时钟说明

ACM32F4xx，ACM32FP4xx，ACM32F3xx

目录

前言

本应用笔记举例介绍了时钟相关的配置以及应用，如系统时钟的配置，包括如何利用 RCH 和 PLL 分别进行配置；RCH 相关的使用，UART 波特率设置中对 RCH 的应用，以及 CLKOUT 的说明。以解决客户在使用过程中对系统时钟配置不清晰，或由于时钟偏差造成的模块无法使用等问题。

本应用笔记主要包括两部分内容：

- 第 1 部分介绍系统时钟的配置。
- 第 2 部分介绍 RCH 时钟相关的应用。
- 第 3 部分介绍 UART 中关于 RCH 的使用。
- 第 4 部分介绍 CLKOUT 的使用。

1. 系统时钟配置

本章节介绍了如何配置系统时钟 SYS_CLK，如图所示，系统时钟 SYS_CLK 来源包括 XTL、XTH、RCH、RC32K、PLL 五种方式，PLL 可以选择 XTH 或 RCH 的 16 分频后作为输入。

时钟来源通过系统寄存器中的时钟控制寄存器 1(CCR1)进行选择。

图 1－1 系统时钟来源

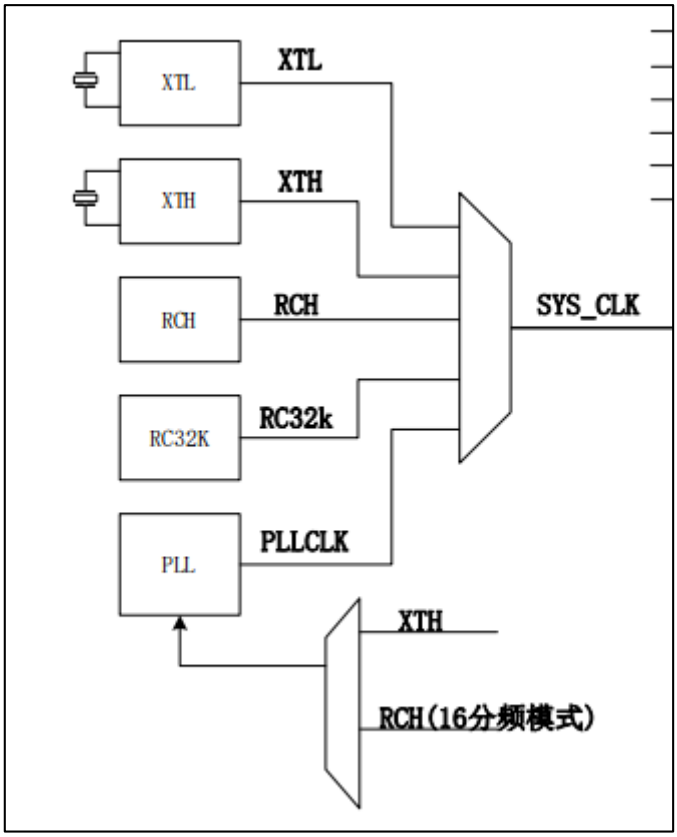


图 1－2 时钟控制寄存器

2.10.5. **时钟控制寄存器 1/CCR1(偏移：10h)**

比特	名称	属性	默认值	功能描述
31:3	RSV	-	-	保留
2:0	SYS_CLK_SEL	RW	000	系统时钟 SYS_CLK 选择： 000：来自 RCH 001：来自 RC32K 010：来自 XTH 011：来自 XTL 1xx：来自 PLLCLK

1.1 通过 RCH 配置

系统时钟 SYS_CLK 源可选择内部高速 RC 产生的 RCH 时钟，最高 64M。在航芯提供的 HAL 库中，当初始化代码中所选系统时钟<=64M 时，使用内部 RCH 时钟，默认支持 64M/32M 频率，且 PCLK=HCLK，通过系统寄存器中的时钟控制寄存器 2（CCR2）进行分频配置。

注:若需要使用 RCH 产生更小的系统时钟，如 16M/8M，请修改图中红框代码的分频因子即可。

图 1-3 通过 RCH 配置代码

```
bool System_Clock_Init(uint32_t fu32_Clock)
{
    volatile uint32_t lu32_sysdiv, lu32_pclkdiv, lu32_timeout, lu32_pll_src, lu32_pclk_div_para,lu32_result

    lu32_result = 0;
    lu32_pll_src = PLL_SOURCE_FROM;

    SET_EFC_RD_WAIT(RD_WAIT_ENSURE_OK)
    if(0 == (SCU->RCHCR & SCU_RCHCR_RCHRDY))
    {
       SCU->RCHCR |=  SCU_RCHCR_RCH_EN;
       while(0 == (SCU->RCHCR & SCU_RCHCR_RCHRDY));  // wait RCH ready
    }
    SCU->CCR1 = 0;  // select RC64M as default

    if (fu32_Clock <= 64000000)
    {
        if ((SCU->RCHCR) & SCU_RCHCR_RCH_DIV)
        {
            SCU->RCHCR &= (~SCU_RCHCR_RCH_DIV);
            while(0 == (SCU->RCHCR & SCU_RCHCR_RCHRDY));
        }

        if (fu32_Clock == 32000000)
        {
            gu32_SystemClock = fu32_Clock;
            lu32_sysdiv = 2;
            lu32_pclkdiv = 1;  // pclk = hclk
        }
        else
        {
            gu32_SystemClock = 64000000;
            lu32_sysdiv = 1;
            lu32_pclkdiv = 1;
        }
```

1.2 通过 PLL 进行配置

系统时钟 SYS_CLK 源可选择 PLL 进行配置，在航芯提供的 HAL 中，当初始化代码中所选系统时钟>64M 时，使用 PLL 进行配置，其中 PCLK=HLCK/2，如图。

图 1-4 通过 PLL 配置代码

System_ACM32F4.c

```
251     SCU->CCR1 = 0;  // select RC64M as default
252
253     if (fu32_Clock <= 64000000)
254     {
255         if ((SCU->RCHCR) & SCU_RCHCR_RCH_DIV)
256         {
257             SCU->RCHCR &= (~SCU_RCHCR_RCH_DIV);
258             while(0 == (SCU->RCHCR & SCU_RCHCR_RCHRDY));
259         }
260
261         if (fu32_Clock == 32000000)
262         {
263             gu32_SystemClock = fu32_Clock;
264             lu32_sysdiv = 2;
265             lu32_pclkdiv = 1;  // pclk = hclk
266         }
267         else
268         {
269             gu32_SystemClock = 64000000;
270             lu32_sysdiv = 1;
271             lu32_pclkdiv = 1;
272         }
273
274         gu32_Clocksrc = CLK_SRC_RCH;
275     }
276
277     else // select pll as system clock
278     {
279         lu32_result=Set_Pll_Div(fu32_Clock,lu32_pll_src);
280         lu32_sysdiv = 1;
281         lu32_pclkdiv = 2; // pclk = hclk/2
282         gu32_SystemClock = fu32_Clock;
283     }
```

1.2.1 选择 PLL 时钟源

PLL 时钟源可以选择为片内 RCH 16 分频后的时钟或片外 XTH，通过系统寄存器中的 PLL 模块控制寄存器（PLLCR）进行使能和配置。

图 1-5 PLL 时钟源选择寄存器

2:1	PLL_SRC_SEL	RW	00	PLL 时钟源选择 00：选择片内 RCH（需设置 16 分频模式） 1x：选择片外 XTH
0	PLL_EN	RW	0	PLL 模块使能 0: 不使能，PLL 模块处于 power down 状态 1: 使能，PLL 模块使能

在航芯提供的 HAL 库代码中，可直接通过 *PLL_SOURCE_FROM* 宏进行选择，可选择为 RC64M 16 分频后的 RC4M，或外部 XTH8M，外部 XTH12M，如图。

图 1－6 PLL 时钟源选择代码

```
bool System_Clock_Init(uint32_t fu32_Clock)
{
    volatile uint32_t lu32_sysdiv, lu32_pclkdiv, lu32_timeout, lu32_pll_src, lu32_pclk_div_para,lu32_result;

    lu32_result = 0;
    lu32_pll_src = PLL_SOURCE_FROM;
```

System_ACM32F4.h

```
25
26  #define PLLCLK_SRC_RC4M                 (0x00)
27  #define PLLCLK_SRC_XTH_8M               (0x01)
28  #define PLLCLK_SRC_XTH_12M              (0x02)
29
30  #define PLL_SOURCE_FROM                 (PLLCLK_SRC_RC4M)
31
```

注:若使用的外部 XTH 为其他频率，可参照直接修改代码即可。

1.2.2 配置系统时钟

在选择完 PLL 的时钟源后，即可根据输入配置 PLLCLK 作为系统时钟，通过系统寄存器中的 PLL 模块控制寄存器（PLLCR）的 PLL 分频因子进行配置，如图。

图 1－7 PLL 分频因子说明

19:16	PLL_M	RW	0001	输出分频控制字段。PLL_M 含有输出分频控制字段。对应分频值为(PLL_M+1)
14:12	PLL_N	RW	001	降频因子分频器字段。PLL_N 含有 PLL 输入时钟的分频因子，该值的实际作用是参考频率的分频因子。对应分频值为（PLL_N+1）
11:9	RSV	-	-	保留
8:3	PLL_F	RW	0x01	增频因子分频器字段。PLL_F 含有 PLL 反馈回路中分频器的分频因子，该值的实际作用是参考频率的倍频因子。对应倍频值为(PLL_F+12) PLL 输出公式为: Fpll= (Fin* (PLL_F+12)/(PLL_N+1))/ (PLL_M+1)

例如当 PLL 时钟源为 RC4M 时，若需要配置为 180M 系统时钟，则可分别将 PLL_F=33,PLL_N=0,PLL_M=0,则 Fpll=4M*(33+12)/(0+1)/(0+1)=180M。

为了简化 PLL 配置流程，航芯提供的 HAL 库中内置了一个二维数组，用于选择不同时钟源及不同的系统频率配置，无需用户自己编写。若需要配置更多的时钟频率，只需在此二维数组上添加配置即可。

图 1-8 PLL 配置二维数组

System_ACM32F4.c

```
131
132 /*******************************************************************************************
133   System_Clock_Map={PLL_SOURCE , F_SystemClock ,PLL_F,PLL_N,PLL_M }
134  * PLL_SOURCE: PLLCLK_SRC_RC4M   PLLCLK_SRC_XTH_8M   PLLCLK_SRC_XTH_12M
135  * F_SystemClock : Clock eg:180000000   120000000
136  * PLL_F       : 0-63
137  * PLL_N       : 0-7
138  * PLL_M       : 0-15
139  *******************************************************************************************/
140 const uint32_t System_Clock_Map[][5] =//Fpll=(Fin* (PLL_F+12)/(PLL_N+1))/ (PLL_M+1)
141 {
142     { PLLCLK_SRC_RC4M   ,  180000000, 33,0,0},    // Fpll=4*(33+12)/(0+1)/(0+1)=180
143     { PLLCLK_SRC_RC4M   ,  120000000, 18,0,0},    // Fpll=4*(18+12)/(0+1)/(0+1)=120
144     { PLLCLK_SRC_XTH_8M ,  180000000, 33,1,0},    // Fpll=8*(33+12)/(1+1)/(0+1)=180
145     { PLLCLK_SRC_XTH_8M ,  120000000, 18,1,0},    // Fpll=8*(18+12)/(1+1)/(0+1)=120
146     { PLLCLK_SRC_XTH_12M,  180000000, 18,1,0},    // Fpll=12*(18+12)/(1+1)/(0+1)=180
147     { PLLCLK_SRC_XTH_12M,  120000000, 18,2,0},    // Fpll=12*(18+12)/(2+1)/(0+1)=120
148     { 0xffffffff        ,         0 ,  0,0,0},    // end flag=0xffffffff
149 };
150
```

最后通过 uint32_t Set_Pll_Div(uint32_t fu32_Clock,uint32_t lu32_pll_src)函数进行配置，此函数只需指定要配置系统时钟频率 fu32_Clock 以及所用的 PLL 时钟源 lu32_pll_src 即可，代码中会自动从上述二维数组中获取要配置的系统时钟，PLL 时钟源，分频因子等对 PLLCR 和 CCR1 进行配置。

图 1-9 Set_Pll_Div 函数代码

System_ACM32F4.c

```
151 /*******************************************************************************
152  * Function    : Set_Pll_Div
153  * Description : Set Pll Div
154  * Input       : fu32_Clock: System core clock frequency, measured as Hz
155                  lu32_pll_src: PLL_SOURCE, can be PLLCLK_SRC_RC4M PLLCLK_SRC_XTH_8M PLLCLK_SRC_XTH_12M
156  * Output      : 0: success, other value: fail reason
157  * Author      : cwt
158  *******************************************************************************/
159 uint32_t Set_Pll_Div(uint32_t fu32_Clock,uint32_t lu32_pll_src)
```

图 1-10 Set_Pll_Div 函数代码

```
198      /*SET PLL DIV*/
199     for(i = 0; System_Clock_Map[i][0] != 0xffffffff; i++)
200     {
201         if(System_Clock_Map[i][0]==lu32_pll_src && System_Clock_Map[i][1]==fu32_Clock)
202         {
203             u32_pll_F=System_Clock_Map[i][2];
204             u32_pll_N=System_Clock_Map[i][3];
205             u32_pll_M=System_Clock_Map[i][4];
206 
207             SCU->PLLCR = (SCU->PLLCR &(~(0x1FFFU << 3))) | (u32_pll_F << 3) | (u32_pll_N << 12) | (u32_pll_M << 16);
208             if(lu32_pll_src==PLLCLK_SRC_XTH_8M||lu32_pll_src==PLLCLK_SRC_XTH_12M)
209             {
210                 SCU->PLLCR = (SCU->PLLCR & (~(0x3U << 1)) ) | (2 << 1); // select XTH
211             }
212             else
213             {
214                SCU->PLLCR = (SCU->PLLCR & (~(0x3U << 1)) );   // select RC64M/16
215             }
216 
217             SCU->PLLCR |= SCU_PLLCR_PLL_UPDATE_EN;
218             while(!(SCU->PLLCR & (SCU_PLLCR_PLL_FREE_RUN) ) );
219 
220             SCU->CCR1 = SCU_CCR1_SYS_PLL;  // configure system clock as PLL clock
221             u32_result=0;
222 
223             break;
224         }
225     }
226 
```

2. RCH 时钟相关

本章节介绍了如何通过 NVR 获取 RCH 的实测值，因为在实际情况中，RC64M 时钟并不是刚好等于 64M，在需要准确配置波特率的场景，如 UART 通讯时，需要知道当前 RCH 的实际值，并进行 Trim 等操作，本章将做一一介绍。

2.1. RCH 实测值获取

以下地址记录了 RC64M 的实测值，为 CP 测试后写入，定义如下：

地址	位宽	说明	备注
0x0008022C	32bit	RC64M 除以 16 后的分频值，单位为 K	常温下 CP 测试实测值，封装以及焊接后 RC64M 实际值会改变

如：读出该地址 0x0008022C 的数值为 0xF7D，则实际的 RC64M 值=0xF7D*16000=63440K。

2.2. RCH 的 Trim

当 RC64M 实测值比理论值相差过多时，可自行进行 Trim，通过系统寄存器中的 RCH 模块控制寄存器（RCHCR）进行配置，Trim 值越高，频率越高，如下图所示。

图 2－1 RCH 的 Trim 说明

7:1	RCH_TRIM[6:0]	RW	0x0	RCH 时钟 TRIM 值 TRIM 值越高频率越高
0	RCH_EN	RW	1	RCH 模块使能

3. UART 中关于 RCH 的使用

本章节介绍了在 UART 中如何根据 RCH 的实测值和理论值之间的差异对波特率进行配置，由于 RCH 并不是准确等于 64M，故采用 RCH 配置的系统时钟也存在误差，若不对 RCH 进行 Trim 校准的情况下直接使用理论系统时钟进行计算，可能导致波特率不准确，无法进行通讯。

在航芯提供的 HAL 库中，在 UART 配置波特率时，将判断当前时钟源是否来自 RCH，若来自 RCH，则会利用第二章的方法，获取 RCH 的实测值，然后跟 64M 理论值进行比例换算，根据比例系数计算出当前准确的 PCLK 时钟频率，再继续比特率配置，如图。

图 3－1 UART 波特率设置方法

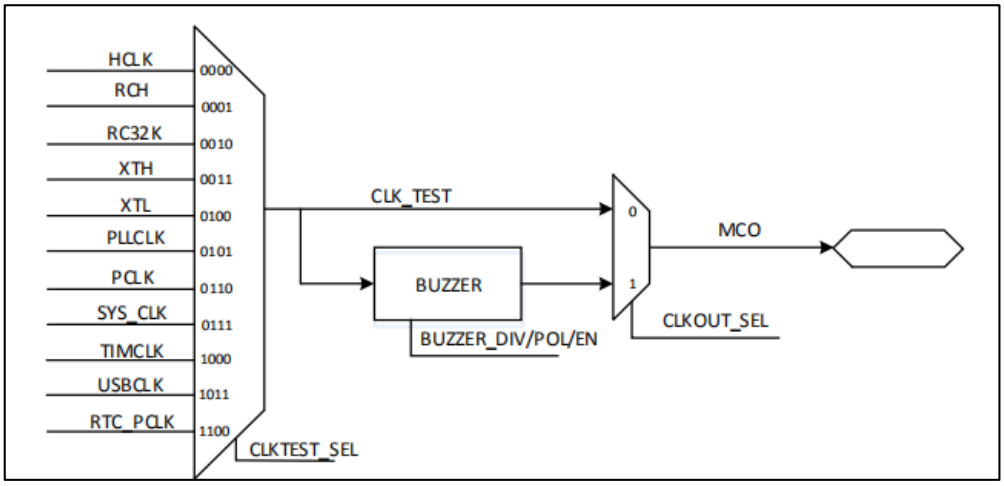
HAL_Uart.c

```
757
758 /*******************************************************************************
759  * Function    : UART_Config_BaudRate
760  * Description : Config BaudRate
761  * Input       :
762  * Output      :
763  * Author      : Chris_Kyle                          Data : 2020
764  *******************************************************************************/
765 static void UART_Config_BaudRate(UART_HandleTypeDef *huart)
766 {
767     uint32_t lu32_PCLK;
768     uint32_t lu32_IBAUD, lu32_FBAUD;
769     uint64_t lu64_TempValue;
770
771     if (CLK_SRC_RCH == System_Get_Clk_Source() )
772     {
773         uint32_t  RCH_Real= *((uint32_t *)0x8022c);
774         uint32_t  RCH_Ratio= (RCH_Real*16)/64;      // RCH_Ratio = RCH_Real/64
775         lu32_PCLK = (System_Get_APBClock()/1000)*RCH_Ratio;
776     }
777     else
778     {
779         lu32_PCLK = System_Get_APBClock();
780     }
```

4. CLKOUT 的使用

芯片的时钟输出可以通过 MCO 引脚。如图所示，输出的时钟信号可以通过 CLKTEST_SEL 选择，可以来自于系统时钟 SYS_CLK，也可以来自于其他时钟源如 XTL（低速晶振），然后可以直接输出，也可以通过一个 Buzzer 控制电路再输出。通过 Buzzer 可以将时钟进行分频，极性翻转后再输出。

图 4－1 CLKOUT 示意图



在航芯提供的 HAL 库中，内嵌了时钟输出函数，使用了 Buzzer 进行输出，默认输出时钟为 HCLK，若需要输出上文所述的其他时钟，则需用户自行修改此函数，设置 CLKOCR 的 CLK_TEST 即可。

图 4－2 时钟输出函数

System_ACM32F4.c

```
723 /*******************************************************************************
724 * Function    : System_Set_Buzzer_Divider
725 * Description : set buzzer divide factor
726 * Input       :
727               div: div factor, if div = 80 then output buzzer freq=HCLK/80
728               enable: FUNC_DISABLE and FUNC_ENABLE
729 * Output      : none
730 * Author      : xwl                          Date : 2021?ê
731 *******************************************************************************/
732 void System_Set_Buzzer_Divider(uint32_t div, FUNC_DISABLE_ENABLE enable)
733 {
734     if (FUNC_ENABLE == enable)
735     {
736         SCU->CLKOCR = (SCU->CLKOCR & (~(0x1FFFFU << 5) ) ) | (div << 5);
737         SCU->CLKOCR |= BIT23;
738     }
739     else
740     {
741         SCU->CLKOCR &= (~BIT23);
742     }
743 }
744
```

在系统初始化时，默认将 CLKOUT 关闭，Buzzer 分频默认为 80，若需使能只需修改形参即可。如要输出原始的 HCLK，则将调用修改为：

System_Set_Buzzer_Divider(79, FUNC_ENABLE); //输出即为 HCLK/(79+1)的值

图 4－3 时钟输出函数调用

System_ACM32F4.c

```
76 /*****************************************************************************************
77  * Function    : System_Init
78  * Description : Initialize the system clock, accelerate function and system tick.
79  * Input       : none
80  * Output      : none
81  * Author      : Chris_Kyle
82  *****************************************************************************************/
83 void System_Init(void)
84 {
85     SCU->RCR |= SCU_RCR_REMAP_EN;    // always remap enabled
86     System_Set_Buzzer_Divider(80, FUNC_DISABLE); // disable clock out
87     /* 3 bits for pre-emption priority, 0 bits for subpriority */
88     NVIC_SetPriorityGrouping(NVIC_PRIORITY_GROUP_3);
89
90     /* Initialize the system clock */
91     if (false == System_Clock_Init(DEFAULT_SYSTEM_CLOCK))
92     {
93         while(1);
94     }
```

联系我们

公司：上海爱信诺航芯电子科技有限公司

地址：上海市闵行区合川路 2570 号科技绿洲三期 2 号楼 702 室

邮编：200241

电话：+86-21-6125 9080

传真：+86-21-6125 9080-830

Email： Service@AisinoChip.com

Website： www.aisinochip.com

版本维护

版本	日期	作者	描述
V1.0	2022-03-03	Aisinochip	初始版