密级
机密

技术报告

名称： WH 处理器 API 接口说明

编号：

版本号：V0.1

作者 吕人杰

项目

日期 2018-09-18

深圳优矽科技有限公司

版本更新历史：

日期	版本	更新人	更新内容
2018-09-18	V0.1	吕人杰	首次发布

阅读说明

在阅读本文档之前，请先阅读该部分。

（1）本文档主要针对 WH 处理器上的各个外设例如 GPIO、UART、SPI 等做了相关介绍，并给了软件上调用的简单例子（部分）。

（2）本文档不仅限于 WH 处理器外设，同时也介绍了 LS 开发板上的外设调用过程，例如 OLED、ADC 等。

（3）本文档所有提及的寄存器操作都是基于指针寻址的操作，各个设备的物理地址在相应设备说明中已经列出，同时每个设备的功能寄存器地址都是基于该设备物理首地址的偏移地址。每个地址都是 8bit 操作，也可进行 32bit 操作（注意偏移地址）。

（例：GPIO 的物理地址为 0x60000000~0x60000007,GPIO 控制寄存器偏移地址为 0x04,即 GPIO 控制寄存器实际地址为 0x60000000+0x04=0x60000004，对该寄存器赋值的操作在程序中即为 （*((volatile char*）(0x60000004)）= 待赋值 ）

（4）本参考文档给出的例子只是内部人员测试时使用，部分例程在不同环境中可能会有差异，实际需求还需软件开发人员根据设备说明自行编程和使用。

（5）本参考文档基于各个设备的技术文档，在部分细节说明会有简略，同时本文档不涉及外设的具体介绍以及相关时序说明。本文档目的是为软件开发人员对整个处理器外设资源有一个认识和理解，帮助其迅速上手。

设备目录

1. GPIO Driver

1.1. GPIO 介绍

WH 处理器总共含有 4 组 GPIO，分别为 GPIO1,GPIO2,GPIO3,GPIO4。每一组 GPIO 都含有 8 个管脚，即 pin0~pin7。

1.2. GPIO 特性

（1）每一组 GPIO 的输入、输出状态配置由 GPIO 功能寄存器的相应位去控制，数据由数据寄存器去配置。

（2）在软件上可以自由配置 GPIO 寄存器，从而改变输入、输出类型。

1.3. GPIO 寄存器定义

（1）WH 处理器的 GPIO 的功能状态总共由 2 种类型寄存器去控制，分别是 GPIO 控制寄存器与数据寄存器。

（1）GPIO 相关寄存器的操作对于用户来说很简单，就是找到相关寄存器地址并配置的过程，即可实现想要的功能。

（3）考虑到用户操作的方便性，以下相关的寄存器操作都是 1Btye（8bit）。

（4）GPIO 的物理地址范围为 0x60000000~0x60000007。

GPIO 控制寄存器（0x04）

用户 GPIO 控制寄存器调用格式	赋值	描述
GPIO_REG(GPIO_CTRL_OFFSET)	0x01	GPIO1 配置为输出，其余 GPIO 为输入
GPIO_REG(GPIO_CTRL_OFFSET)	0x02	GPIO2 配置为输出，其余 GPIO 为输入
GPIO_REG(GPIO_CTRL_OFFSET)	0x04	GPIO3 配置为输出，其余 GPIO 为输入
GPIO_REG(GPIO_CTRL_OFFSET)	0x08	GPIO4 配置为输出，其余 GPIO 为输入

说明：GPIO 控制寄存器（8bit 操作），低 4 位从低到高分别控制 GPIO1~GPIO4 的输入、输出状态，设置为 1 代表相应 GPIO 端口配置为输出，配置为 0 代表相应 GPIO 端口配置为输入。

GPIO 数据寄存器(0x0~0x3，对应 GPIO1~GPIO4)

用户 GPIO 数据寄存器调用格式	赋值	描述
GPIO_REG(GPIO1)	0x07	GPIO1 端口数据配置为 0x07
GPIO_REG(GPIO2)	0x07	GPIO2 端口数据配置为 0x07
GPIO_REG(GPIO3)	0x07	GPIO3 端口数据配置为 0x07
GPIO_REG(GPIO4)	0x07	GPIO4 端口数据配置为 0x07

说明：GPIO 数据寄存器（8bit 操作），赋值的数据（8 位）在相应 GPIO 组被配置为输出时，数据会输出到 GPIO 的 8 个 pin 上，对应关系为：数据第 0 位对应 pin0，第 7 位对应 pin7。当 GPIO 组被配置为输入时，也可以直接读取该组 GPIO 上的数据。

1.4. GPIO 功能函数定义及操作

WH 处理器提供了一些函数接口供用户使用，用户可以通过寄存器控制的方式去使用 GPIO，也可以调用相关 GPIO 函数接口去实现相应功能。

以下为一些封装好的常用 GPIO 功能函数，用户可选择使用。

（1）GPIO 功能配置函数

void gpio_function(char gpio_class , char function) ;

参数 gpio_class	参数 function	说明
GPIO1 或 GPIO2 或 GPIO3 或 GPIO4	0x00 或 0x01	该函数调用一次后，表示对应 GPIO 组的输入输出状态，fuction 为 0x00 表示输入，0x01 表示输出。

（2）GPIO 数据配置函数

void gpio_write_byte(char gpio_class , char value) ;

参数 gpio_class	参数 value	说明
GPIO1 或 GPIO2 或 GPIO3 或 GPIO4	例：0x73	该函数调用一次后，将对相应的 GPIO 组进行数据赋值（该 GPIO 组被配置为输出），数据第 0 位对应 pin0，数据第 7 位对应 pin7。

（3）GPIO 数据读取函数

char gpio_read_byte(char gpio_class) ;

参数 gpio_class	说明
GPIO1 或 GPIO2 或 GPIO3 或 GPIO4	该函数调用一次后，将会读取对应 GPIO 组上的 pin0~pin7 的数据（该 GPIO 组被配置为输入），数据第 0 位对应 pin0，数据第 7 位对应 pin7。

1.5. GPIO 函数简单调用例程

（1）GPIO 端口数据输出

```
void main ()
 {
```

```
    gpio_function( GPIO1，0x01 ) ;      //配置 GPIO1 为输出
    gpio_write_byte(GPIO1 , 0x08);      //配置 GPIO1 的输出值为 0x08
}
```

（2）GPIO 端口数据读取

```
void main ()
{
    char data_back ;
    gpio_function( GPIO1，0x00 ) ;          //配置 GPIO1 为输入

    data_back = gpio_read_byte(GPIO1)  ;    //读取 GPIO1 组数据
}
```

2. SPI Driver

2.1. SPI 介绍

WH 处理器含有 2 路 SPI 通道，WH 处理器作为主设备（Master）来控制从设备（slave）。其中一路 SPI 在 LS 开发板上连接在了 OLED 控制上，另一路连接在了 SD 控制上。

2.2. SPI 特性

（1）spi 的接口模式、spi 的通道选择、spi 数据发送等都可以通过寄存器配置来实现相应功能。
（2）软件上可以自由配置 SPI 相关的控制、数据、状态寄存器，从而使 SPI 工作在想要的工作条件下。

2.3. SPI 寄存器定义

（1）WH 处理器中，SPI 相关的寄存器都是 8 位操作，主要通过控制它的控制、状态以及数据寄存器来达到合适的工作状态。
（2）SPI 相关的寄存器在这里都是 1Byte(8bit)操作。
（3）SPI 的物理地址范围为 0x60000018~0x6000001b。

SPI 控制寄存器：SPI_CR(0x00)

SPI_CR 寄存器	说明
SPI_CR[7:4]	控制寄存器高 4 位设置 SPI 的数据位数，0x0~0xf 对应数据位宽为 8~16 位。
SPI_CR[3:0]	控制寄存器低 4 位设置 SPI 总线上的频率，频率计算公式为：频率=系统频率/（(SPI_CR[3:0]+1)*2 ）。

SPI 状态寄存器：SPI_SR(0x01)

SPI_SR 寄存器	说明
SPI_SR[7]	保留
SPI_SR[6:5]	SPI 通道选择，00 为 0 通道，01 为 1 通道
SPI_SR[4]	保留
SPI_SR[3]	默认为高
SPI_SR[2]	数据传输开始标志，软件将该位置为 1 则 SPI 开始一次数据传输，传输结束硬件清 0
SPI_SR[1] (CPOL)	SPI 的工作模式，与 CPHA 共同决定
SPI_SR[0] (CPHA)	SPI 的工作模式，与 CPOL 共同决定

SPI 数据寄存器：SPI_DATA_L (0x02)、SPI_DATA_H(0x03)

数据寄存器	说明
SPI_DATA_L	SPI 发送数据低 8 位
SPI_DATA_H	SPI 发送数据高 8 位（当 SPI 传输数据为 8 位，则数据高 8 位写 0）

2.4. SPI 功能函数定义及操作

2.4.1 SPI 操作主要分 2 步，SPI 的初始化以及数据发送或数据接收。用户可以直接通过赋值各个寄存器来完成 SPI 的控制，控制格式为：

控制寄存器：SPI_REG(SPI_CR)=value

状态寄存器：SPI_REG(SPI_SR)=value

数据寄存器：SPI_REG(SPI_DATA_L)=value、
SPI_REG(SPI_DATA_H)=value

各个寄存器 value 的值请根据对应寄存器各位的说明进行选择。

2.4.2 为了方便用户基于我们 WH 处理器进行 2 次开发，我们提供了几个函数接口供用户使用，用户只需填入几个参数即可完成 SPI 的基本控制。

（1）SPI 初始化函数

void spi_init(unsigned char cr , unsigned char sr);

参数	说明
cr	SPI 控制寄存器填入值
sr	SPI 状态寄存器填入值

（2）SPI 数据发送函数

void spi_send_byte(unsigned char datah ,unsigned char datal);

参数	说明
datah	发送数据高 8 位（若发送数据为 8 位，则该值赋为 0x00）
datal	发送数据低 8 位

（3）SPI 数据接收函数

short int spi_receive_data() ;

说明：当程序中调用该函数时,会得到一个函数返回值（16bit），高 8 位存放 datah 的数据，低 8 位存放 datal 的数据。

2.5. SPI 函数简单调用例程

（1）SPI 发送数据

```
void main()
 {
    //配置 SPI 为 8 位数据，0 通道，频率=系统频率/(( 2+1)*2)
    spi_init(0x02 ,0x08);
    //发送 8 位数据 0x10
    spi_send_byte(0x00,0x10);
    //发送 8 位数据 0x22
    spi_send_byte(0x00,0x22);
 }
```

（2）SPI 接收数据

```
void main()
 {
    short   int   back_data;
    //配置 SPI 为 8 位数据，0 通道，频率=系统频率/(( 2+1)*2)
    spi_init(0x02 ,0x08);
              .
              .
              .
    //接收 SPI 数据
    back_data= spi_receive_data() ;
}
```

3. UART Driver

3.1. UART 介绍

WH 处理器有 2 路 UART 串口,在 LS 开发板上,一路 UART 串口被连接在了 FT2232 芯片上，还有一路 UART 暂时被连在了 LS 扩展板的引脚上。

3.2. UART 特性

（1）WH 处理器上的 UART 在软件层上只需通过简单的 UART 寄存器配置，即可实现相应收发功能;
（2）可以通过寄存器配置来设置 UART 的波特率，同时我们 UART 校验位是偶校验位;
（3）WH 处理器的 2 路 UART 的物理地址分别为 0x60000010~0x60000017、0x600000c0~0x600000c7。
（4）WH 处理器的 UART0 和 UART1 的中断号分别为 1、2。

3.3. UART 寄存器定义

（1）WH 处理器 UART 寄存器总共分为中断状态寄存器（ISR)、中断允许寄存器(IER)、控制寄存器(CR)、数据寄存器(DATA)、波特率参数低位寄存器(BRPL)、波特率参数高位寄存器（BRPH)。
（2）UART 每一个寄存器建议都是 8bit 操作。

UART 中断状态寄存器:ISR（0x00)

寄存器各位	操作	说明
ISR[7]	R/W	ETU 计数溢出标志（ECT0) 当使能打开时，若 etu 计数器的值等于 ECR 设定的值，该位即为 1，写 0 清除该标志，写 1 没有影响。
ISR[6]	R/W	FIF0 非空标志(FIFO_NE) 为 0 则 FIFO 空;为 1 则 FIFO 非空，软件清除此位。
ISR[5]	R/W	FIFO 半满标志(FIFO_HF) 为 0 则 FIFO 非半满;为 1 则 FIFO 半满，软件清除此位。
ISR[4]	R/W	FIFO 全满标志(FIFO_FU) 为 0 则 FIFO 非全满;为 1 则 FIFO 全满，软件清除此位。
ISR[3]	R/W	Rx_FIFO 接收溢出错误（FIFO_OV) 为 0 则没有接收溢出错误发生；为 1 则发生了接收溢出错误。
		快速发送转接收标志（T2R) 为 0 则发送没有完成；为 1 则发送完成，软件查询到此标志后可立即设置当前工作模式为接收模式，软件清 0。

		注:在发送模式下,该位表示 T2R,接收模式下表示 FIFO_OV
ISR[2]	R	发送完成标志（TXEND） 为 0 则发送没有完成;为 1 则发送完成,硬件设置,软件请 0。
ISR[1]	R/W	发送/接收奇偶校验错误标示（TRE） 为 0 则发送/接收完成时无奇偶校验错误；为 1 则发送/接收完成时有奇偶校验错误。
ISR[0]	R/W	硬件自动重传 3 次（RETE_3） 为 0 则数据重传没有超过 3 次；为 1 则有奇偶错的数据重传次数>=3 次。

UART 中断允许寄存器:IER（0x01）

寄存器各位	操作	说明
IER[7]	R/W	ETU 计数溢出中断 为 0 时禁止；为 1 时使能。
IER[6]	R/W	FIFO 非空中断 为 0 时禁止；为 1 时使能。
IER[5]	R/W	FIFO 半满中断 为 0 时禁止；为 1 时使能。
IER[4]	R/W	FIFO 全满中断 为 0 时禁止；为 1 时使能。
IER[3]	R/W	FIFO_OV/T2R 中断 为 0 时禁止；为 1 时使能。
IER[2]	R/W	发送完成中断 为 0 时禁止；为 1 时使能。
IER[1]	R/W	发送/接收奇偶校验错误中断 为 0 时禁止；为 1 时使能。
IER[0]	R/W	硬件自动重传超过 3 次中断 为 0 时禁止；为 1 时使能。

UART 控制寄存器：CR（0x02）

寄存器各位	操作	说明
保留	/	
保留	/	
保留	/	
CR[4]	R/W	接收 FIFO 清除（FLUSH） 为 0 则不清空接收 FIFO；为 1 则清空接收 FIFO。
CR[3]	R/W	发送/接收模式选择（TRS） 为 0 选择接收模式；为 1 选择发送模式。
CR[2]	R/W	硬件自动重传 3 次使能 为 0 软件重发；为 1 硬件重发，发送前设置此位，如出现奇偶校验错误，由硬件自动重发 3 次。

CR[1]	R/W	奇偶校验方式选择 为 0 偶校验；为 1 奇校验。
CR[0]	R/W	正反向模式选择（DIS） 为 0 正向模式，发送数据 LSB 先发，接收数据 LSB 先收； 为 1 反向模式，发送数据 MSB 先发，接收数据 MSB 先收。

UART 数据寄存器：DATA（0x03）

寄存器	操作	说明
DATA[7:0]	R/W	接收、发送数据寄存器

波特率参数低位寄存器：BRPL（0x04）

寄存器	操作	说明
BRPL[7:0]	R/W	波特率参数低位寄存器 BRPL 与 BRPH 构成 12 位分频器。 例：系统时钟 48MHZ，为获得 115200 波特率，则 BPR=48*1000000/115200=417，则 BRPH=0x01, BRPL=0xa1。

波特率参数高位寄存器：BRPH（0x05）

寄存器	操作	说明
保留	/	
BPRH[3:0]	R/W	波特率参数高位寄存器 与 BPRL 配合使用。 注：因为总共是 12 位分频器，最大分频值为 BPRH=0x0f，BPRL=0xff，所以当系统时钟为 48Mhz 时，能分频得到的最低波特率为 48*1000000/4095=11722,因此在这种情况下得不到比它更低的波特率。

3.4. UART 功能函数定义及操作

3.3.1 UART 操作主要分 2 步，初始化以及配置收/发模式。不管什么操作，都是通过配置寄存器实现。

3.3.2 初始化内容主要为：配置 UART 波特率，配置 UART 中断使能，配置奇偶校验。

3.3.3 用户可以自己通过寻址方式进行配置，也可以调用下述函数快速实现 UART 收发功能。

（1）UART 初始化函数

void uart_init(int baud_rate);

参数	说明
baud_rate	波特率参数，若系统时钟为 48MHZ，则支持输入 11722 以上的波特率。 函数内部默认中断关闭，偶校验。

（2）UART 字符发送函数

void uart_putchar(char ch);

参数	说明
ch	该函数为 UART 字符发送函数，每调用一次发送一个 8 位数据，ch 为待发送数据。

（3）UART 字符串发送函数

void uart_puts(const char *str);

参数	说明
*str	该函数将待发送的字符串数据进行发送，*str 为待发送的字符串数据。

（4）UART 数据接收函数

char uart_getchar(void);

参数	说明
/	调用该函数，会接收外部通过串口传来的数据，每调用一次会返回最先收到的数据，并依次往后推。

3.5. UART 函数简单调用例程

（1）UART 串口发送数据

```
void main()
{
    //UART 初始化，波特率为 115200
    uart_init(115200);
```

```
    //UART 发送字符串“uctechip”
    uart_puts(“uctechip”);
    //UART 发送数据 0x11
    uart_putchar(0x11);
}
```

(2) UART 串口接收数据

```
void main()
{
    volatile  char  back;
    //UART 初始化，波特率为 115200
    uart_init(115200);

    while(1)
    {   //接收外部串口传来的数据
        back=uart_getchar();

        if(back == 0x01)
            uart_puts(“uctechip”);
        else if(back == 0x02)
            uart_putchar(0x12);
    }
}
```

4. I2C Driver

4.1. I2C 介绍

WH 处理器有 1 路 I2C，通过配置各个寄存器从而达到理想的工作模式。在 LS 开发板上，I2C 被连在了 ADC 模块上，控制 ADC 的工作模式。

4.2. I2C 特性

WH 处理器上的 I2C 外设支持以下特性：

（1） 可配置的 8 位时钟分频系数；
（2） 主模式向从模式切换
（3） 可选应答位、可编程从设备地址、中断请求
（4） 支持多字节传输

4.3. I2C 寄存器定义

（1）WH 处理器上的 I2C 寄存器都是 8bit 操作，通过配置寄存器从而使 I2C 工作在合适的模式。

（2）WH 处理器总共有 5 个可访问寄存器，分别为地址寄存器、预分频寄存器、控制寄存器、状态寄存器、数据寄存器。

（3）WH 处理器的 I2C 物理地址范围为 0x60000060~0x60000067。

（4）WH 处理器的 I2C 的中断号为 3。

I2C 地址寄存器：ADDR（0x03），复位值（0x00）

寄存器	操作	说明
ADDR[7:1]	R/W	从设备地址。该地址将和 I2C 发出的从设备地址进行匹配。 注：该寄存器保存着 I2C 模块作为从设备时的地址，即会对该地址呼叫进行应答。I2C 作为主设备时，不应该呼叫自己的从设备地址。
保留	R	

I2C 预分频寄存器：FDR（0x04），复位值（0x00）

寄存器	操作	说明
FDR[7:0]	R/W	预分频系数，实际分频系数为（FDR+1）*4 注：I2C 总线传输速率最快可达 400kb/s,该值应根据系统时钟计算。

I2C 控制寄存器：CR（0x05）,复位值（0x00）

寄存器	操作	说明
保留	R	该寄存器第 7 位保留
CR[6]	R/W	中断使能位（MIEN） 为 0 中断不使能；为 1 中断使能。如果 SR 中的 MINT 被设置，则产生中断请求。
CR[5]	W	重复开始位（RS） 为 0 则不产生重复开始条件；为 1 则产生。
CR[4]	R/W	应答位/STOP 条件位（TACK） 为 0 则在接收 1 字节数据后，在应答周期产生应答，即拉低 SDA 线； 为 1 即接收一字节数据后，不产生应答；或者主设备发完当前字节后产生一个 STOP 条件。
CR[3]	R/W	模式选择（MTX） 为 0 则为接收模式；为 1 则为发送模式。
CR[2]	R/W	主/从模式选择位，START 位。（MSTA） 为 0 则为从模式；为 1 则为主模式，当该位从 0 跳变为 1，模块产生一个 START 条件。 当 STOP 条件产生或者模块作为主设备失去仲裁，则该位清 0。软件置 1。
保留	R	该寄存器第 1 位保留
CR[0]	R/W	模块使能位（MEN） 为 0 则模块不使能，为 1 则模块使能。

I2C 状态寄存器：SR（0x06）,复位值（0x02）

寄存器	操作	说明
SR[7]	R	数据传输完成标志位（MTF） 为 0 则为传输过程；为 1 则传输完成。 接收模式下，读数据寄存器清除该位；发送模式下，写数据寄存器清除该位。
SR[6]	R	从地址匹配位（MAAS） 为 0 则代表从地址不匹配;为 1 代表从地址匹配。 当 ADDR 中的地址和总线呼叫地址相匹配时，字节传输完成时，该位被设置。写 CR 将清除该位。
SR[5]	R	总线状态位 为 0 表示 I2C 总线空闲；为 1 表示总线忙。 当 START 条件被检测到，该位设置，STOP 条件检测到，该位清 0。
SR[4]	R/W	仲裁丢失状态位 为 0 则仲裁没有丢失；为 1 则仲裁丢失。

		当发生仲裁丢失时，该位被自动设置。
SR[3]	R	从设备读/写状态位（SRW） 当 MAAS 被设置后，该位在如下条件有效：一个完整的传输发生，没其他传输被初始化；且 I2C 被配置为从模式，从地址匹配。当接收到 STOP 或 START 条件，该位清除。
SR[2]	R/W	中断状态位（MINT） 为 0 没有中断；为 1 有中断。当 MIEN 被设置，将输出中断请求。 该位写 0 清除中断，在中断使能打开的情况下，下述条件下会产生中断： （1）一字节数据被传输（第 9 个 SCL 下降沿产生中断）； （2）从地址匹配（第 9 个 SCL 下降沿产生中断）； （3）仲裁丢失；
SR[1]	R	应答接收状态位（RACK） 为 0 则最近的发送周期接收到应答；为 1 则没有接收到应答。START 条件或者 STOP 条件将清除该位。
保留	/	SR 第 0 位保留

I2C 数据寄存器：DR（0x07），复位值（0x00）

寄存器	操作	说明
DR[7:0]	R/W	8bit 数据，在 I2C 总线上，DATA 数据高位先发送，低位数据最后发送。发送模式下保存待发送的数据，接收模式下保存最近接收到的数据；在从接收模式下，若上个数据未被读走，下个数据已准备好，则拉低 CLK 来等待数据读取。

4.4. I2C 功能函数定义及操作

I2C 操作主要分为初始化、发送、接收 3 部分。
WH 处理器中 I2C 模块的操作可以简单划分为以下几部分：

4.4.1 初始化

在系统上电后，当要使用 I2C 模块时，首先需要使能 I2C 模块（MEN），在模块使能打开后，设置 I2C 的地址寄存器和预分频寄存器。

4.4.2 I2C 主模式写操作

首先向数据寄存器写一个数据（地址字节），其次设置 MIEN=1（打开中断使能）、TACK=0、MTX=1(设置为发送模式)、MSTA=1（该位由 0 变为 1 即产生一个 START 条件），从而开始一次地址呼叫。当 I2C 模块完成一次字节传输后，会在第 9 个周期下降沿发出中断；软件接收到中断后，向数据寄存器写一个数据（数据字节，清除 MTF）；如果该数据写完成后不继续写，应该设置 TACK=1（产生一个 STOP 信号）。然后向状态寄存器写零清除中断。

4.4.3 I2C 主模式读操作

首先向数据寄存器写一个数据（地址字节），其次设置 MIEN=1（打开中断使能）、TACK=0、MTX=1(设置为发送模式)、MSTA=1（该位由 0 变为 1 即产生一个 START 条件），从而开始一次地址呼叫。当 I2C 完成地址呼叫后，会在第 9 个周期下降沿发出中断；软件接收到中断之后，向数据寄存器写一次数据（为了清除 MTF），然后设置 MTX=0（接收模式），清除中断。该次读完成后不继续读，应设置 TACK=1（产生 STOP 条件），然后写零清除中断，否则只清除中断，继续读下一个数据（产生中断表示数据接收完成）。

注：函数调用例子待后续补充，用户可根据上述信息使用 I2C。

5. ETIMER Driver

5.1. ETIMER 介绍

WH 处理器总共有 2 组 ETIMER 模块，其中一组作为 PWM 输出连在了 LS 开发板的蜂鸣器模块上。

5.2. ETIMER 特性

WH 处理器上的 PWM 共有以下特性：

（1）共有 5 种工作模式、4 种中断，可通过编程自由改变。

（2）根据寄存器模式配置，可有定时器模式、计时器模式、计数器模式、PWM 输出模式以及 PWM 测量模式。

5.3. ETIMER 寄存器定义

（1）WH 处理器的 ETIMER 模块共有 5 种类型寄存器，分别为控制寄存器（EtimerCR）、中断状态寄存器（EtimerINTSRC）、计数次数控制寄存器（CntCR）、PWM 控制寄存器（PwmCR）、测量方波次数数据寄存器（TestDR）。

（2）每一种寄存器的操作方式都为 32bit,当然可根据每种寄存器功能在使用上再进行细分为 8bit 操作（需注意偏移地址）。

（3）WH 处理器的 2 路 ETIMER 寄存器地址范围分别为：0x60000080~0x6000009f,0x600000a0~0x600000bf。

（4）WH 处理器的 ETIMER0 和 ETIMER1 的中断号分别为 6、7。

控制寄存器：EtimerCR（0x00）,复位值（0x00000000）

寄存器	操作	说明
EtimerCR[31:24]	R	CntNum[15:8] 计数器的数值高 8 位。
EtimerCR[23:16]	R	CntNum[7:0] 计数器的数值低 8 位。
EtimerCR[15:8]	R/W	保留。
EtimerCR[7]	R	Etimer_Busy 为 1 代表定时器工作。
EtimerCR[6]	R/W	CcinEdgeselect 为 1 代表事件的上升边沿计数。
EtimerCR[5]	R/W	CntHold 为 1 则计数器保持。
EtimerCR[4]	R/W	Reset 为 1 则进行复位。
EtimerCR[3]	R/W	Start 为 1 则计数器启动（clk 为计数来源）
EtimerCR[2:0]	R/W	EtimerMode 计数器模式选择 001：定时器模式

		010：计时器模式 011：事件计数器模式 100：pwm 输出模式 101：pwm 测量模式

中断状态寄存器：EtimerINTSRC（0x04）,复位值（0x00000000）

寄存器	操作	说明
EtimerINTSRC[31:4]	R/W	保留
EtimerINTSRC[3]	R/W	CntOvStatus CntOv 标志，写 1 清除
EtimerINTSRC[2]	R/W	PWMTimOv PWMTimOv 标志，写 1 清除
EtimerINTSRC[1]	R/W	TimSetErr TimSetErr 标志，写 1 清除
EtimerINTSRC[0]	R/W	PWMSetValErr PWMSetValErr 标志，写 1 清除

计数次数控制器：CntCR(0x08),复位值（0x00000000）

寄存器	操作	说明
CntCR[31:24]	R/W	CCValue[15:8] CCValue 的数据高 8 位
CntCR[23:16]	R/W	CCValue[7:0] CCValue 的数据低 8 位
CntCR[15:8]	R/W	Cntvalue[15:8] Cntvalue 的数据高 8 位
CntCR[7:0]	R/W	Cntvalue[7:0] Cntvalue 的数据低 8 位

PWM 控制寄存器：PwmCR（0x0c）,复位值（0x00000000）

寄存器	操作	说明
PwmCR[31:24]	R	PWMTimNum[15:8] PWM 波已产生次数的高 8 位
PwmCR[23:16]	R	PWMTimNum[7:0] PWM 波已产生次数的低 8 位
PwmCR[15:8]	R/W	PWMTimValue[15:8] 设置产生 PWM 波的次数高 8 位
PwmCR[7:0]	R/W	PWMTimValue[7:0] 设置产生 PWM 波的次数低 8 位

测量方波次数数据寄存器：TestDR（0x10）,复位值（0x00000000）

寄存器	操作	说明
TsetDR[31:24]	R	CyclNum[15:8]

		测量方波时钟周期数高 8 位
TsetDR[23:16]	R	CyclNum[7:0] 测量方波时钟周期数低 8 位
TsetDR[15:8]	R	PlusNum[15:8] 测量方波时钟脉冲数高 8 位
TsetDR[7:0]	R	PlusNum[7:0] 测量方波时钟脉冲数低 8 位

5.4. ETIMER 功能函数定义及操作

（1）ETIMER 的操作主要分为 3 步骤：配置工作模式、配置模式参数、启动设备。

（2)每个步骤基于寄存器操作,ETIMER 的配置建议 32bits 操作,当然 8bits 操作也是可以的，注意偏移地址。

ETIMER 不同模式简介

（1）定时器模式

EtimerCR[2:0]=3’b001 时，计数器工作在定时器模式下，计数来源为 clk,设置计数初值（Cntvalue）后，启动计数器，计数器由初值开始，每一个时钟来临，计数器减 1。减到 0 时，计数器恢复计数初值，产生中断溢出（CntOv），计数器停止计数。（若不设置计数初值，默认值为 33000;若设置初值为 0，则产生定时器数值设置错误中断(TimSetErr)）。

（2）计时器模式

EtimerCR[2:0]=3’b010 时，计数器工作在计时器模式下，计数来源为 clk，设置计数初值（Cntvalue）后，启动计数器，计数器由初值开始，每一个时钟来临，计数器加 1。由 0 加到满值时，产生主计数器溢出中断（CntOv），计数器停止计数。（若不设置计数初值，默认值为 33000）。

（3）事件计数器模式

EtimerCR[2:0]=3’b011 时，计数器工作在事件计数器模式下，计数来源为外部事件的下降沿或上升沿，需设置计数目标值（Cntvalue）。设置后，启动计数器，计数器由 0 开始，当触发边沿来临，计数器加 1，计数到满值时，计数器清 0，产生主计数器溢出中断（CntOv），等待触发边沿继续计数，如此循环。（若没设置目标值，默认目标值为 33000）。

（4）PWM 输出模式

EtimerCR[2:0]=3’b100 时，ETIMER 模块工作在 pwm 输出模式，计数来源为 clk，需要设置比较值（CCValue）和目标值（Cntvalue），这里目标值的时间宽度为 1 个 PWM 波的周期时间，比较值为 PWM 波低电平持续时间。设置完后，启动计数器，产生连续的 PWM 波，直至复位，停止产生。输出保持为 0。若设置 PWM 的产生次数（PWMTimValue）,产生相应次数后，

输出保持 0。(该模式可以软件复位，若设置比较值大于目标值，则会报 PWM 数值设置错误中断 PWMSetValErr)。

（5）PWM 测量模式

EtimerCR[2:0]=3’b101 时，ETIMER 模块工作在 PWM 测量模式下。输入 PWM 波，启动计数器，可以测量其周期和脉冲。计数来源是 clk。其周期和脉冲分别为：

T=CyclNum*clk

t=PlusNum*clk

（此模式若没有设置目标值，则默认目标值为 33000，若比较值大于目标值，则会报 PWM 数值设置错误的中断 PWMSetValErr)。

注：函数调用例子待后续补充，用户可根据上述信息使用 ETIMER。

6. VGA Driver

6.1. VGA 介绍

WH 处理器含有一组 VGA 模块,对外连在 LS 扩展板上的有 VGA 的 RGB888 信号、行场同步信号、像素时钟信号 clk、blank 信号。

6.2. VGA 特性

（1）WH 处理器 VGA 核基于开源 VGA 核，支持 RGB565、RGB888、灰度输出等格式,因为需要和处理器相匹配,我们对 VGA 核做了些调整,目前只支持 RGB888,分辨率为 640x480@60HZ 的标准 VGA 输出（后续会逐步完善）。

（2）WH 处理器 VGA 核支持兵乓操作，图像切换会在 2 个地址间切换（软件控制）。

（3）WH 处理器 VGA 核支持视频地址更改，即可更改 2 个视频地址存储器，从而达到向任意地址索取视频数据（地址允许情况下）。

6.3. VGA 寄存器定义

（1）完整的 VGA 核寄存器定义见《VGA 核技术文档》。在本文档中，只列出用户可操作，且需要去配置的。

（2）VGA 核内部寄存器主要有：控制寄存器、状态寄存器、水平定时寄存器、垂直定时寄存器、水平和垂直长度寄存器、视频存储基址寄存器 A、视频存储基址寄存器 B 等这 7 种寄存器（其余在 VGA 模块中不使用）。

（3）WH 处理器 VGA 核的物理地址范围为：0x60003000~0x60003ffc。

（4）WH 处理器 VGA 核寄存器操作建议使用 32bit 操作，8bit 操作也可以，需注意地址偏移量。

（5）WH 处理器 VGA 核的中断号为 9。

控制寄存器：CTRL（0x00）,复位值（0x00000000）

寄存器	操作	说明
CTRL[31:26]	R/W	保留
CTRL[25]	R/W	HC1R,硬件 Cursor1 分辨率 0:32x32 像素模式 1:64x64 像素模式
CTRL[24]	R/W	HC1E，Cursor1 使能 0：禁用硬件 Cursor1 1:硬件 Cursor1 启用
CTRL[23:22]	R/W	保留
CTRL[21]	R/W	HC1R,硬件 Cursor1 分辨率 0:32x32 像素模式 1:64x64 像素模式
CTRL[20]	R/W	HC1E，Cursor1 使能

		0：禁用硬件 Cursor1 1:硬件 Cursor1 启用
CTRL[19:16]	R/W	保留
CTRL[15]	R/W	BL，消隐信号控制 0：数据无效段为 1，数据有效段为 0 1：数据无效段为 0，数据有效段为 1
CTRL[14]	R/W	CSL，混合信号控制 0：帧、行同步段为 1，其余为 0 1：帧、行同步段为 0，其余为 1
CTRL[13]	R/W	VSL，垂直同步脉冲控制 0：同步脉冲为高电平 1：同步脉冲为低电平
CTRL[12]	R/W	HSL，水平同步脉冲控制 0：同步脉冲为高电平 1：同步脉冲为低电平
CTRL[11]	R/W	PC，8 位伪彩色 0：8 位灰度 1：8 位伪彩色
CTRL[10:9]	R/W	CD，颜色深度 11：保留 10：每像素 24 位 01：每像素 16 位 00：每像素 8 位
CTRL[8:7]	R/W	VBL，视频内存突发长度 11：8 个循环 10：4 个循环 01：2 个循环 00：1 个循环
CTRL[6]	R/W	CBSWE 0：禁用颜色查找表库切换 1：启用颜色查找表库切换
CTRL[5]	R/W	VBSWE 0：禁用视频内存库切换 1：启用视频内存库切换
CTRL[4]	R/W	CBSIE（中断使能） 0：禁用颜色查找表库切换中断使能 1：启动颜色查找表库切换中断使能
CTRL[3]	R/W	VBSIE（中断使能） 0：禁用视频存储切换中断使能 1：启用视频存储切换中断使能
CTRL[2]	R/W	HIE（中断使能） 0：禁用水平同步脉冲中断使能 1：启用水平同步脉冲中断使能

CTRL[1]	R/W	VIE（中断使能） 0：禁用垂直同步脉冲中断使能 1：启用垂直同步脉冲中断使能
CTRL[0]	R/W	VEN（视频使能） 0：禁用视频系统 1：启用视频系统

状态寄存器：STAT（0x04）,复位值（0x00000000）

寄存器	操作	说明
STAT[31:25]	R	保留
STAT[24]	R	HC1A，硬件 Cursor1 可用
STAT[23:21]	R	保留
STAT[20]	R	HC0A，硬件 Cursor0 可用
STAT[19:18]	R	保留
STAT[17]	R	ACMP，有源 CLUT 存储器页面
STAT[16]	R	AVMP，有源视频存储器页面
STAT[15:8]	R	保留
STAT[7]	R/W	CBSINT
STAT[6]	R/W	VBSINT 视频已缓存一帧数据结束状态位
STAT[5]	R/W	HINT 水平中断状态位
STAT[4]	R/W	VINT 垂直中断状态位
STAT[3:2]	R/W	保留
STAT[1]	R/W	LUINT 线路 FIFO 欠载中断挂起
STAT[0]	R/W	SINT 系统错误中断挂起

水平定时寄存器：HTIM（0x08）,复位值（0x00000000）

寄存器	操作	说明
HTIM[31:24]	R/W	Thsync 水平同步脉冲宽度
HTIM[23:16]	R/W	Thgdel 水平门延迟时间（相当于 VGA 水平前廊）
HTIM[15:0]	R/W	Thgate 水平选通时间（相当于 VGA 数据显示长度）

垂直定时寄存器：VTIM（0x0C）,复位值（0x00000000）

寄存器	操作	说明
VTIM[31:24]	R/W	Tvsync 垂直同步脉冲宽度

VTIM[23:16]	R/W	Tvgdel 垂直门延迟时间（相当于 VGA 垂直前廊）
VTIM[15:0]	R/W	Tvgate 垂直门控时间（相当于 VGA 数据显示宽度）

水平和垂直长度寄存器：HVLEN（0x10）,复位值（0x00000000）

寄存器	操作	说明
HVLEN[31:16]	R/W	Thlen 水平长度
HVLEN[15:0]	R/W	Tvlen 垂直长度

视频存储器基地址寄存器 A：VBARa（0x14）,复位值（0x00000000）

寄存器	操作	说明 ·
VBARa[31:2]	R/W	VBA 视频基地址 A
VBARa[1:0]	R	始终为 0

视频存储器基地址寄存器 B：VBARb（0x18）,复位值（0x00000000）

寄存器	操作	说明
VBARb[31:2]	R/W	VBA 视频基地址 B
VBARb[1:0]	R	始终为 0

6.4. VGA 功能函数及操作

WH 处理器的 VGA 核在使用上采用 640x480@60HZ 的分辨率。为此基于这个标准，在相关寄存器配置中，一些寄存器的配值是确定的，以下给出寄存器配置说明，针对处理器 VGA 核，以 RGB888 为例，没有提及的配置操作默认为 0 即可。

CTRL 寄存器配置：

（1）在 CTRL 寄存器中，我们需要将 BL、CSL、VSL、HSL 配置为 1 这是为了满足 VGA 模块对外输出信号 blank、csync、vsync、hsync 信号的要求（有些显示器需要该种信号配置）。

（2）将颜色深度 CD 的值配为 10（每像素 24 位，即 RGB888 输出）。

（3）对于中断使能，我们只使用 VBSIE 或 VIE。设置 VBSIE 为高时，每当有一帧数据被缓存进 VGA 核内部数据 FIFO 时，状态寄存器中的 VBSINT 会拉高，当 VBSINT 与 VBSIE 同时为高时，会产生中断；设置 VIE 为高时，每当显示一帧图像后，状态寄存器中的 VINT 会拉高，当 VIE 与 VINT 同时为高时，会产生中断。另外 VBSINT 与 VINT 与对应的中断使能无关，条件满足自动会拉高，需要软件往对应位置写 1 即可清除状态信号，从而清除中断；当然也可以关闭中

断使能信号（往对应位置写 0），从而不会产生中断。

（4）对于 VGA 核电路使能信号 VEN，在配置完控制寄存器基本参数、水平定时器、垂直定时器、水平和垂直长度寄存器、视频存储器基地址寄存器 A、视频存储器基地址寄存器 B 后，将 VEN 置为 1，电路启动，VGA 开始工作。（默认循环从视频存储器地址 A 中的地址开始读取一帧的数据）。

STAT 寄存器使用：

在 STAT 寄存器中，对于用户来说，只需关注 VINT、VBSINT 这 2 个状态信号即可。VINT 代表当前显示器显示完成一帧图像，VBSINT 代表已有一帧完整数据被写入 VGA 核内部缓存 FIFO。VBSINT 与 VINT 都需要软件向 STAT 寄存器相应位写 1 清除。

HTIM、VTIM、HVLEN 寄存器使用：

这 3 种寄存器是对 VGA 显示分辨率的控制，需要结合具体的像素时钟来配置。目前 VGA 输出分辨率为 640x480@60HZ，所以根据该标准 VGA 分辨率（同步段、前廊、数据有效段、后廊）来配置者 3 这种寄存器，得到的结果如下：

HTIM: Thsync=0x5f; Thgdel=0x2f; Thgate=0x27f;

VTIM: Tvsync=0x1; Tvgdel=0x20; Tvgate=0x1df;

HVLEN: Thlen=0x31f; Tvlen=20c;

VBARa、VBARb 寄存器使用：

VBARa 和 VBARb 寄存器分别保存着 2 个数据来源的起始地址，默认情况下（不执行地址寄存器切换操作），VGA 核会一直循环读取 VBARa 寄存器里的值作为图像数据的内存访问首地址，并以该地址开始，读取 640x480x3 的字节数据量（VGA 分辨率为 640x480，每个像素 3 字节）。

frame_buffer 操作：

WH 处理器内部的 VGA 核，对于 frame_buffer 操作有 2 种实现方式。

1、单寄存器 A 地址切换操作

VGA 核内部有 2 个视频基地址存储器，在不切换视频地址寄存器 A、B 的情况下，默认会一直从 A 地址开始读取数据。

（1）在视频电路 VEN 置 1 启动后。软件向视频地址寄存器 A 写入一个新地址（下一帧数据来源地址，如果不想更换下一帧数据地址，则可省略该步操作或者向寄存器写同样的地址，视频的第一帧数据为在 VEN 信号置为 1 前，地址寄存器 A 最后确定的值）。

（2）之后每当检测到状态寄存器 STAT 中的 VBSINT 信号为 1 时，再往地址寄存器 A 写入下一帧所需的地址，然后再往 STAT 中的 VBSINT 信号位写 1 清除该信号。

2、地址寄存器 A、B 切换操作

该种方式与上一种的区别在于，这种方式用到了视频地址寄存器 A 和视频地址寄存器 B，切换是在这 2 者之间。操作方式如下：

（1）在视频电路 VEN 置 1 启动后。软件向 CTRL 寄存器的 VBSWE 位写 1（跳转

至视频地址寄存器 B)，切换下一帧图像的数据来源地址（第一帧数据来源默认来自地址寄存器 A 的值)。若不想切换下一帧的数据来源，则无需此操作。(2)之后每当检测到状态寄存器 STAT 中的 VBSINT 信号为 1 时，如果需要切换视频地址寄存器 A 或 B,则往 VBSWE 写 1(软件置 1,硬件清 0),然后再向 VBSINT 写 1 清除该状态位即可。

注：这里需明确一个概念，VGA 核在 VEN 置为 1 启动后，图像的第一帧数据起始地址来自视频地址寄存器 A 的值，对于图像第 2 帧的数据来源操作，在上述 2 种操作的第 1 步中已写明。对于图像的第 3 帧数据来源操作见上述 2 种操作的第 2 步，也就是说检测到第 1 个 VBSINT 信号时，此时是对图像的第 3 帧数据来源进行 0 操作，第 2 次检测到 VBSINT 信号时，此时是对图像的第 4 帧数据来源进行操作，以此类推。

此外用户也可以结合上述 2 种操作来进行 frame_buffer 操作，具体请根据自身需要来操作。

VGA 视频数据存储格式操作说明:

WH 处理器对于图像数据的存储目前建议存储在 DDR3 中，但是每一个像素点的数据（含 3 个字节，RGB888）对于连续的存储地址来说，其需要按照一定的格式进行放置，软件编程时需要注意这点。

根据 VGA 控制寄存器（CTRL）中 CD（颜色深度）位的操作，目前在我们的 VGA 核中，只允许配置为 10（RGB888，每像素 24bit)，在像素数据写入过程中（软件寻址)，:以下将举例说明：

1：假设视频数据的初始寻址地址为 0x0,由于处理器每读取一次数据是 32bit,所以在对这 32bit 数据处理中，VGA 核的操作是将读取的第一个 4byte 数据中的高 3byte 数据最为 RGB 数据，R 为最高地址的 byte 数据，最低 byte 数据，VGA 核会将它作为第 2 个像素的 R 数据。(即，地址 0x3,0x2,0x1 分别作为第一个像素的 R1、G1、G1 数据，地址 0x00 作为 R2 数据)。

2：第 2 次读取数据的地址会从 0x4 开始的 4 个 byte，在这里，VGA 会将地址 0x7、0x6 的数据分别作为 G2、B2 数据，0x5,0x4 的数据分别作为 R3、G3 的数据。

3：第 3 次读取数据操作会从地址 0x8 开始的 4 个 byte，在这里,VGA 会将地址 0xb 的数据作为 B3 的数据，0xa、0x9、0x8 的数据分别作为 R4、G4、B4 的数据。

4：之后以步骤 1 的操作继续进行循环操作，从 0xC 开始循环上述操作。因此在软件上对地址赋值的操作需要按上述进行。

7. CAN Driver

7.1. CAN 介绍

WH 处理器含有 1 组 CAN 模块，CAN 模块对外连在了 LS 扩展板的 can 的收发控制器上。

7.2. CAN 特性

WH 处理器 CAN 模块包含 can 设备所需的所有功能。其具有如下特性：

（1）可配置的波特率设置；

（2）可配置的单双滤波配置；

（3）模式切换；

（4）中断相应；

7.3. CAN 寄存器定义

（1）WH 处理器内部 CAN 模块总共有 32 个寄存器，每个寄存器 8 位，不同模式下，这些寄存器表达的含义并不完全相同。

（2）用户需要根据当前所处的模式去配置相应的寄存器，从而正常使用 can 设备。

(3)WH 处理器内部的 CAN 模块在功能上共有 4 种模式,分别为 basic_mode、basic_reset_mode、extend_mode、extend_reset_mode。

（4）不同模式寄存器操作会有所不同，为考虑到用户更便携使用 CAN 模块。因此我们建议用户只考虑使用 extend_mode 和 extend_reset_mode。之后给出的寄存器说明和配置也只围绕这 2 种模式。

（5）WH 处理器 CAN 模块内部寄存器地址范围为 0x60000040~0x6000005f。

（6）WH 处理器 CAN 设备的中断号为 5。

模式：Extend_Mode

Addr	R/W	Initial	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	note
0x00	R						acceptance_filter_mode 0:双滤 1:单滤	self_test_mode	listen_only_mode	mode 0:rst 1: normal	0:无效 1:有效
	W									mode 1:rst 0:	

										normal	
0x01	R										tx_request= bit0 | bit4 ; self_rx_request= bit4 & ~ bit0 ; 0:无效 1:有效
	W					bit4	clear_data_overrun	Release_buffer	abort_rx	bit0	
0x02	R		node_buff_off	error_status	transmit_status	receive_status	transmission_complete_reg	transmit_buffer_status_reg	overrun_status_reg	receive_buffer_status_reg	0:无效 1:有效
	W										
0x03	R		bus_error_irq	arbitration_lost_irq	error_passive_irq	0	data_overrun_irq	error_irq	transmit_irq	receive_irq	0:无效 1:有效
	W										
0x04	R		bus_error_irq_en	arbitration_lost_irq_en	error_passive_irq_en		data_overrun_irq_en_ext	error_warning_irq_en_ext	transmit_irq_en_ext	receive_irq_en_ext	0:无效 1:有效
	W		bus_error_irq_en	arbitration_lost_irq_en	error_passive_irq_en		data_overrun_irq_en_ext	error_warning_irq_en_ext	transmit_irq_en_ext	receive_irq_en_ext	
0x06	R		bus_timing_0[7:0]								bus_timing_0[7:6] 同步跳转宽度 bus_timing_0[5:0] 分频系数
	W										
0x07	R		bus_timing_1[7:0]								bus_timing_1[7] 采样数 bus_timing_1[6:4] 传播段 2（seg2） bus_timing_1[3:0] 传播段 1（seg1）
	W										
0x0b	R					arbitration_lost_capture [4:0]					仲裁丢失捕获寄存器
	W										

0x0c	R		error_capture_code_tye[7:6] 00:bit_err 01:form_err 10:stuff_err 11:crc_err,ack_err		error_capture_code_direction	error_capture_code_segment[4:0]					error_capture_code_segment 捕获错误段
	W										
0x0d	R		error_warning_limit[7:0]								错误警告上限，扩展模式下，当接收错误计数或者发送错误计数大于该值，就会进入错误状态
	W										
0x0e	R		rx_err_cnt[7:0]								接收错误计数器
	W										
0x0f	R		tx_err_cnt[7:0]								发送错误计数器
	W										
0x10	R		rx_data_0[7:0]								接收数据计数器 0
	W		tx_data_0[7:0]								发送数据计数器 0
0x11	R		rx_data_1[7:0]								接收数据计数器 1
	W		tx_data_1[7:0]								发送数据计数器 1
0x12	R		rx_data_2[7:0]								接收数据计数器 2
	W		tx_data_2[7:0]								发送数据计数器 2
0x13	R		rx_data_3[7:0]								接收数据计数器 3
	W		tx_data_3[7:0]								发送数据计数器 3
0x14	R		rx_data_4[7:0]								接收数据计数器 4
	W		tx_data_4[7:0]								发送数据计数器 4
0x15	R		rx_data_5[7:0]								接收数据计数器 5

	W		tx_data_5[7:0]							发送数据计数器 5
0x16	R		rx_data_6[7:0]							接收数据计数器 6
	W		tx_data_6[7:0]							发送数据计数器 6
0x17	R		rx_data_7[7:0]							接收数据计数器 7
	W		tx_data_7[7:0]							发送数据计数器 7
0x18	R		rx_data_8[7:0]							接收数据计数器 8
	W		tx_data_8[7:0]							发送数据计数器 8
0x19	R		rx_data_9[7:0]							接收数据计数器 9
	W		tx_data_9[7:0]							发送数据计数器 9
0x1a	R		rx_data_10[7:0]							接收数据计数器 10
	W		tx_data_10[7:0]							发送数据计数器 10
0x1b	R		rx_data_11[7:0]							接收数据计数器 11
	W		tx_data_11[7:0]							发送数据计数器 11
0x1c	R		rx_data_12[7:0]							接收数据计数器 12
	W		tx_data_12[7:0]							发送数据计数器 12
0x1d	R		rx_message_counter[7:0]							接收信息计数
	W									
0x1f	R		extended_mode 1:extend 0:basic							extended_mode 只读

模式：Extend_Reset_Mode

Addr	R/W	Initial	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	note
0x00	R						acceptance_filter_mode 0:双滤 1:单滤	self_test_mode	listen_only_mode	mode 1:rst 0: normal	0:无效 1:有效
	W						acceptance_filter_mode 0:双滤 1:单滤	self_test_mode	listen_only_mode	mode 0:rst 1: normal	
0x01	R										
	W										
0x02	R		node_buff_off	error_status	transmit_status	receive_status	transmission_complete_reg	transmit_buffer_status_reg	overrun_status_reg	receive_buffer_status_reg	0:无效 1:有效
	W										
0x03	R		bus_error_irq	arbitration_lost_irq	error_passive_irq	0	data_overrun_irq	error_irq	transmit_irq	receive_irq	0:无效 1:有效
	W										
0x04	R		bus_error_irq_en	arbitration_lost_irq_en	error_passive_irq_en		data_overrun_irq_en_ext	error_warning_irq_en_ext	transmit_irq_en_ext	receive_irq_en_ext	0:无效 1:有效
	W		bus_error_irq_en	arbitration_lost_irq_en	error_passive_irq_en		data_overrun_irq_en_ext	error_warning_irq_en_ext	transmit_irq_en_ext	receive_irq_en_ext	
0x06	R		bus_timing_0[7:0]								bus_timing_0[7:6] 同步跳转宽度 bus_timing_0[5:0] 分频系数
	W		bus_timing_0[7:0]								

0x07	R		bus_timing_1[7:0]	bus_timing_1[7] 采样数 bus_timing_1[6:4] 传播段 2（seg2） bus_timing_1[3:0] 传播段 1（seg）
	W		bus_timing_1[7:0]	
0x06	R			写入 bus_timing_0
	W		bus_timing_1[7:0]	
0x0b	R		arbitration_lost_capture[4:0]	仲裁丢失捕获寄存器
	W			
0x0c	R		error_capture_code_tye[7:6] 00:bit_err 01:form_err 10:stuff_err 11:crc_err,ack_err / error_capture_code_direction / error_capture_code_segment[4:0]	error_capture_code_segment 捕获错误段
	W			
0x0d	R		error_warning_limit[7:0]	错误警告上限，扩展模式下，当接收错误计数或者发送错误计数大于该值，就会进入错误状态
	W		error_warning_limit[7:0]	
0x0e	R		rx_err_cnt[7:0]	接收错误计数器
	W		rx_err_cnt[7:0]	
0x0f	R		tx_err_cnt[7:0]	发送错误计数器
	W		tx_err_cnt[7:0]	
0x10	R		acceptance_code_0[7:0]	接收标识符寄存器 0
	W		acceptance_code_0[7:0]	
0x11	R		acceptance_code_1[7:0]	接收标识符寄存器 1
	W		acceptance_code_1[7:0]	

0x12	R		acceptance_code_2[7:0]							接收标识符寄存器 2
	W		acceptance_code_2[7:0]							
0x13	R		acceptance_code_3[7:0]							接收标识符寄存器 3
	W		acceptance_code_3[7:0]							
0x14	R		acceptance_mask_0[7:0]							接收掩码寄存器 0
	W		acceptance_mask_0[7:0]							
0x15	R		acceptance_mask_1[7:0]							接收掩码寄存器 1
	W		acceptance_mask_1[7:0]							
0x16	R		acceptance_mask_2[7:0]							接收掩码寄存器 2
	W		acceptance_mask_2[7:0]							
0x17	R		acceptance_mask_3[7:0]							接收掩码寄存器 3
	W		acceptance_mask_3[7:0]							
0x1d	R		rx_message_counter[7:0]							接收信息计数
	W									
0x1f	R		extended_ mode 1:extend 0:basic							extended_mode 只读

7.4. CAN 功能函数及操作

WH 处理器的 CAN 模块操作，用户在使用中，只需关注 Extend_mode 和 Extend_reset_mode 的操作即可，其中 Extend_reset_mode 就是 Extend_mode 的复位模式。

CAN 设备

8. OLED Driver

8.1. OLED 介绍

WH 处理器内部第 1 路 SPI 在 LS 扩展板上，连在了 OLED 上，通过 SPI 控制 OLED 的工作模式及点亮。因为 OLED 属于 LS 扩展板上的设备，不属于 WH 处理器内部模块，所以不含有 WH 处理器内部寄存器操作。关于 OLED 的具体介绍请参考《OLED-SSD1309Revision11》,这里只介绍其使用方法，帮助用户快速上手。

8.2. OLED 特性

LS 扩展板上的 OLED 型号为 SSD1309，其部分特性如下：

（1）具有 128x64 分辨率的点阵面板；

（2）本 OLED 显示颜色为绿色；

（3）工作温度范围为：-40℃至 80℃；

（4）可编程模式；

8.3. OLED 用户使用说明