
```

```
//Uart1 中断接收函数
void SCI1_Handler(void)
{
    uint8_t ch = 0;
    if(Bit_Read(SCI1->STAT,SCI_STAT_RDRF_MASK))
    {
     ch = (SCI1->DATA &0xFF );
    //User'S Code...
    //LTPrintf("ch   %x   \r\n", ch);
    }
}
```

此 2 组端口配置为 Eport，程序如下：

```
void PIN_EPORT2_0(void)        //配置 TXD0 为 EPORT2[0]
{
    Bit_Set(CCM->EPORT2FCR, 1 << 0);
}

void PIN_EPORT2_1(void)        //配置 RXD0 为 EPORT2[1]
{
    Bit_Set(CCM->EPORT2FCR, 1 << 1);
}

void PIN_EPORT2_2(void)        //配置 TXD1 为 EPORT2[2]
{
    Bit_Set(CCM->EPORT2FCR, 1 << 2);
}

void PIN_EPORT2_3(void)        //配置 RXD1 为 EPORT2[3]
{
    Bit_Set(CCM->EPORT2FCR, 1 << 3);
}
```

EPORT2[X] 程序可以参考第 2 章 Eport 配置程序。

6. CANTX/INT2[6], CANRX/INT2[7] 端口

对应芯片脚位 LT168A: Pin22-Pin23；LT168B: Pin32-Pin33

此组端口默认为 CAN 功能，程序请参考 CAN 测试 demo

此组端口配置为 Eport，程序如下：

```
void PIN_EPORT2_6(void)        //配置 CANTX 为 EPORT2[6]
{
    Bit_Set(CCM->EPORT2FCR, 1 << 6);
}

void PIN_EPORT2_7(void)        //配置 CANRX 为 EPORT2[7]
{
    Bit_Set(CCM->EPORT2FCR, 1 << 7);
}
```

EPORT2[X] 程序可以参考第 2 章 Eport 配置程序。

7. BOOT/INT1[7] 端口对应芯片脚位

对应芯片脚位 LT168A: Pin24；LT168B: Pin34

此端口默认为 EPORT1[7]，程序可以参考第 2 章 Eport 配置程序。BOOT/INT1[7] 是启动位检测端口，上电期间不可以为低电平。

8. EBI_D[0-15], EBI_CS#, EBI_RS, EBI_WR#, EBI_RD# 端口

对应芯片脚位 LT168A: Pin26-Pin36；LT168B: Pin35-Pin54

此组端口默认为 MCU8 位/MCU16 位/RGB 屏硬件接口，对应脚位不可更改，当这些端口不接屏时，可以通过程序配置为 GPIO 端口。屏接口程序在串口屏工程代码中已写好驱动，可以不用修改。

配置为 GPIO 端口程序如下：

```
//配置 EBI_CS#/EBI_RS/EBI_WR#/EBI_RD# 端口为 EPORT1[5]/EPORT1[1]/EPORT1[2]/EPORT1[4]
void PIN_EPORT1_1245(void)
{
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x01);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x02);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x03);
    OPTION->CCR &= ~(1 << 15);
    OPTION->CCR &= 0xFFFC;
    /* EBI switch EPORT1_1,EPORT1_2,EPORT1_4,EPORT1_5 */
    MDPI->IOCR |= 1 << 1;

    Bit_Set(MDPI->IOCR, 1 << 17);
}
```

EPORT1[X] 程序可以参考第 2 章 Eport 配置程序。

```
//EBI_D[0-15] 端口配置为 EBI 功能--------------------------------
void PIN_EBI_Default(void)
{
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x01);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x02);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x03);
    OPTION->CCR &= ~(1 << 15);
    OPTION->CCR &= 0xFFFC;
}

void EBI_Port_15_8_As_GPIO_Init(FunctionState state)      //配置 EBI_D[8-15] 为 GPIO 功能
{
    Bit_Clear(MDPI->IOCR, EBI_IOCR_D3_GP3_MASK);
    Bit_Set(MDPI->IOCR, EBI_IOCR_D3_GP3(state));
}
```

```
void EBI_Port_7_0_As_GPIO_Init(FunctionState state)        //配置 EBI_D[0-7] 为 GPIO 功能
{
    Bit_Clear(MDPI->IOCR, EBI_IOCR_D2_GP2_MASK);
    Bit_Set(MDPI->IOCR, EBI_IOCR_D2_GP2(state));
}

//配置 EBI_D[0-15] 端口内部上拉使能
void EBI_Port_PullUp_Config(ebi_pin_te pin, FunctionState state)
 {
    Bit_Clear(MDPI->IOPCR, 1 << (pin + 16));
    Bit_Set(MDPI->IOPCR, state << (pin + 16));
}

//配置 EBI_D[0-15] 端口输出高低电平
void EBI_Port_GPIO_Output_Status(ebi_pin_te pin, BitStatus status)
{
    if(status == Reset)
        Bit_Clear(MDPI->GPIODO, 1 << (pin + 16));
    else
        Bit_Set(MDPI->GPIODO, 1 << (pin + 16));
}

//配置 EBI_D[0-15] 端口输入/输出
void EBI_Port_GPIO_Config(ebi_pin_te pin, ebi_port_gpio_direction_te dir)
{
    Bit_Clear(MDPI->GPIODIR, 1 << (pin + 16));
    Bit_Set(MDPI->GPIODIR, dir << (pin + 16));
}

//读取 EBI_D[0-15] 端口电平状态
BitStatus EBI_Port_GPIO_Input_Statu(ebi_pin_te pin)
{
    return ((MDPI->GPIODI >> (pin + 16)) & 0x01);
}
```

8.1. 例程 1 配置 EBI_D[0] 输出高低电平

```
PIN_EBI_Default();
EBI_Port_7_0_As_GPIO_Init(Enable);                    //配置 EBI_D[0-7] 为 GPIO
EBI_Port_GPIO_Config(EBI_Pin_0, EBI_Port_GPIO_Output);  //配置 EBI_D[0] 为输出
EBI_Port_GPIO_Output_Status(EBI_Pin_0, Set);          //配置 EBI_D[0] 输出高电平
EBI_Port_GPIO_Output_Status(EBI_Pin_0, Reset);        //配置 EBI_D[0] 输出高电平
```

8.2. 例程 2 配置 EBI_D[8] 输入并读取电平状态

```
unsigned char temp;
PIN_EBI_Default();
EBI_Port_15_8_As_GPIO_Init(Enable);                   //配置 EBI_D[8-15] 为 GPIO
EBI_Port_GPIO_Config(EBI_Pin_8, EBI_Port_GPIO_Input);   //配置 EBI_D[0] 为输出
EBI_Port_PullUp_Config(EBI_Pin_8, Enable);            //配置 EBI_D[8] 内部上拉使能
Temp = EBI_Port_GPIO_Input_Statu(EBI_Pin_8);          //读取 EBI_D[8] 状态
```

Note：EBI_D[0-15] 配置为 GPIO 功能时，没有外部中断功能

9. IIC_SCL, IIC_SDA 端口对应芯片脚位

对应芯片脚位 LT168B: Pin66-Pin56

硬件 IIC 程序配置请参考 SDK 的 IIC demo 程序,

```
//从机-----------------------------------
void I2CSlave_Init(I2C_TypeDef *I2Cx, uint16_t addrID);
void I2C1_IRQHandler(void);
void I2CSlave_SendBytes(I2C_TypeDef *I2Cx, uint8_t *buffer,uint16_t length);
bool I2CSlave_ReceiveBytes(I2C_TypeDef *I2Cx, uint8_t *buffer,uint16_t length);

//主机-----------------------------------
void I2CMaster_Init(I2C_TypeDef *I2Cx, uint8_t preScaler);
bool I2CMaster_SendBytes(I2C_TypeDef *I2Cx,uint16_t addrID, uint8_t *buffer,uint16_t length);
bool I2CMaster_ReceiveBytes(I2C_TypeDef *I2Cx,uint16_t addrID, uint8_t *buffer,uint16_t length);
```

配置为 GPIO 端口程序如下:

```
unsigned char temp;
I2C->I2CC = 0x00;
I2C->I2CPCR = 0xC3;          //配置 SCL/SDA 为 GPIO

I2C->I2CDDR = 0x03;          //配置 SCL/SDA 为 GPIO 输出
I2C->I2CPDR |= 0x01;         //SCL 输出高电平
//I2C->I2CPDR &= ~0x01;      //SCL 输出低电平
I2C->I2CPDR |= 0x02;         //SDA 输出高电平
//I2C->I2CPDR &= ~0x02;      //SDA 输出低电平

I2C->I2CDDR = 0x00;          //配置 SCL/SDA 为 GPIO 输入
Temp = I2C->I2CPDR &0x01;    //读取 SCL 电平状态
Temp = I2C->I2CPDR &0x02;    //读取 SDA 电平状态
```

10. PGMIO/TXD2/INT2[4], PGMCK/RXD2/INT2[5] 端口

对应芯片脚位 LT168A: Pin37, Pin44；LT168B: Pin57, Pin64

此 2 个端口默认为 PGM 下载功能，可通过程序配置为 UART2 或者 EPORT2[4-5]，

配置为 UART2 功能:

```
void PIN_RXD2_TXD2(void)
{
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x01);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x02);
    OPTION->CCR = ((OPTION->CCR & 0xFFFC) | 0x03);
    OPTION->CCR |= (1 << 12);
    OPTION->CCR &= 0xFFFC;
}

void PIN_TXD2_Default(void)          //先配置 PGMIO 为 TXD2
{
    Bit_Clear(CCM->EPORT2FCR, 1 << 4);
}

void PIN_RXD2_Default(void)          //先配置 PGMCK 为 RXD2
{
    Bit_Clear(CCM->EPORT2FCR, 1 << 5);
}
```

串口功能可参考第 5 章串口配置函数。

配置为 EPORT2[4-5] 功能:

```
PIN_RXD2_TXD2(void);                 //先配置为串口
void PIN_EPORT2_4(void)              //先配置 PGMIO 为 EPORT2[4]
{
    Bit_Set(CCM->EPORT2FCR, 1 << 4);
}

void PIN_EPORT2_5(void)              //先配置 PGMCK 为 EPORT2[5]
{
    Bit_Set(CCM->EPORT2FCR, 1 << 5);
}
```

11. QSPI1 端口

对应芯片脚位 LT168A: Pin38-Pin43；LT168B: Pin58-Pin63

此组端口只能作为 QSPI 功能或者 4 线 SPI 功能，配置程序可参考第 3 章程序说明。

12. 延时函数

```
void Delay_1Ms(void)
{
    EPT->EPTRLD = (SYS_CLOCK/1000)-1;
    EPT->EPTCNT = 0;
    EPT->U32_EPTCSR.CLKSRC = 1;
    EPT->U32_EPTCSR.INTEN = 0;
    EPT->U32_EPTCSR.CNTEN = 1;
    for (; (EPT->EPTCSR & 0x10001) == 0x01;)
    {
        asm("nop");
    }
    EPT->U32_EPTCSR.CNTEN = 0;
    EPT->EPTCNT = 0;
}

void DelayMS(volatile unsigned int delaymS)
{
    uint32_t ii;
    for(ii=0;ii<delaymS;ii++) Delay_1Ms();
}

void DelayUS(volatile unsigned int delayuS)
{
    uint8_t temp = 0;
    EPT->U32_EPTRLD.RLD = SYS_CLOCK/1000000UL*delayuS-1;
    EPT->U32_EPTCSR.INTEN =0;
    EPT->U32_EPTCSR.CLKSRC = 1;
    EPT->U32_EPTCSR.CNTEN = 1;
    for (; (EPT->EPTCSR & 0x10001) == 0x01;)
    {
        asm("nop");
    }
    EPT->U32_EPTCSR.CNTEN = 0;
    EPT->EPTCNT = 0;
}
```

13. PIT 定时器

```
void PIT1_Init(void)
{
    //PRE 0=1 1=2 2=4 3=8 4=16 5=32 6=64 7=128 ……
    //timeout period = PRE * (PM + 1) * clocks
    PIT1->PCSR=0x0;     //PIT3->U16PCSR.EN=0;        //清零 EN 位
    PIT1->U16_PCSR.PRE = 5;                          //预分频
    PIT1->PMR = 10*(SYS_CLOCK/1000)/32;              //（168MHZ）10ms timer
    PIT1->U16_PCSR.OVW = 1;
    PIT1->U16_PCSR.PIE = 1;                          //中断使能 PIE
    PIT1->U16_PCSR.RLD = 1;                          //重装位 RLD
    PIT1->U16_PCSR.EN = 1;                           //PIT 使能 EN
    EIC->IER |=IRQ(10);
}

void PIT1_Handler(void)
{
    PIT1->U16_PCSR.PIF = 1;                          //Clear PIF interrupt flag
    //User'S code...
}
```

14. EFlash 应用说明

```
#define flh_sAddr     0x6007F000              //(512K 地址开始)
Uint32_t DATA[5] = {x};
void SaveData(void)                           //4K 空间用于存储
{
    uint8_t buff[4096] = {0};

    memcpy(&buff[0], DATA[0], 4);
    memcpy(&buff[4], DATA[1], 4);
    memcpy(&buff[8], DATA[2], 4);
    memcpy(&buff[12], DATA[3], 4);
    memcpy(&buff[16], DATA[4], 4);

    EFlash_Init();
    EFLASH_Write(flh_sAddr,buff,4096);
}

uint8_t LT_TpGetAdjdata(void)
{
    uint8_t i, buff[4096] = {0};
    EFlash_Read(flh_sAddr,buff,20);
    memcpy(DATA[0], &buff[0], 4);
    memcpy(DATA[1], &buff[4], 4);
    memcpy(DATA[2], &buff[8], 4);
    memcpy(DATA[3], &buff[12], 4);
    memcpy(DATA[4], &buff[16], 4);
    return 1;
}
```