拉普兰德 K60 底层库——OSKinetis

函数手册

库版本 2.2

底层库更新记录

版本	描述	日期
0.0	● 创建文档“LPLD_K60 底层开发包”	2012/2/16
0.1	● 底层驱动包添加 PIT 模块相关函数，并更新相应说明	2012/2/22
0.2	● 文档更名为“K60 底层开源驱动-LPLD_OSKinetis” ● 底层驱动包添加 eDMA 模块相关函数，并更新相应说明 ● 添加 GPIO 模块外部中断设置函数 ● 简化 ADC 模块初始化函数 ● 合并 PIT 模块中断底层函数	2012/3/21
0.3	● 底层驱动包添加 LPTMR 模块相关函数，并更新相应说明	2012/3/26
0.4	● 添加 FTM 模块的输入捕捉相关函数 ● 底层驱动包添加 UART 模块相关函数，并更新相应说明	2012/3/31
0.5	● 底层驱动包添加 I2C 模块相关函数，并更新相应说明	2012/4/7
0.6	● 底层驱动包添加 SDHC 模块及磁盘 IO 模块相关函数，并更新相应说明 ● 底层驱动包以磁盘 IO 模块底层函数为基础，添加开源的 FatFs 文件系统，并更新相应说明	2012/6/24
1.0	● 更新 ADC、GPIO、I2C、LPTMR、PIT、UART 等底层函数的参数名，使用旧版函数的程序需做小量兼容性修改 ● 修复 ADC 单端采集 BUG	2012/11/11
2.0	● 更新 GPIO 的 LPLD_GPIO_SetIsr()函数，使用后自动使能相应中断 ● 增加 GPIO、PIT、LPTMR 的清除中断函数 ● 增加 CAN、DAC、SPI、PDB、RTC、ENET、USB 模块的底层函数库	2012/11/22
2.1	● 修改 PDB 中断函数 BUG ● 增加 FLASH 模块底层库函数库	2012/12/31
2.2	● 重新编写 SDHC 底层驱动，解决多块读写 BUG ● 修改 SPI 底层驱动初始化函数的注释错误 ● 修改 nRF24L01\MAG3110\MMA7660 等例程的 BUG ● 增加 DMA 实现脉冲累加功能的例程	2013/4/5

注：自 V2.2 版后增加对示例程序的修改记录。

目录

1 文档信息

1.1 开发包描述

由拉普兰德电子技术所编写的 K60 底层库的代号为“LPLD_OSKinetis”，该驱动的所有源文件、头文件均存放在 lib/LPLD 目录下。基于 Kinetis K60 微处理器的功能模块分别定义不同的源文件和头文件。用户在用到某一模块时，只需要在工程文件的头部调用相应模块的头文件即可。

1.2 开发包特点

一步超频：只需更改宏定义，即可改变当前工程的内核频率，无需调用任何代码。

调用简单：所有模块的使用均可直接调用接口函数，无需了解底层配置。

自定义中断：只需将自定义的函数名作为参数传入模块初始化函数，即可轻松实现自定义中断，且中断函数不用清除标志位，如同编写普通函数一样简单。

代码全开源：所有底层代码均为明码形式，免费发布于网络，永远不会捆绑产品出售，用户可遵循开源协议自行学习与修改。

1.3 开源与更新

LPLD_OSKinetis 开发包为拉普兰德电子技术有限公司开发并维护的底层库，包内所有代码均已开源，开发者需遵守代码头部的开源协议。

开源代码会随着本公司内部的逐步测试更新更多模块，同时接受开发者的修改建议，如果你发现代码中存在漏洞或错误，请发电邮至：laplenden@126.com。我们会不定期发布新的代码包，逐步完善所有底层模块的应用。

开发包的更新地址：http://www.lpld.cn/?page_id=75

1.4 如何开始

本文档仅对底层库函数做相关说明，有关 **K60 的学习、工程的建立、例程的使用方法** 请见其他文档：

K60 相关资料：http://www.lpld.cn/?tag=k60

QQ 讨论群：**184156168**（必须输入验证信息 K60）

2 模块函数说明

以下将分别介绍 MCG、GPIO、ADC、FTM、eDMA、LPTMR、UART、I2C 等模块的函数说明。

2.1 MCG 模块

MCG 模块全称 Multipurpose Clock Generator，意为多用途时钟发生器。该模块的所有函数定义在 HAL_MCG.c 代码内。该模块下的函数可以改变 K60 各模块的时钟频率。

2.1.1 LPLD_PLL_Setup()

该函数基于官方示例代码 mcg.c 中的 pll_init()函数修改而来。并将原函数的参数简化到一个，开发者设置一个参数，即可定义不同的内核频率和其他系统时钟。该函数一般不出现在开发者的应用代码中，因为该函数已经在 main()主函数之前被系统初始化代码调用了。该函数的调用处为 sysinit()函数，除非用户有特殊需求，否则系统的初始化部分代码一般不用修改。

该函数在设置用户期望的内核时钟(Core Clk)后，会自动根据内核时钟分频得到其他的系统时钟，如 Bus 时钟、FlexBus 时钟、Flash 时钟。以上 3 种时钟频率在官方文档中均定义不能超过 50Mhz、50Mhz 和 25Mhz，因此我们建议用户调用 PLL_100、PLL_150、PLL_200 这 3 个参数，因为在以上参数下，其他系统时钟均可达到最优。

函数原型

unsigned char LPLD_PLL_Setup(unsigned char pll_option);

参数

pll_option -内核时钟频率选项，该参数为枚举类型中的变量，详细参数如下表所示：

参数值	描述
PLL_60	将内核时钟设置为 x MHz,其中 x 为 PLL 后的数字。
PLL_70	
PLL_80	
PLL_90	
PLL_100	

PLL_110	
PLL_120	
PLL_130	
PLL_150	
PLL_170	
PLL_180	
PLL_200	
PLL_209	

表 2.1.1.a

返回

该函数返回内核频率，单位为 Mhz，如果为 0，则初始化失败。

2.2 GPIO 模块

GPIO 模块全称 General purpose input/output，意为多通用输入输出。该模块的所有函数定义在 HAL_GPIO.c 代码内。该模块下的函数可以配置所有可用 IO 口的输入输出、触发、中断等操作。

2.2.1 LPLD_GPIO_Init()

该函数用来初始化指定的 GPIO 口，包括 GPIO 口的组别、指定的位数、输入输出方向、上拉或下拉配置、中断配置等操作。

函数原型

uint8 LPLD_GPIO_Init(PTx ptx, uint8 port_bit, uint8 dir, uint8 data1, uint8 irqc);

参数

ptx -端口号，该参数选择 GPIO 口 A~E 其中的一组，具体参数选项见下表：

参数值	描述
PTA	端口 A
PTB	端口 B
PTC	端口 C
PTD	端口 D
PTE	端口 E

表 2.2.1.a

port_bit -端口位数，该参数选择具体某一端口的某一位，值为 **0~31**。

dir -端口数据方向，该参数配置端口的输入输出方向。**DIR_INPUT** 为输入；**DIR_OUTPUT** 为输出。

data1 -输出初始电平或输入上拉、下拉，具体参数选项键下表：

参数值	描述
INPUT_PDOWN	如果 dir=DIR_INPUT,输入内部下拉
INPUT_PUP	如果 dir=DIR_INPUT,输入内部上拉
OUTPUT_L	如果 dir=DIR_OUTPUT,输出低电平
OUTPUT_H	如果 dir=DIR_OUTPUT,输出高电平

表 2.2.1.b

irqc -输入中断配置，该参数配置 PORTx_PCRn 寄存器 中的 IRQC 位，具体参数选项键下

表：

参数值	描述
IRQC_DIS	禁止中断和 DMA 请求
IRQC_DMARI	在输入信号上升沿产生 DMA 请求
IRQC_DMAFA	在输入信号下降沿产生 DMA 请求
IRQC_DMAET	在输入信号上升和下降沿产生 DMA 请求
IRQC_L	在逻辑 0 时产生中断
IRQC_RI	在输入信号上升沿产生中断
IRQC_FA	在输入信号下降沿产生中断
IRQC_ET	在输入信号上升和下降沿产生中断
IRQC_H	在逻辑 1 时产生中断

表 2.2.1.c

返回

0 为初始化失败，1 为初始化成功。

2.2.2 LPLD_GPIO_Set()

设置 Port 口的输出值。该函数控制某端口的全部位的输出。

函数原型

void LPLD_GPIO_Set(PTx ptx, uint32 data32);

参数

ptx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.2.1.a**。

data32 -输出数据,该参数为 32 位无符号型数据。参数范围 **0x00000000~0xFFFFFFFF** 低到高代表 Port 口的第 0~31 位数据。

2.2.3 LPLD_GPIO_Set_b()

设置 Port 口某一位的输出值。

函数原型

void LPLD_GPIO_Set_b(PTx ptx, uint8 port_bit, uint8 data1);

参数

ptx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.2.1.a**。

port_bit - Port 端口基地址的某一位，参数为 **0~31**。

data1 -输出数据。**OUTPUT_L** 为低电平；**OUTPUT_H** 为高电平。

返回

无

2.2.4 LPLD_GPIO_Toggle()

设置 Port 口输出值的翻转。

函数原型

void LPLD_GPIO_Toggle(PTx ptx, uint32 data32);

参数

ptx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.2.1.a**。

data32 -Port 端口翻转设置,该参数为 32 位无符号型数据。参数范围 **0x00000000~0xFFFFFFFF**，低到高代表 Port 口的第 0~31 位的翻转，1 为反转，0 为保持不变。

返回

无

2.2.5 LPLD_GPIO_Toggle_b()

设置 Port 端口某一位的翻转。

函数原型

void LPLD_GPIO_Toggle_b(PTx ptx, uint8 port_bit);

参数

ptx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.2.1.a**。

port_bit -Port 端口某一位翻转设置，参数为 **0~31**，翻转设置的端口位数。

返回

无

2.2.6 LPLD_GPIO_Get()

取得 Port 口的 0~31 位数据。

函数原型

uint32 LPLD_GPIO_Get(PTx ptx);

参数

ptx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.2.1.a**。

返回

该函指定 Port 口的 0~31 位数据。

2.2.7 LPLD_GPIO_Get_b()

取得 Port 口 0~31 位某一位的数据。

函数原型

uint8 LPLD_GPIO_Get_b(PTx ptx, uint8 port_bit);

参数

ptx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.2.1.a**。

port_bit -设置 Port 端口的某一位,参数为 **0~31**。

返回

该函指定 Port 口 port_bit 位数据。**OUTPUT_L** 为低电平；**OUTPUT_H** 为高电平。

2.2.8 LPLD_GPIO_SetIsr()

设置 GPIO 通道用户定义的中断服务函数，同时使能相应中断。

函数原型

uint8 LPLD_GPIO_SetIsr(PTx ptx, GPIO_ISR_CALLBACK isr_func);

参数

ptx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.2.1.a**。

isr_func -用户中断程序入口地址。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回

0 为初始化失败，1 为初始化成功。

2.2.9 LPLD_GPIO_ClearIsr()

清除 GPIO 通道用户定义的中断服务函数，同时禁用相应中断。

函数原型

uint8 LPLD_GPIO_ClearIsr(PTx ptx);

参数

ptx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.2.1.a**。

返回

0 为初始化失败，1 为初始化成功。

2.2.10 GPIO 中断示例

如果用户需要调用 GPIO 模块并产生中断，首先要在工程文件中包含头文件“HAL_GPIO.h”。然后再在工程目录下的 isr.h 文件中复制如下代码：

```
// GPIO 模块中断服务定义
#undef    VECTOR_103
#define VECTOR_103 LPLD_GPIO_Isr
#undef    VECTOR_104
#define VECTOR_104 LPLD_GPIO_Isr
#undef    VECTOR_105
#define VECTOR_105 LPLD_GPIO_Isr
#undef    VECTOR_106
#define VECTOR_106 LPLD_GPIO_Isr
#undef    VECTOR_107
#define VECTOR_107 LPLD_GPIO_Isr
//以下函数在 LPLD_Kinetis 底层包，不必修改
extern void LPLD_GPIO_Isr(void);
```

然后在你的工程文件中定义一个中断函数，该函数体中只需运行自定义代码即可，无需清除标志寄存器，但需判断产生中断引脚位数才可以区分是哪个引脚产生的中断。将该中断函数名作为参数传入 LPLD_GPIO_SetIsr()函数即可，但首先应该在 LPLD_GPIO_Init()函数中配置中断触发方式。例如：

```
//GPIO 中断函数定义
void porta_isr(void);

//主函数
void main (void)
{
   //配置 PortA0 为输入，
   //内部上拉，
   //下降沿触发中断。
   LPLD_GPIO_Init(PTA, 0, DIR_INPUT, INPUT_PUP, IRQC_FA);
   //设置 PortA 的中断函数为 porta_isr
   LPLD_GPIO_SetIsr (PTA, porta_isr);
   while(1) {}
}

//用户自定义中断函数
void porta_isr()
{
   //判断标志位，以识别中断位数是否为 A0
```

```
    if(PORTA_ISFR==0x00000001){
        printf("PortA0 产生中断。");
    }
}
```

2.3 ADC 模块

ADC 模块全称 Analog-to-Digital Converter，意为模拟转数字。该模块的所有函数定义在 HAL_ADC.c 代码内。该模块下的函数可以配置 ADC 模块的初始化，获取指定 AD 通道的转换值等。

2.3.1 LPLD_ADC_Init()

ADC 模块通用初始化函数，可用于配置非中断模式、单次转换、软件触发 AD 转换。

函数原型

uint8 LPLD_ADC_Init(ADCx adcx, uint8 mode, uint8 diff);

参数

adcx - ADC 模块号，该参数具体选项见下表：

参数值	描述
ADC0	选择 ADC0 模块
ADC1	选择 ADC1 模块

表 2.3.1a

mode -AD 转换精度，该参数设置 ADC 模块的转换精度，单端和差分输入时的精度不一样，具体参数选项见下表：

参数值	描述
MODE_8	单端 8 位精度，差分 9 位精度
MODE_12	单端 12 位精度，差分 13 位精度
MODE_10	单端 10 位精度，差分 11 位精度
MODE_16	单端 16 位精度，差分 16 位精度

表 2.3.1b

diff -单端输入或差分输入。**CONV_SING** 为单端输入；**CONV_DIFF** 为差分输入。

返回

0 初始化失败，1 为初始化成功。

2.3.2 LPLD_ADC_SE_Get()

该函数取得 ADC 模块单端输入的转换值，获取方式为软件触发方式。

函数原型

uint16 LPLD_ADC_SE_Get(ADCx adcx, uint8 channel);

参数

adcx -ADC 寄存器基地址，该参数具体选项见**表 2.3.1a**。

channel -ADC 通道号，不同的通道号对应不同的引脚，注意有些通道并非都有 ADC0 或 ADC1，具体参数选项如下表所示：

参数值	描述	
	ADC0	ADC1
0	PGA0_DP	PGA1_DP
1	PGA2_DP	PGA3_DP
2	PGA0_DP	PGA1_DP
3	ADC0_DP3	PGA0_DP
4	NC	PTE0
5	NC	PTE1
6	NC	PTE2
7	NC	PTE3
8	PTB0	PTB0
9	PTB1	PTB1
10	PTA7	PTB4
11	PTA8	PTB5
12	PTB2	PTB6
13	PTB3	PTB7
14	PTC0	PTB10
15	PTC1	PTB11
16	ADC0_SE16	ADC1_SE16
17	PTE24	PTA17
18	PTE25	VREF Output
19	PGA0_DM	PGA1_DM
20	PGA2_DM	PGA3_DM
23	DAC0_OUT	DAC1_OUT
26	TemperatureSensor	Temperature Sensor
27	Bandgap	Bandgap

29	VREFH	VREFH
30	VREFL	VREFL

表 2.3.1a

返回

0 -配置错误，或电压值为 0；

其他值 -AD 通道转换值，右对齐。

2.4 FTM 模块

FTM 模块全称 FlexTimer Module，意为 Flex 定时器模块。该模块的所有函数定义在 HAL_FTM.c 代码内。该模块下的函数可以配置输入捕获、输出比较、PWM 生成等功能。

2.4.1 LPLD_FTM0_PWM_Init()

该函数初始化 FTM0 模块的 PWM 功能，用户只需输入期望频率值，函数就会自动根据总线频率计算出相关配置参数并自动初始化相关寄存器。

函数原型

uint8 LPLD_FTM0_PWM_Init(uint32 freq);

参数

Freq -期望频率，单位 Hz。用户输入的期望 PWM 频率不能超过总线时钟(Bus Clk)最高频率，即 50MHz。

返回

0 初始化失败，1 为初始化成功。

2.4.2 LPLD_FTM0_PWM_Open()

该函数控制 FTM0 模块的相应 PWM 通道开启，并配置占空比。

函数原型

uint8 LPLD_FTM0_PWM_Open(uint8 channel, uint32 duty);

参数

channel - PWM 输出通道。PWM 通道输出分布在不同端口，具体参数选项见下表：

参数值	描述
0	PTC1 输出
1	PTC2 输出
2	PTC3 输出

3	PTC4 输出
4	PTD4 输出
5	PTD5 输出
6	PTD6 输出
7	PTD7 输出

表 2.4.2a

duty -PWM 输出占空比。范围为 0-~10000，对应占空比 0.00%~100.00%。

返回

0 为配置失败，1 为成功。

2.4.3 LPLD_FTM0_PWM_ChangeDuty()

该函数改变 FTM0 模块 PWM 输出通道的占空比。

函数原型

uint8 LPLD_FTM0_PWM_ChangeDuty(uint8 channel, uint32 duty);

参数

channel - PWM 输出通道。PWM 通道输出分布在不同端口，具体参数选项**表 2.4.2a**。

duty -PWM 输出占空比。范围为 0-~10000，对应占空比 0.00%~100.00%。

返回

0 为配置失败，1 为成功。

2.4.4 LPLD_FTM1_PWM_Init()

该函数初始化 FTM1 模块的 PWM 功能，用户只需输入期望频率值，函数就会自动根据总线频率计算出相关配置参数并自动初始化相关寄存器。

函数原型

uint8 LPLD_FTM1_PWM_Init(uint32 freq);

参数

Freq -期望频率，单位 Hz。用户输入的期望 PWM 频率不能超过总线时钟(Bus Clk)最高频率，即 50MHz。

返回

0 初始化失败，1 为初始化成功。

2.4.5 LPLD_FTM1_PWM_Open()

该函数控制 FTM1 模块的相应 PWM 通道开启，并配置占空比。

函数原型

uint8 LPLD_FTM1_PWM_Open(uint8 channel, uint32 duty);

参数

channel - PWM 输出通道。PWM 通道输出分布在不同端口，具体参数选项见下表：

参数值	描述
0	PTA8 输出
1	PTA9 输出

表 2.4.5a

duty -PWM 输出占空比。范围为 0-~10000，对应占空比 0.00%~100.00%。

返回

0 为配置失败，1 为成功。

2.4.6 LPLD_FTM1_PWM_ChangeDuty()

该函数改变 FTM0 模块 PWM 输出通道的占空比。

函数原型

uint8 LPLD_FTM1_PWM_ChangeDuty(uint8 channel, uint32 duty);

参数

channel - PWM 输出通道。PWM 通道输出分布在不同端口，具体参数选项**表 2.4.5a**。

duty -PWM 输出占空比。范围为 0-~10000，对应占空比 0.00%~100.00%。

返回

0 为配置失败，1 为成功。

2.4.7 LPLD_FTM0_InputCapture_Init()

FTM0 模块输入捕捉功能初始化函数。

函数原型

```
uint8 LPLD_FTM0_InputCapture_Init(uint8 channel,
                                  uint8 edge,
                                  uint8 ps,
                                  FTM_ISR_CALLBACK isr_func);
```

参数

channel -输入捕捉通道。输入通道分布在不同端口，具体参数见表 **2.4.7a**：

参数值	描述
0	PTC1 输入
1	PTC2 输入
2	PTC2 输入
3	PTC4 输入
4	PTD4 输入
5	PTD5 输入
6	PTD6 输入
7	PTD7 输入

表 2.4.7a

edge -捕捉边缘选择。选择参数见表 **2.4.7b**：

参数值	描述
1	上升沿捕捉
2	下降沿捕捉
3	上升和下降沿捕捉

表 2.4.7b

ps -计数器时钟分频，计数器时钟由总线时钟分频得来，该分频值越小计数器时钟频率越高，可由大值向小值试。选择参数见表 **2.4.7c**：

参数值	描述
0	1 分频
1	2 分频
2	4 分频
3	8 分频
4	16 分频
5	32 分频
6	64 分频
7	128 分频

表 2.4.7c

isr_func -用户中断程序入口地址，该中断函数可由输入电平跳变触发或计数器计数溢出触发，用户需在函数中对触发方式进行判断。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回

0 为配置失败，1 为成功。

2.4.8 LPLD_FTM1_InputCapture_Init()

FTM1 模块输入捕捉功能初始化函数。

函数原型

```
uint8 LPLD_FTM1_InputCapture_Init(uint8 channel,
                                  uint8 edge,
                                  uint8 ps,
                                  FTM_ISR_CALLBACK isr_func);
```

参数

channel -输入捕捉通道。输入通道分布在不同端口，具体参数见表 **2.4.8a**：

参数值	描述

0	PTA8 输入
1	PTA8 输入

表 2.4.8a

edge -捕捉边缘选择。选择参数见表 **2.4.8b**：

参数值	描述
1	上升沿捕捉
2	下降沿捕捉
3	上升和下降沿捕捉

表 2.4.8b

ps -计数器时钟分频，计数器时钟由总线时钟分频得来，该分频值越小计数器时钟频率越高，可由大值向小值试。选择参数见表 **2.4.8c**：

参数值	描述
0	1 分频
1	2 分频
2	4 分频
3	8 分频
4	16 分频
5	32 分频
6	64 分频
7	128 分频

表 2.4.8c

isr_func -用户中断程序入口地址，该中断函数可由输入电平跳变触发或计数器计数溢出触发，用户需在函数中对触发方式进行判断。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回

0 为配置失败，1 为成功。

2.4.9 LPLD_FTM2_InputCapture_Init()

FTM1 模块输入捕捉功能初始化函数。

函数原型

uint8 LPLD_FTM2_InputCapture_Init(uint8 channel,

uint8 edge,

uint8 ps,

FTM_ISR_CALLBACK isr_func);

参数

channel -输入捕捉通道。输入通道分布在不同端口，具体参数见表 **2.4.9a**：

参数值	描述
0	PTA10 输入
1	PTA11 输入

表 2.4.9a

edge -捕捉边缘选择。选择参数见表 **2.4.9b**：

参数值	描述
1	上升沿捕捉
2	下降沿捕捉
3	上升和下降沿捕捉

表 2.4.9b

ps -计数器时钟分频，计数器时钟由总线时钟分频得来，该分频值越小计数器时钟频率越高，可由大值向小值试。选择参数见表 **2.4.9c**：

参数值	描述
0	1 分频
1	2 分频
2	4 分频
3	8 分频
4	16 分频
5	32 分频
6	64 分频
7	128 分频

表 2.4.9c

isr_func -用户中断程序入口地址，该中断函数可由输入电平跳变触发或计数器计数溢出触发，用户需在函数中对触发方式进行判断。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回

0 为配置失败，1 为成功。

2.4.10 FTM 中断示例

如果用户需要调用 FTM 模块的中断函数一般可由计数器溢出、通道输入输出和错误触发，要使用中断函数，首先要在工程文件中包含头文件“HAL_FTM.h”。然后再在工程目录下的 isr.h 文件中复制如下代码：

```
// FTM 模块中断服务定义
#undef    VECTOR_078
#define VECTOR_078 LPLD_FTM_Isr
#undef    VECTOR_079
#define VECTOR_079 LPLD_FTM_Isr
#undef    VECTOR_080
#define VECTOR_080 LPLD_FTM_Isr
//以下函数在 LPLD_Kinetis 底层包，不必修改
extern void LPLD_FTM_Isr (void);
```

然后在你的工程文件中定义一个中断函数，该函数体中只需运行自定义代码即可，无需清除标志寄存器，但需判断产生中断的触发源是计数溢出还是通道中断或其他。将该中断函数名作为参数传入 LPLD_FTMx_InputCapture_Init()函数即可。例如：

```
//调用外部变量，获得总线频率
extern int periph_clk_khz;
//调用外部变量，获得 FTM1 的分频系数
extern uint8 LPLD_FTM1_Divider;
//全局变量，存储捕获的脉冲频率
uint32 Frq1;

//FTM 中断函数定义
void ftm1_isr (void);

//主函数
void main (void)
{
  //配置 FTM1 的 0 通道为脉冲捕捉口，上升沿触发捕捉，时钟分频系数 128，中断函数 ftm1_isr
  LPLD_FTM1_InputCapture_Init(0, 1, 7, ftm1_isr);
  while(1) {}
}

//用户自定义中断函数
void ftm1_isr (void)
```

```
{
  //输入捕捉边缘检测中断
  if(FTM1_C0SC & FTM_CnSC_CHF_MASK)
  {
    //用户自定义代码 开始
    Frq1=FTM1_C0V;  //获得计数值
    Frq1=periph_clk_khz*1000/LPLD_FTM1_Divider/Frq1; //根据总线频率和分频系数计算脉冲频率

    //用户自定义代码 结束

    //清空FTM1 COUNTER
    FTM1_CNT = 0;
    //清除输入中断标志
    FTM1_C0SC &=(~FTM_CnSC_CHF_MASK);
  }
  //输入捕捉计数器溢出中断
  else if(FTM1_SC & FTM_SC_TOF_MASK)
  {

    //用户自定义代码 开始

    //用户自定义代码 结束

    //清除计数溢出中断标志
    FTM1_SC &=(~FTM_SC_TOF_MASK);
  }
}
```

2.5 PIT 模块

PIT 模块全称 Periodic Interrupt Timer，意为周期中断定时器。该模块的所有函数定义在 HAL_PIT.c 代码内。该模块下的函数可以配置 K60 开启定时器，并产生周期性中断。

2.5.1 LPLD_PIT_Init()

该函数为 PIT 模块通用定时器，可以配置 PIT 通道，指定中断周期和中断函数，并使能中断。

函数原型

uint8 LPLD_PIT_Init(PITx pitx, uint32 period_us, PIT_ISR_CALLBACK isr_func);

参数

pitx -PIT 模块号。具体参数选项见下表：

参数值	描述
PIT0	周期中断定时器 0
PIT1	周期中断定时器 1
PIT2	周期中断定时器 2
PIT3	周期中断定时器 3

表 2.5.1.a

period_us -定时周期。范单位为微秒，即 us。

isr_func -用户中断程序入口地址。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回

0 为配置失败，1 为成功。

2.5.2 LPLD_PIT_SetIsr()

设置 PIT 模块用户定义的中断服务函数，同时使能相应中断。

函数原型

uint8 LPLD_PIT_SetIsr(PITx pitx, PIT_ISR_CALLBACK isr_func);

参数

pitx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.5.1.a**。

isr_func -用户中断程序入口地址。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回

0 为初始化失败，1 为初始化成功。

2.5.3 LPLD_PIT_ClearIsr()

清除 PIT 模块用户定义的中断服务函数，同时禁用相应中断。

函数原型

uint8 LPLD_PIT_ClearIsr(PITx pitx);

参数

pitx -端口号，该参数选择 Port A~E 其中的一组，具体参数同**表 2.5.1.a**。

返回

0 为初始化失败，1 为初始化成功。

2.5.4 PIT 定时中断示例

如果用户需要调用 PIT 模块并产生中断，首先要在工程文件中包含头文件“HAL_PIT.h”。然后再在工程目录下的 isr.h 文件中复制如下代码：

```
//PIT 模块中断服务定义
#undef    VECTOR_084
#define VECTOR_084 LPLD_PIT_Isr
#undef    VECTOR_085
#define VECTOR_085 LPLD_PIT_Isr
```

```
#undef    VECTOR_086
#define VECTOR_086 LPLD_PIT_Isr
#undef    VECTOR_087
#define VECTOR_087 LPLD_PIT_Isr
//以下函数在LPLD_Kinetis 底层包，不必修改
extern void LPLD_PIT_Isr (void);
```

然后在你的工程文件中定义一个中断函数，该函数体中只需运行自定义代码即可，无需操作标志寄存器。然后将该函数名作为参数传入 LPLD_PIT_Init()函数即可。例如：

```
//定时中断函数定义
void my_pit0_isr(void);

//主函数
void main (void)
{
  //初始化定时器 0，周期1000000us 即 1 秒，中断函数为my_pit0_isr
  LPLD_PIT_Init(PIT0, 1000000, my_pit0_isr);
  while(1)
  {
  }
}

//用户自定义中断函数
void my_pit0_isr()
{
  printf("pit0\r\n");
}
```

2.6 eDMA 模块

eDMA 全称 enhanced direct memory access，模块的主要作用是为源存储器和目的存储器之间搭建桥梁，可以直接将数据由源存储器传输到目的存储器上，整个过程无需 CPU 的参与。该模块的所有函数均定义在 HAL_eDMA.c 文件中，涉及 eDMA 的宏定义、结构体均在 HAL_eDMA.h 头文件中。

2.6.1 LPLD_DMA_Init()

该函数用于初始化 DMA 模块，用户在使用该函数之前要先配置结构体参数 DMA_Config 的必选参数（见表 **2.6.1a**），然后本函数会自动补全其他可选参数，并调用 LPLD_eDMA_Config()函数对 DMA 寄存器进行配置，该函数执行完毕以后不开启 DMA 模块。用户可调用 LPLD_DMA_Start()函数开启 DMA 模块。

函数原型

*uint8 LPLD_DMA_Init(LPLD_eDMA_Cfg_t *DMA_Config);*

参数

DMA_ConfigDMA* –DMA 的全部初始化参数定义在 LPLD_eDMA_Cfg_t 结构体中，用户在调用此函数的之前需要声明该结构体的变量，然后对该变量中的参数进行初始化。结构体中的全部参数见下表 **2.6.1a，其中加**粗体**为配置时必须设置的参数。

参数名称	**描述**
Channelx	DMA 通道号。 参数值见表 **2.6.1b**
Peri_DmaReq	DMA 请求源，在 HAL_eDMA.h 中，利用宏定义形式列出。
Minor_loop_Length	传输总的字节数 （主循环计数器的次数）。
Trans_bytesNum	DMA 一次传输字节的个数。
Source_Addr	数据传输的源地址。
Source_Size	源地址传输数据的宽度。 参数值见表 **2.6.1c**
Source_Addr_inc	执行完 DMA 请求后，源地址是否累加。 参数值见表 **2.6.1d**
Source_Adj_Addr	主循环计数器归零，源地址是否调整。
Dest_Addr	数据传输的目的地址。 参数值见表 **2.6.1e**
Dest_Size	目的地址接收数据的宽度。 参数值见表 **2.6.1f**

Dest_Addr_inc	执行完 DMA 请求后，目的地址是否累加。
Dest_Adj_Addr	主循环计数器归零，目的地址是否调整。
Dma_irqc	DMA 中断选择。 参数值见表 **2.6.1g**
Dma_AutoClose	主循环计数器归零，DMA 是否自动关闭 参数值见表 **2.6.1h**
isr_func	用于存放中断回调函数的首地址

表 2.6.1a

结构体 *LPLD_eDMA_Cfg_t* 参数说明

Channelx

参数值	**描述**
0	DMA 通道 0
1	DMA 通道 1
2	DMA 通道 2
3	DMA 通道 3
……	……
15	DMA 通道 15

表 2.6.1b

Peri_DmaReq

-外设的 DMA 请求源，在 HAL_eDMA.h 中定义，用户根据具体的外设来配置 DMA 请求源。在 HAL_eDMA.h 中定义了 DMA_MUX1 的 DMA 请求源共有 56 个

Minor_loop_Length

-DMA 需要传输字节的个数，也是主循环计数器的计数个数，每传输一个字节，主循环计数器减一，用户最大可以设置的传输字节数为 32768 个。

Trans_bytesNum

-该参数表示 DMA 触发一次所传输字节的个数。默认值为 1，一次传输一个字节。

Source_Addr

-源地址的长度为 32 位。用户可以将源地址设置为外设的数据寄存器的地址，如 ADC 的数据寄存器的地址，IO 的数据寄存器的地址等等，也可以将源地址设置为存储区的首地址。

Source_Size

-该参数值的宏定义在 HAL_eDMA.h 中，默认值为 DMA_SRC_8BIT，即源地址传输数据的宽度为 1 个字节。

参数值	描述
DMA_SRC_8BIT	源地址传输数据的宽度为 8 位，1 个字节。
DMA_SRC_16BIT	源地址传输数据的宽度为 16 位，2 个字节。
DMA_SRC_32BIT	源地址传输数据的宽度为 32 位，4 个字节。

表 2.6.1c

Source_Addr_inc

-该参数值的宏定义在 HAL_eDMA.h 中，默认初始值为 ADDR_HOLD，源地址不增加。

参数值	描述
ADDR_HOLD	执行完 DMA 请求后，源地址不增加
ADDR_INCREASE	执行完 DMA 请求后，源地址加一

表 2.6.1d

Source_Adj_Addr

-用户根据实际情况设置该参数值，该参数设置完毕后，当主循环计数器归零后源地址将调整到该参数所设置的地址。默认初始值为 0：不调整源地址。

Dest_Addr

-用户可以将目的地址设置为外设的数据寄存器的地址，如 ADC 的数据寄存器的地址，IO 的数据寄存器的地址等等，也可以将目的地址设置为存储区的首地址。目的地址的长度为 32 位。

Dest _Size

-该参数值的宏定义在 HAL_eDMA.h 中，默认值为 DMA_DST_8BIT。

参数值	描述
DMA_DST_8BIT	目的地址传输数据的宽度为 8 位
DMA_ DST_16BIT	目的地址传输数据的宽度为 16 位

DMA_ DST_32BIT	目的地址传输数据的宽度为 32 位

表 2.6.1e

Dest _Addr_inc

-该参数值的宏定义在 HAL_eDMA.h 中，默认初始值为 ADDR_INCREASE，执行完 DMA 请求后，目的地址加一。

参数值	描述
ADDR_HOLD	执行完 DMA 请求后，目的地址不增加
ADDR_INCREASE	执行完 DMA 请求后，目的地址加一

表 2.6.1f

Dest _Adj_Addr

-用户根据实际情况设置该参数值，该参数设置完毕后，当主循环计数器归零后目的地址将调整到该参数所设置的地址。默认初始值为 0，不调整目的地址。

Dma_irqc

-默认初始值为 0，关闭 DMA 中断。

参数值	描述
0	关闭 DMA 中断
1	主循环计数器计数减到一半，产生中断
2	主循环计数器计数减到零时，产生中断

表 2.6.1g

Dma_AutoClose

-默认初始值为 0，主循环计数器归零后关闭 DMA。

参数值	描述
0	主循环计数器归零后关闭 DMA
1	主循环计数器归零后保持 DMA 开启

表 2.6.1h

isr_func

-用户中断程序入口地址。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回值

0：配置失败，1：配置成功

2.6.2 LPLD_eDMA_Config()

此函数为内部函数，用户无需调用。

该函数调用用户配置的 DMA 参数结构体变量来配置 DMA 的相关寄存器。在 LPLD_DMA_Init()函数中已经调用该函数，因此用户一般可以不直接调用该函数。

函数原型

*uint8 LPLD_eDMA_Config(LPLD_eDMA_Cfg_t *DMA_Config);*

参数

DMA_Config* -DMA 配置结构体变量，见表 **2.6.1a。

返回值

0：配置失败，1：配置成功

2.6.3 LPLD_DMA_EnableReq()

该函数用于使能 DMA 相关通道的 DMA 请求，使能 DMA 请求就表示 DMA 模块开始运行。当 DMA 初始化完毕以后，利用该函数使 DMA 开始运行。

函数原型

void LPLD_DMA_EnableReq(uint8 chx,uint8 enable);

参数

chx -DMA 通道值，一共 16 个 DMA 通道 ，见表 **2.6.3a**。

参数值	描述
0	DMA 通道 0
……	……
15	DMA 通道 15

表 2.6.3a

enable -开启或关闭 DMA 请求，见表 **2.6.3b**。

参数值	描述
0	禁止 DMA 相关通道的 DMA 请求
1	使能 DMA 相关通道的 DMA 请求

表 2.6.3b

返回值

无

2.6.4 LPLD_DMA_Reload()

该函数用于重新加载DMA的目的地址、目的地址的偏移量以及重新更新主循环计数器。

函数原型

```
Void LPLD_DMA_Reload( uint8 chx,
                      uint32 dest_base_addr,
                      uint32 dest_offset_addr,
                      uint16 loop_length);
```

参数

chx -DMA 通道值，一共 16 个 DMA 通道 ，见表 **2.6.3a**。

dest_base_add -重新加载目的地址，用户可以将目的地址设置为外设的数据寄存器的地址，如 ADC 的数据寄存器的地址，IO 的数据寄存器的地址等等，也可以将目的地址设置为存储区的首地址。目的地址的长度为 32 位。

dest_offset_addr -重新加载目的地址的偏移量，利用该参数可以根据用户的需求设置目的地址的偏移量。最终的目的地址=目的地址+目的地址的偏移量，如果不想使目的地址偏移，设置此参数为 0。

loop_length -更新主循环计数器的值，最大长度 32768。

返回值

无

2.7 LPTMR 模块

LPTMR 模块全称 Low power timer，意为低功耗定时器模块。该模块的所有函数定义在 HAL_LPTMR.c 代码内。该模块下的函数可以配置 K60 在各种功耗模式下作为定时器和脉冲累加器使用。

2.7.1 LPLD_LPTMR_Init()

LPTMR 模块初始化函数。可以配置 LPTMR 为低功耗定时器模式、脉冲累加模式。

函数原型

```
uint8 LPLD_LPTMR_Init(uint8 mode,
                      uint16 period_ms,
                      uint8 channel,
                      uint8 irq_en,
                      LPTMR_ISR_CALLBACK isr_func);
```

参数

mode -选择 LPTMR 模块的工作模式，其参数见下表“

参数值	描述
MODE_LPTMR	低功耗定时器模式
MODE_PLACC	脉冲累加模式

表 2.7.1a

period_ms –配置定时器的定时值，单位 ms。当 *mod* 配置为脉冲累加时，此值为 **0**；否则，填入值为 **1~65535** 的整数。

channel –选择输入累加端口，其参数见下表：

参数值	描述
0	设置为定时器时选择为 0
LPTMR_ALT1	设置 PTA19 为脉冲累加输入
LPTMR_ALT2	设置 PTC5 为脉冲累加输入

表 2.7.1b

irqc –输入中断配置，其参数见下表：

参数值	描述
0	不使能中断
1	使能中断

表 2.7.1c

isr_func -用户中断程序入口地址。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回值

0：配置失败，1：配置成功

2.7.2 LPLD_LPTMR_Reset()

低功耗定时器复位，复位以后清空低功耗定时器计数寄存器和定时比较标志位。

函数原型

void LPLD_LPTMR_Reset(void);

参数

无

返回值

无

2.7.3 LPLD_LPTMR_GetPulseAcc()

得到脉冲累加的值。

函数原型

uint16 LPLD_LPTMR_GetPulseAcc(void);

参数

无

返回值

返回 0~65535 之间的整数。

2.7.4 LPLD_LPTMR_DelayMs()

低功耗定时器精准延时 N 毫秒。此函数和普通延时函数一样消耗 CPU 周期，但是比普通延时精准。

函数原型

void LPLD_LPTMR_DelayMs(uint16 period_ms);

参数

period_ms –配置延时的定时值，单位 ms。填入值为 0 至 65535 的整数。

返回值

无

2.7.5 LPLD_LPTMR_SetIsr()

设置 LPTMR 模块的用户定义的中断服务函数，同时使能相应中断。

函数原型

uint8 LPLD_LPTMR_SetIsr(LPTMR_ISR_CALLBACK isr_func);

参数

isr_func -用户中断程序入口地址。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回

0 为初始化失败，1 为初始化成功。

2.7.6 LPLD_LPTMR_ClearIsr()

清除 LPTMR 模块的用户定义的中断服务函数，同时禁用相应中断。

函数原型

uint8 LPLD_LPTMR_ClearIsr();

参数

无

返回

0 为初始化失败，1 为初始化成功。

2.8 UART 模块

UART 模块全称 Universal Asynchronous Receiver/Transmitter，意为通用异步收发器，即我们常说的串口模块。该模块的所有函数定义在 HAL_UART.c 代码内。该模块下的函数可以配置 K60 的各个串口以指定的速率接收和发送串口数据。

2.8.1 LPLD_UART_Init()

UART 模块通用初始化函数，可以指定串口通道和波特率。

函数原型

void LPLD_UART_Init(UARTx uartx, int baud);

参数

uartx -UART 模块号，该参数的值和对应的发送接收引脚见表 **2.8.1a**：

参数值	描述
UART0	串口 0 -Txd:PTB17 -Rxd:PTB16
UART1	串口 1 -Txd:PTC4 -Rxd:PTC3
UART2	串口 2 -Txd:PTD3 -Rxd:PTD2
UART3	串口 3 -Txd:PTC17 -Rxd:PTC16
UART4	串口 4 -Txd:PTC15 -Rxd:PTC14
UART5	串口 5 -Txd:PTE8 -Rxd:PTE9

表 2.8.1a

baud -串口通讯波特率，推荐设置以下常用波特率：**4800、9600、19200、38400、57600、115200**。

返回值

无

2.8.2 LPLD_UART_PutChar()

串口查询方式发送一个字节。

函数原型

void LPLD_UART_PutChar(UARTx uartx, char ch);

参数

uartx -UART 模块号，该参数的值和对应的发送接收引脚见表 **2.8.1a**。

ch -待发送的 1 个字节。

返回值

无

2.8.3 LPLD_UART_GetChar()

串口查询方式接收一个字节。

函数原型

char LPLD_UART_GetChar(UARTx uartx);

参数

uartx -UART 模块号，该参数的值和对应的发送接收引脚见表 **2.8.1a**。

返回值

串口接收的 1 个字节。

2.8.4 LPLD_UART_PutCharArr()

串口查询方式发送字节型数组，即发送一个字符串。

函数原型

*void LPLD_UART_PutCharArr(UARTx uartx, char *ch, int len);*

参数

uartx -UART 模块号，该参数的值和对应的发送接收引脚见表 **2.8.1a**。

**ch* -待发送的字节数组的头地址。

len -待发送的字节数组的长度。

返回值

无

2.8.5 LPLD_UART_RIE_Enable()

使能串口数据接收中断，设置中断函数入口。

函数原型

void LPLD_UART_RIE_Enable(UARTx uartx, UART_ISR_CALLBACK isr_func);

参数

uartx -UART 模块号，该参数的值和对应的发送接收引脚见表 **2.8.1a**。

isr_func -用户中断程序入口地址，当使能接收中断后，串口接收到 1 个字节便触发该函数。该参数为用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回值

无

2.8.6 LPLD_UART_RIE_Disable()

禁用串口数据接收中断。

函数原型

void LPLD_UART_RIE_Disable(UARTx uartx);

参数

uartx -UART 模块号，该参数的值和对应的发送接收引脚见表 **2.8.1a**。

返回值

无

2.9 I2C 模块

I2C 模块全称 Inter-Integrated Circuit，即我们常说的两线式串行总线模块。该模块的所有函数定义在 HAL_I2C.c 代码内。该模块下的函数可以配置 K60 与其他外围设备通过 I2C 总线进行通信。

2.9.1 LPLD_I2C_Init()

I2C 通用初始化函数，可以指定 I2C 通道和通信速率。

函数原型

uint8 LPLD_I2C_Init(I2Cx i2cx, uint8 port, uint8 scl_frq);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**：

参数值	描述
I2C0	I2C0 号模块
I2C1	I2C1 号模块

表 2.9.1a

port-通道端口位的选择，每个通道有两组端口可以选择，对应的参数见**表 2.9.1b**：

i2cx	port 参数值	描述
I2C0	0	PTB2-I2C0_SCL PTB3-I2C0_SDA
	1	PTD8-I2C0_SCL PTD9-I2C0_SDA
I2C1	0	PTE1-I2C1_SCL PTE0-I2C1_SDA
	1	PTC10-I2C1_SCL PTC11-I2C1_SDA

表 2.9.1b

scl_frq -选择 IIC 模块的 SCL 频率，对应的参数见**表 2.9.1c**：

参数值	描述
I2C_SCL_50KHZ	50khz
I2C_SCL_100KHZ	100khz
I2C_SCL_150KHZ	150khz

I2C_SCL_200KHZ	200khz
I2C_SCL_250KHZ	250khz
I2C_SCL_300KHZ	300khz

返回值

0：配置失败，1：配置成功

2.9.2 LPLD_I2C_Start()

I2C 通道开始信号产生。

函数原型

void LPLD_I2C_Start(I2Cx i2cx);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**。

返回值

无

2.9.3 LPLD_I2C_ReStart()

I2C 通道再次产生开始信号。

函数原型

void LPLD_I2C_ReStart (I2Cx i2cx);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**。

返回值

无

2.9.4 LPLD_I2C_Stop()

I2C 通道产生停止信号。

函数原型

void LPLD_I2C_Stop(I2Cx i2cx);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**。

返回值

无

2.9.5 LPLD_I2C_WaitAck()

I2C 设置等待应答信号，开启则等待，关闭则不等待。

函数原型

void LPLD_I2C_WaitAck(I2Cx i2cx, uint8 is_wait);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**。

is_wait -选择是否等待应答，该参数的值见**表 2.9.5a。**

参数值	描述
I2C_ACK_OFF	关闭等待 Ack。
I2C_ACK_ON	开启等待 Ack，并等待 ACK 信号。

表 2.9.5a

返回值

无

2.9.6 LPLD_I2C_Write()

I2C 发送一个字节数据给目的地址设备。

函数原型

void LPLD_I2C_Write(I2Cx i2cx, uint8 data8);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**。

data8 -要发送的字节数据。

返回值

无

2.9.7 LPLD_I2C_Read()

I2C 从外部设备读一个字节数据。

函数原型

uint8 LPLD_I2C_Read(I2Cx i2cx);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**。

返回值

I2C 读取的数据。

2.9.8 LPLD_I2C_SetMasterWR()

I2C 主机读写模式配置。

函数原型

void LPLD_I2C_SetMasterWR(I2Cx i2cx, uint8 mode);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**。

mode -读写选择，该参数的值见**表 2.9.8a**。

参数值	描述
I2C_MWSR	主机发送从机接收。
I2C_MRSW	主机接收从机发送。

表 2.9.8a

返回值

无。

2.9.9 LPLD_I2C_StartTrans()

I2C 开始传输函数，需要设置外围设备地址和读写模式。

函数原型

void LPLD_I2C_StartTrans(I2Cx i2cx, uint8 addr, uint8 mode);

参数

i2cx -选择 I2C 模块号，该参数的值和对应的通道号见**表 2.9.1a**。

addr -外围设备地址。

mode -读写选择，该参数的值见**表 2.9.8a**。

返回值

无。

2.10 SDHC 模块、磁盘 IO 模块及其文件系统

SDHC 就是高容量 SD 存储的缩写，K60 的 SDHC 模块支持 POLL 和 DMA 模式，本驱动采用 POLL 模式。该模块的所有函数定义在 HAL_SDHC.c 代码内。V2.2 以后的 SDHC 驱动代码，提取自飞思卡尔官方的 MQX 驱动库。

磁盘 IO 模块为定义在 SDHC 模块驱动之上的功能层模块，旨在将 SDHC 模块所控制的 SD 卡在功能上看做是一个磁盘系统，方便更高层的文件系统来调用。

本开源驱动包所采用的文件系统为 FatFs，其底层磁盘驱动函数均调用 FML_DiskIO.c 内的函数。

为了方便起见，本章节的函数介绍只介绍磁盘 IO 模块层的主要函数，因为用户在使用时一般不需要调用 SDHC 模块的底层函数，因此就不在此具体介绍了，有兴趣的朋友可以研究下磁盘 IO 模块的函数是如何调用 SDHC 底层函数的。

2.10.1 LPLD_Disk_Initialize()

磁盘初始化。该函数内部调用 SDHC 模块的 LPLD_SDHC_InitCard()函数用以初始化 SD 卡及 SDHC 模块相关寄存器。

函数原型

DSTATUS LPLD_Disk_Initialize(uint8 drv);

参数

drv -物理磁盘号，只能为 0。

返回值

返回磁盘当前状态 DSTATUS，具体参数见**表 2.10.1a**。

参数值	描述
RES_OK	成功
RES_ERROR	读/写错误
RES_WRPRT	写保护
RES_NOTRDY	未准备好
RES_PARERR	参数错误
RES_NONRSPNS	未响应

表 2.10.1a

2.10.2 LPLD_Disk_Status()

返回磁盘状态。该函数内部调用 SDHC 模块的 LPLD_SDHC_GetStatus()函数。

函数原型

DSTATUS LPLD_Disk_Status(uint8 drv);

参数

drv -物理磁盘号，只能为 0。

返回值

返回磁盘当前状态 DSTATUS，具体参数见**表 2.10.1a**。

2.10.3 LPLD_Disk_Read()

读磁盘的一个或多个扇区。该函数内部调用 SDHC 模块的 LPLD_SDHC_ReadBlocks()函数来读 SD 卡的扇区。

函数原型

DRESULT LPLD_Disk_Read(uint8 drv, uint8 buff, uint32 sector, uint8 count);*

参数

drv -物理磁盘号，只能为 0。

buff -数据读出所存的地址指针。

sector -扇区起始号。

count -所读的扇区数。

返回值

返回磁盘当前状态 DSTATUS，具体参数见**表 2.10.1a**。

2.10.4 LPLD_Disk_Write()

写磁盘的一个或多个扇区。该函数内部调用 SDHC 模块的 LPLD_SDHC_WriteBlocks()函数来写 SD 卡的扇区。注意，当_READONLY 不为 0 时，即磁盘 IO 模块系统为只读模式时，该函数不被编译。

函数原型

DRESULT LPLD_Disk_Write(uint8 drv, uint8 buff, uint32 sector, uint8 count);*

参数

drv -物理磁盘号，只能为 0。

buff -数据读出所存的地址指针。

sector -扇区起始号。

count -所读的扇区数。

返回值

返回磁盘当前状态 DSTATUS，具体参数见**表 2.10.1a**。

2.10.5 LPLD_Disk_IOC()

磁盘功能控制函数。该函数调用 SDHC 底层模块的相应驱动函数来实现对 SD 卡的相应功能控制。

函数原型

DRESULT LPLD_Disk_IOC(uint8 drv, uint8 ctrl, void buff);*

参数

drv -物理磁盘号，只能为 0。

ctrl -控制命令，具体参数见**表 2.10.5a**。

参数值	描述
CTRL_SYNC	确定磁盘驱动已经完成写操作挂起的处理。当磁盘 IO 模块有一个会写缓存，会立即擦除扇区。该命令不能再只读模式使用。

GET_SECTOR_SIZE	以 WORD 型指针变量的形式返回扇区大小。此命令不能用在可变扇区大小的配置，_MAX_SS 默认为 512。
GET_SECTOR_COUNT	以 UINT32 型指针变量的形式返回磁盘的可用扇区数。该命令仅被 f_mkfs 函数调用以决定可创建多大的卷。
GET_BLOCK_SIZE	以 UINT32 类型的指针变量返回返回 flash 内存中擦除的扇区数。合法的数值为 2 的 1 至 32768 次方。返回 1 代表擦除大小或磁盘设备未知。该命令仅被 f_mkfs 函数调用并试图将擦除的扇区边界进行数据对齐。
CTRL_ERASE_SECTOR	擦除由 UINT32 类型指针数组指定的 flash 内存，{<start sector>, <end sector>}。如果介质为非 flash 内存,则该命令无效。FatFs 系统不会检查结果，如果擦除失败也不会影响文件函数。当_USE_ERASE 为 1 时移动一个簇链会调用此命令。

表 2.10.5a

buff - IO 操作所需数据的指针。

返回值

返回磁盘当前状态 DSTATUS，具体参数见**表 2.10.1a**。

2.10.6 FatFs 文件系统

在磁盘 IO 模块的基础上，本开源驱动包移植了 FatFs 的开源文件系统。本驱动包未对该文件系统的功能函数进行相应修改，用户使用时完全参照官方手册即可使用。也可以参照本驱动里提供的例程“LPLD_SDHC_FatFs”进行修改。

这里简要介绍下其应用接口函数：

f_mount – 注册或注销一个磁盘工作区

f_open – 打开或创建一个文件

f_close – 关闭一个文件

f_read – 读文件

f_write – 写文件

f_lseek – 移动文件指针

f_truncate – 截断文件

f_sync -刷新缓存信息

具体使用方法请见：http://elm-chan.org/fsw/ff/00index_e.html

2.11 DAC 模块

DAC 模块全称 Digital-to-Analog Converter，意为数字到模拟转换器。该模块的所有函数定义在 HAL_DAC.c 代码内。该模块下的函数可以配置 DAC 模块的初始化，输出指定数值的模拟电压等。

2.11.1 LPLD_DAC_Init()

DAC 模块初始化函数，在此函数中设置默认参数以及调用 LPLD_DAC_Config 配置寄存器。

函数原型

*uint8 LPLD_DAC_Init(LPLD_DAC_Cfg_t *DAC_Config);*

参数

** DAC_Config* –DAC 的全部初始化参数定义在 LPLD_DAC_Cfg_t 结构体中，用户在调用此函数的之前需要声明该结构体的变量，然后对该变量中的参数进行初始化。结构体中的全部参数见下表 **2.11.1a**，其中加**粗体**为配置时必须设置的参数。

参数名称	描述
dacx	DAC 模块号。 参数值见表 **2.11.1b**
Water_Mark_Mode	选择水印模式的字数。 参数值见表 **2.11.1c**
Buffer_Enable	配置缓冲区使能。 参数值见表 **2.11.1d**
Buffer_Mode	配置缓冲区的模式。 参数值见表 **2.11.1e**
Triger_Mode	配置触发模式。 参数值见表 **2.11.1f**
Buffer_Init_Pos	配置缓冲区的起始位置。
DAC_irqc	选择 DAC 模块中断。 参数值见表 **2.11.1g**
isr_func	设置回调函数。

表 2.11.1a

结构体 *LPLD_DAC_Cfg_t* 参数说明

dacx

参数值	描述

DAC0	选择 DAC0 模块
DAC1	选择 DAC1 模块

表 2.11.1b

Water_Mark_Mode

-默认初始值为 WATER_MODE_1WORD。

参数值	描述
WATER_MODE_1WORD	水印模式 1 字节
WATER_MODE_2WORD	水印模式 2 字节
WATER_MODE_3WORD	水印模式 3 字节
WATER_MODE_4WORD	水印模式 4 字节

表 2.11.1c

Buffer_Enable

-默认初始值为 BUFFER_DISABLE。

参数值	描述
BUFFER_DISABLE	缓冲区读指针禁用
BUFFER_ENABLE	缓冲区读指针使能

表 2.11.1d

Buffer_Mode

-默认初始值为 BUFFER_MODE_NORMAL。

参数值	描述
BUFFER_MODE_NORMAL	缓冲区正常模式
BUFFER_MODE_SWING	缓冲区摇摆模式
BUFFER_MODE_ONETIMESCAN	缓冲区单次扫描模式

表 2.11.1e

Triger_Mode

-默认初始值为 TRIGER_MODE_NONE。

参数值	描述
TRIGER_MODE_NONE	无触发模式
TRIGER_MODE_SOFTWARE	软件触发模式
TRIGER_MODE_HARDWARE	硬件触发模式

表 2.11.1f

Buffer_Init_Pos

-默认初始值为 0，取值范围 0~15。

Buffer_Up_Limit

-默认初始值为 15，取值范围 0~15。

DAC_irqc

-默认初始值为 DAC_IRQ_DISABLE。

参数值	描述
WATER_MODE_1WORD	关闭中断
DAC_IRQ_POINTER_BOTTOM	缓冲计数指针指向底时触发中断
DAC_IRQ_POINTER_TOP	缓冲计数指针指向顶时触发中断
DAC_IRQ_WATER_MARK	达到水印设置的字时触发中断

表 2.11.1g

isr_func

-默认初始值为 NULL，即无回调函数。

返回

0 为初始化失败，1 为初始化成功。

2.11.2 LPLD_DAC_Reset_Reg()

复位 DAC 通道以及各个寄存器。

函数原型

void LPLD_DAC_Reset_Reg(DACx dacx);

参数

dacx - DAC 模块号。具体参数选项见**表 2.11.1b**。

返回

无

2.11.3 LPLD_DAC_Set_Buffer()

设置 DAC 通道的缓冲区。

函数原型

void LPLD_DAC_Set_Buffer(DACx dacx, uint8 DACx_DATn, uint16 data16);

参数

dacx - DAC 模块号。具体参数选项见**表 2.11.1b**。

DACx_DATn - DAC 缓冲区号，缓冲区号取值范围是 **0~15**。

data16 - DAC 缓冲区数据。

返回

无

2.11.4 LPLD_DAC_Soft_Trig()

设置软件触发信号。

函数原型

void LPLD_DAC_Soft_Trig(DACx dacx);

参数

dacx - DAC 模块号。具体参数选项见**表 2.11.1b**。

返回

无

2.12 CAN 模块

CAN 模块全称 Controller Area NetWork，意为总线控制器局域网，Kinetis 系列中亦称为 FlexCAN 模块，有易用、灵活之意。该模块的所有函数定义在 HAL_CAN.c 代码内。该模块下的函数可以配置 K60 的 CAN 模块进行数据的收发。

2.12.1 LPLD_CAN_Init()

该函数用于 Flex_CAN 模块初始化。

函数原型

uint8 LPLD_CAN_Init(CANx canx, uint32 baud_khz, uint8 selfloop);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

参数值	描述
CAN0	选择 CAN0 模块
CAN1	选择 CAN1 模块

表 2.12.1a

baud_khz -设置 CAN 总线波特率，单位 **kHz**。

selfloop -置 CAN 总线自循环模式。具体参数选项见**表 2.12.1b**。

参数值	描述
CAN_NOSELFLOOP	不循环
CAN_SELFLOOP	循环

表 2.12.1b

返回

0 为初始化失败，1 为初始化成功。

2.12.2 LPLD_CAN_SetIsr()

该函数用于 Flex_CAN 模块的 16 个中断源配置。

函数原型

void LPLD_CAN_SetIsr(uint8 can_int_type, CAN_ISR_CALLBACK isr_func);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

can_int_type -设置相应 CAN 模块中断类型的回调函数。具体参数选项见**表 2.12.2a**。

参数值	描述
FLEXCAN_MB_INT	邮箱中断
FLEXCAN_BUS_OFF_INT	总线关闭
FLEXCAN_ERROR_INT	错误中断
FLEXCAN_TRANS_WARNING_INT	发送警告中断
FLEXCAN_RECV_WARNING_INT	接收警告中断
FLEXCAN_WAKEUP_INT	唤醒中断
FLEXCAN_IMEU_INT	独立匹配元素更新
FLEXCAN_LOST_RECV_INT	接收丢失中断

表 2.12.2a

isr_func -用户中断程序入口地址。用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)

返回

无

2.12.3 LPLD_CAN_SendData()

该函数用于 Flex_CAN 模块向总线发送数据。

函数原型

*uint8 LPLD_CAN_SendData(CANx canx, uint16 mbx, uint32 id, uint8 len, uint8 *data);*

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

mbx -设置对应的邮箱号。具体参数选项见**表 2.12.3a**。

参数值	描述
MB_NUM_0	0 号邮箱
……	……

MB_NUM_15	15 号邮箱

表 2.12.3a

id -目标位置的 ID 号。

len -送数据的字节数，最大 8 个字节。取值范围 **0~8**。

**data* -发送数据的缓冲区。

返回

0 失败，1 成功。

2.12.4 LPLD_CAN_RecvData()

该函数用于接收 Flex_CAN 模块从总线获取的数据。

函数原型

*uint8 LPLD_CAN_RecvData(CANx canx, uint16 mbx, uint32 *id, uint8 *len, uint8 *data);*

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

mbx -设置对应的邮箱号。具体参数选项见**表 2.12.3a**。

**id* -目标位置的 ID 号。

len -接收到的数据长度指针。

**data* -接收到的数据缓冲区指针。

返回

0 失败，1 成功。

2.12.5 LPLD_CAN_Enable_RX_Buf()

该函数用于使能 Flex_CAN 模块的接收缓冲区。

函数原型

void LPLD_CAN_Enable_RX_Buf(CANx canx, uint16 mbx, uint32 id);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

mbx -设置对应的邮箱号。具体参数选项见**表 2.12.3a**。

**id* -接收缓冲的 ID，用于和接收到的 ID 进行匹配。

返回

无

2.12.6 LPLD_CAN_Enable_Interrupt()

该函数用于按位使能 Flex_CAN 模块的邮箱中断。

函数原型

void LPLD_CAN_Enable_Interrupt(CANx canx, uint16 mbx);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

mbx -设置对应的邮箱号。具体参数选项见**表 2.12.3a**。

返回

无

2.12.7 LPLD_CAN_Disable_Interrupt()

该函数用于按位禁止 Flex_CAN 模块的邮箱中断。

函数原型

void LPLD_CAN_Disable_Interrupt(CANx canx, uint16 mbx);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

mbx -设置对应的邮箱号。具体参数选项见**表 2.12.3a**。

返回

无

2.12.8 LPLD_CAN_GetFlag()

该函数用于获得 Flex_CAN 模块的邮箱中断标志位。

函数原型

uint32 LPLD_CAN_GetFlag(CANx canx, uint16 mbx);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

mbx -设置对应的邮箱号。具体参数选项见**表 2.12.3a**。

返回

邮箱所对应的中断标志位。

2.12.9 LPLD_CAN_ClearFlag()

该函数用于按位清除 Flex_CAN 模块的邮箱中断标志位。

函数原型

void LPLD_CAN_ClearFlag(CANx canx, uint16 mbx);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

mbx -设置对应的邮箱号。具体参数选项见**表 2.12.3a**。

返回

无

2.12.10 LPLD_CAN_ClearAllFlag()

该函数用于清除所有 Flex_CAN 模块的邮箱中断标志位。

函数原型

void LPLD_CAN_ClearAllFlag(CANx canx);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

返回

无

2.12.11 LPLD_CAN_Unlock_MBx()

该函数用于解锁 Flex_CAN 模块中的接收邮箱，通过读取自由计数器的值。

函数原型

uint16 LPLD_CAN_Unlock_MBx(CANx canx);

参数

canx -设置 CAN 模块号。具体参数选项见**表 2.12.1a**。

返回

返自由计数器的值。

2.13 SPI 模块

SPI 模块全称 Serial Peripheral Interface，意为串行外设总线，Kinetis 系列中亦称为 DSPI 模块。该模块的所有函数定义在 HAL_SPI.c 代码内。该模块下的函数可以配置 K60 的 SPI 模块进行数据的收发。

2.13.1 LPLD_SPI_Init()

该函数用于初始化 SPI 模块。

函数原型

uint8 LPLD_SPI_Init(SPIx spix,uint8 sck_div,uint8 pcs_num);

参数

spix -设置 SPI 通道。具体参数选项见**表 2.13.1a**。

参数值	描述
SPI0	选择 SPI0 号模块
SPI1	选择 SPI1 号模块
SPI2	选择 SPI2 号模块

表 2.13.1a

sck_div -选择 SPI 模块 SCK 的分频值。具体参数选项见**表 2.13.1b**。

参数值	描述
SPI_SCK_DIV_2	2 分频
SPI_SCK_DIV_4	4 分频
SPI_SCK_DIV_6	6 分频
SPI_SCK_DIV_8	8 分频
SPI_SCK_DIV_16	16 分频
SPI_SCK_DIV_32	32 分频
SPI_SCK_DIV_64	64 分频
SPI_SCK_DIV_128	128 分频
SPI_SCK_DIV_256	256 分频
SPI_SCK_DIV_512	512 分频
SPI_SCK_DIV_1024	1024 分频
SPI_SCK_DIV_2048	2048 分频
SPI_SCK_DIV_4096	4096 分频

SPI_SCK_DIV_8192	8192 分频
SPI_SCK_DIV_16384	16384 分频
SPI_SCK_DIV_32768	32768 分频

表 2.13.1b

pcs_num -选择 SPI 片选端口的数量。具体参数选项见**表 2.13.1c**。

参数	片选数量	SPI0 初始化引脚	SPI1 初始化引脚	SPI2 初始化引脚
1	1	PCS0 PORTA14	PCS0 PORTB10	PCS0 PORTD11
2	2	PCS0 PORTA14 PCS1 PORTC3	PCS0 PORTB10 PCS1 PORTB9	PCS0 PORTD11 PCS1 PORTD15
3	3	PCS0 PORTA14 PCS1 PORTC3 PCS2 PORTC2	PCS0 PORTB10 PCS1 PORTB9 PCS2 PORTE5	
4	4	PCS0 PORTA14 PCS1 PORTC3 PCS2 PORTC2 PCS3 PORTC1	PCS0 PORTB10 PCS1 PORTB9 PCS2 PORTE5 PCS3 PORTE6	
5	5	PCS0 PORTA14 PCS1 PORTC3 PCS2 PORTC2 PCS3 PORTC1 PCS4 PORTC0		
6	6	PCS0 PORTA14 PCS1 PORTC3 PCS2 PORTC2 PCS3 PORTC1 PCS4 PORTC0 PCS5 PORTB23		

表 2.13.1c

返回

0 为初始化失败，1 为初始化成功。

2.13.2 LPLD_SPI_Master_Read()

SPI 主机读取从机数据。

函数原型

uint8 LPLD_SPI_Init(SPIx spix,uint8 sck_div,uint8 pcs_num);

参数

spix -设置 SPI 通道。具体参数选项见**表 2.13.1a**。

返回

读取从机 8 位的数据。

2.13.3 LPLD_SPI_Master_Write()

SPI 向从机写数据。

函数原型

void LPLD_SPI_Master_Write(SPIx spix,uint8 data,uint8 pcsx,uint8 pcs_state);

参数

spix -设置 SPI 通道。具体参数选项见**表 2.13.1a**。

data -要发送数据。单位为一个字节，8 位数据。

pcsx -CS 片选端口号。具体参数选项见**表 2.13.3a**。

参数值	描述
SPI_PCS0	0 号片选（SPI0、SPI1、SPI2 含有）
SPI_PCS1	1 号片选（SPI0、SPI1、SPI2 含有）
SPI_PCS2	2 号片选（SPI0、SPI1 含有）
SPI_PCS3	3 号片选（SPI0、SPI1 含有）
SPI_PCS4	4 号片选（SPI0 含有）
SPI_PCS5	5 号片选（SPI0 含有）

表 2.13.3a

pcs_state -一帧数据传输完成后 CS 的状态。具体参数选项见**表 2.13.3b**。

参数值	描述
SPI_PCS_ASSERTED	保持片选有效
SPI_PCS_INACTIVE	片选无效

表 2.13.3b

返回

无

2.13.4 LPLD_SPI_Master_WriteRead()

SPI 向从机写数据，并读取从机数据。

函数原型

uint8 LPLD_SPI_Master_WriteRead(SPIx spix,uint8 data,uint8 pcsx,uint8 pcs_state);

参数

spix -设置 SPI 通道。具体参数选项见**表 2.13.1a**。

data -要发送数据。单位为一个字节，8 位数据。

pcsx -CS 片选端口号。具体参数选项见**表 2.13.3a**。

pcs_state -一帧数据传输完成后 CS 的状态。具体参数选项见**表 2.13.3b**。

返回

读取从机 8 位的数据。

2.14 PDB 模块

PDB 模块全称 Programmable Delay Block，意为可编程延时模块。该模块的所有函数定义在 HAL_PDB.c 代码内。该模块下的函数可以产生触发源实现对 ADC 采样时间的精准控制，以及对 DAC 产生模拟信号周期的精准控制。

2.14.1 LPLD_PDB_Init()

该函数用于初始化 PDB 模块，在该函数中需要设置 PDB 模块的分频系数、分频因子、计数器的模、触发源、加载模式、工作模式和选择 DMA 模式。

函数原型

```
uint8 LPLD_PDB_Init(uint8 prescale,
                    uint8 mult,
                    uint16 mod ,
                    uint8 trgsel,
                    uint8 ldmod,
                    uint8 cont,
                    uint8 dma_en);
```

参数

prescale -设置 PDB 分频系数。具体参数选项见**表 2.14.1a**。

参数	描述
PDB_PRESC_1	设置 prescale = MULT * 1
PDB_PRESC_2	设置 prescale = MULT * 2
PDB_PRESC_4	设置 prescale = MULT * 4
PDB_PRESC_8	设置 prescale = MULT * 8
PDB_PRESC_16	设置 prescale = MULT * 16
PDB_PRESC_32	设置 prescale = MULT * 32
PDB_PRESC_64	设置 prescale = MULT * 64
PDB_PRESC_128	设置 prescale = MULT * 128

表 2.14.1a

mult -设置 PDB 分频因子。具体参数选项见**表 2.14.1b**。

参数	描述
PDB_MULT_1	设置 MULT = 1

PDB_MULT_10	设置 MULT = 10
PDB_MULT_20	设置 MULT = 20
PDB_MULT_40	设置 MULT = 40

表 2.14.1b

mod -设置 PDB 模计数器。当 PDB 计数器等于 MOD 的值，PDB 计数器清零。

trgsel -选择 PDB 触发源。具体参数选项见**表 2.14.1c**。

参数	**描述**
TRIGER_IN0	输入触发源 0
……	……
TRIGER_IN14	输入触发源 14
SOFTWARE_TRIGER	软件触发

表 2.14.1c

ldmod -设置 PDB ADC 通道 x 的预触发通道。具体参数选项见**表 2.14.1d**。

参数	**描述**
LDMOD0	当 LDOK=1 后 MOD, IDLY, CHnDLYm, INTx, 和 POyDLY 寄存器立即加载
LDMOD1	当 LDOK=1 和 PDB 计数器到达 MOD 后 MOD, IDLY, CHnDLYm, INTx, 和 POyDLY 寄存器立即加载
LDMOD2	当 LDOK=1 和一个输入事件设置为 1 后 MOD, IDLY, CHnDLYm, INTx, 和 POyDLY 寄存器立即加载
LDMOD3	当 LDOK=1 后和一个输入事件设置为 1 或 PDB 计数器到达 MOD 后 MOD, IDLY, CHnDLYm, INTx, 和 POyDLY 寄存器立即加载

表 2.14.1d

cont -设置 PDB 工作模式。具体参数选项见**表 2.14.1e**。

参数	**描述**
PDB_ONESHOT	单次模式
PDB_CONTINUE	使能 DMA 模式

表 2.14.1e

dma_en -使能 DMA 功能。具体参数选项见**表 2.14.1f**。

参数	**描述**
PDB_DMA_OFF	禁止 DMA 模式
PDB_DMA_ON	使能 DMA 模式

表 2.14.1f

返回

0 为初始化失败，1 为初始化成功。

2.14.2 LPLD_PDB_ADC_Trigger_Congfig()

该函数设置指定 ADC 模块的 PDB 触发信号、ADC 预触发延时计数器的值、预触发通道以及 ADC 预触发通道的 back to back 模式。

函数原型

```
void LPLD_PDB_ADC_Trigger_Congfig(ADCx adcx,
                                  uint16 *pre_dalay,
                                  uint8 pre_ch,
                                  uint8 pre_bb_mode);
```

参数

adcx -选择 PDB 所触发的 ADC 通道。具体参数选项见**表 2.14.2a**。

参数	描述
ADC0	选择 PDB ADC0 模块
ADC1	选择 PDB ADC1 模块

表 2.14.2a

pre_dalay* -设置 PDB ADC 的预触发延时器的计数值，该参数最大为 **65536

pre_ch -设置 PDB ADC 通道 x 的预触发通道。具体参数选项见**表 2.14.2b**。

参数	描述
PDB_PRECH_DIS	不选择预触发位
PDB_PRECH_EN0	选择预触发 0
PDB_PRECH_EN1	选择预触发 1

表 2.14.2b

pre_bb_mode -设置 PDB ADC 通道 x 的预触发的 back to back 模式。具体参数选项见**表 2.14.2c**。

参数	描述
PDB_PREBB_DIS	禁止 PDB ADC 通道 x 的预触发的 b2b 模式
PDB_PREBB_EN0	选择 PDB ADC 通道 x 的预触发 0 的的 b2b

	模式
PDB_PREBB_EN1	选择 PDB ADC 通道 x 的预触发 1 的的 b2b 模式

表 2.14.2c

返回

无

2.14.3 LPLD_PDB_DAC_Interval_Congfig()

该函数用于设置指定 DAC 模块的 PDB 触发周期，以及 PDB DAC 的外部引脚触发模式。

函数原型

```
void LPLD_PDB_DAC_Interval_Congfig(DACx dacx,
                                   uint16 interval,
                                   uint8 extrigger);
```

参数

dacx -选择 PDB DAC 的通道。具体参数选项见**表 2.14.3a**。

参数	描述
DAC0	选择 PDB DAC0 模块
DAC1	选择 PDB DAC1 模块

表 2.14.3a

interval -DAC 的触发间隔，该参数最大为 **65536**。

extrigger - DAC 外部触发输入使能。具体参数选项见**表 2.14.3b**。

参数	描述
PDB_EXTRG_DIS	禁用外部触发输入
PDB_EXTRG_EN	使能外部触发输入

表 2.14.3b

返回

无

2.14.4 LPLD_PDB_Delay()

该函数用查询标志位的方式等待延时完成。

函数原型

void LPLD_PDB_Delay(uint16 delay);

参数

无

返回

无

2.14.5 LPLD_PDB_SetUp()

该函数用于开启关闭 PDB 计数器。

函数原型

void LPLD_PDB_SetUp(uint8 is_operate);

参数

is_operate -选择 PDB DAC 的通道。具体参数选项见**表 2.14.5a**。

参数	描述
PDB_STOP	关闭 PDB
PDB_OPERATION	开启 PDB

表 2.14.5a

返回

无

2.14.6 LPLD_PDB_SetDelayIsr()

该函数用于设定 PDB 延时中断，并开启中断。

函数原型

void LPLD_PDB_SetDelayIsr(uint16 delay,PDB_ISR_CALLBACK isr_func);

参数

delay -PDB 的延时值，该参数最大为 **65536**。

isr_func -PDB 延时中断服务函数。

返回

无

2.15 ENET 模块

ENET 模块的名字即以太网（Ethernet）的英文缩写。该模块的所有函数定义在 HAL_ENET.c 代码内。该模块下的函数可以对 ENET 模块的 MAC 层、MII 接口进行初始化，可以利用 MAC 层进行以太数据通信。

2.15.1 LPLD_ENET_Init()

ENET 模块初始化，包括 PHY 收发器初始化，MAC 初始化，BD 初始化。

函数原型

void LPLD_ENET_Init(const uint8 MACAddress);*

参数

**MACAddress* - MAC 地址指针。

返回

无

2.15.2 LPLD_ENET_BDInit()

缓冲区描述符初始化。

函数原型

static void LPLD_ENET_BDInit(void);

参数

无

返回

无

2.15.3 LPLD_ENET_HashAddress()

生成给定的 MAC 地址的哈希表。

函数原型

uint8 LPLD_ENET_HashAddress(const uint8 addr);*

参数

**addr* - 6 字节地址指针。

返回

32 位 CRC 校验的高 6 位

2.15.4 LPLD_ENET_MacRecv()

以太帧接收函数。

函数原型

*uint8 LPLD_ENET_MacRecv(uint8 *ch, uint16 *len);*

参数

**ch* -接收数据帧头地址。

**len* -数据帧长度地址。

返回

无

2.15.5 LPLD_ENET_MacSend()

以太帧发送函数。

函数原型

*void LPLD_ENET_MacSend(uint8 *ch, uint16 len);*

参数

**ch* -接收数据帧头地址。

len -数据帧长度。

返回

无

2.15.6 LPLD_ENET_MiiInit()

设置 ENET 模块的 MII 接口时钟，期望频率为 2.5MHz。

MII_SPEED = 系统时钟 / (2.5MHz * 2)

函数原型

void LPLD_ENET_MiiInit(int sys_clk_mhz);

参数

sys_clk_mhz -系统主频，单位 MHz。

返回

无

2.15.7 LPLD_ENET_MiiRead()

MII 接口读。

函数原型

*uint8 LPLD_ENET_MiiRead(uint16 phy_addr, uint16 reg_addr, uint16 *data);*

参数

phy_addr -物理收发器地址。

reg_addr -寄存器地址。

**data* -读出的数据地址指针。

返回

1：读超时；0：读取成功。

2.15.8 LPLD_ENET_MiiWrite()

MII 接口写。

函数原型

uint8 LPLD_ENET_MiiWrite(uint16 phy_addr, uint16 reg_addr, uint16 data);

参数

phy_addr -物理收发器地址。

reg_addr -寄存器地址。

data -写入的数据。

返回

1：写超时；0：写入成功。

2.15.9 LPLD_ENET_SetAddress()

设置 MAC 物理地址。

函数原型

*void LPLD_ENET_SetAddress(const uint8 *pa);*

参数

**pa* - 6 字节的物理地址指针。

返回

无

2.15.10 LPLD_ENET_SetIsr()

ENET 模块中断函数设置。

函数原型

uint8 LPLD_ENET_SetIsr(uint8 type, ENET_ISR_CALLBACK isr_func);

参数

type -中断类型。具体参数选项见**表 2.15.10a**。

参数值	描述
ENET_RXF_ISR	接收中断
ENET_TXF_ISR	发送中断

表 2.15.10a

isr_func -用户中断程序入口地址。用户在工程文件下定义的中断函数名，函数必须为:无返回值,无参数(eg. void isr(void);)。

返回

0：配置错误；1：配置成功。

2.16 USB 模块

USB 是通用串行总线（Universal Serial Bus）的英文缩写。本模块调用飞思卡尔官方的 USB 驱动库，其代码在“LPLD\USB”目录下，并封装了 USB 设备模式的驱动函数，在 HAL_USB.c 文件内。本手册不对飞思卡尔官方驱动函数进行介绍。该模块下的函数可以初始化 K60 的 USB 控制器、初始化 USB 为设备模式的 CDC 类、利用 CDC 类进行数据收发。

2.16.1 LPLD_USB_Init()

K60 USB 模块初始化函数。

函数原型

void LPLD_USB_Init(void);

参数

无

返回

无

2.16.2 LPLD_USB_Device_Init()

USB 设备模式初始化函数。在该函数中初始化 USB 稳压模块，并停用 USB 稳压模块的 Standby 模式，初始化 USB 的 CDC 模式。

函数原型

void LPLD_USB_Device_Init(void);

参数

无

返回

无

2.16.3 LPLD_USB_Device_Enumed()

判断当前 CDC 设备是否被主机枚举。

函数原型

void LPLD_USB_Device_Enumed(void);

参数

无

返回

无

2.16.4 LPLD_USB_VirtualCom_Rx()

将 USB CDC 类模式虚拟成串口模式，该函数是串口接收函数。

函数原型

*uint8_t LPLD_USB_VirtualCom_Rx(uint8_t *rx_buf);*

参数

**rx_buf* -指向用户数据存储区，用于储存接收到的数据。

返回

接收数据的长度，以字节为单位。

2.16.5 LPLD_USB_VirtualCom_Tx()

将 USB CDC 类模式虚拟成串口模式，该函数是串口发送函数。

函数原型

*void LPLD_USB_VirtualCom_Tx(uint8_t *tx_buf,uint8_t len);*

参数

**tx_buf* -指向用户数据存储区，用于储存要发送的数据。

len -要发送的数据长度。

返回

无

2.17 FLASH 模块

FLASH 模块驱动主要用到 K60 的 FTFL 相关寄存器。本模块的驱动函数在 HAL_FLASH.c 文件内。该模块下的函数可以初始化 K60 的 FTFL 模块、对其片内 Flash 进行擦除、写入等操作。

2.17.1 LPLD_Flash_Init()

Flash 模块初始化。

函数原型

void LPLD_Flash_Init(void);

参数

无

返回

无

2.17.2 LPLD_Flash_SectorErase()

擦除 Flash 扇区。

函数原型

uint8 LPLD_Flash_SectorErase(uint32 FlashPtr);

参数

FlashPtr -扇区地址指针，即扇区号乘以 2048。

返回

错误代码，具体参数选项见**表 2.17.2a**。

参数值	描述
FLASH_OK	操作成功
FLASH_FACCERR	Flash 访问错误

FLASH_FPVIO	Flash 保护
FLASH_MGSTAT0	MGSTAT0 非可纠正错误
FLASH_RDCOLERR	Flash 读冲突错误
FLASH_NOT_ERASED	Flash 未擦除
FLASH_CONTENTERR	待写入内容错误

表 2.17.2a

2.17.3 LPLD_Flash_ByteProgram()

编程写入 Flash。

函数原型

*uint8 LPLD_Flash_ByteProgram(uint32 FlashStartAdd, uint32 *DataSrcPtr, uint32 NumberOfBytes);*

参数

FlashStartAdd - Flash 编程起始地址。

**DataSrcPtr* -数据源指针。

NumberOfBytes -数据字节长度。

返回

错误代码，具体参数选项见**表 2.17.2a**。