

KF8 系列 MCU 16 位常数查表方法

在应用程序中，我们经常要用到数据查表的模式，KF8 系列 MCU 带有 RRET 指令用于 8 位立即数查表，但这条指令是把指令和数据结合在一起，当待查表的元素很多时，就会占用大量的 flash 存储空间。本笔记给出了一种利用 KF8 MCU 内部资源，比较方便的查大容量数据表的实现方式。

通常情况下，假定我们定义下面一个数组

```
uint const Arr_num1[] = {0x063F, 0x4F5B, 0x6D66, 0x077D, 0x6F7F, };
```

因为使用了 const 关键字，编译器会把 Arr_num1 数组存放在 Flash 空间中，经编译器编译后在 Flash 中内存分配如下：

0000 B03F	00205	RRET R0, #0x3f
0001 B006	00206	RRET R0, #0x06
0002 B05B	00207	RRET R0, #0x5b
0003 B04F	00208	RRET R0, #0x4f
0004 B066	00209	RRET R0, #0x66
0005 B06D	00210	RRET R0, #0x6d
0006 B07D	00211	RRET R0, #0x7d
0007 B007	00212	RRET R0, #0x07
0008 B07F	00213	RRET R0, #0x7f
0009 B06F	00214	RRET R0, #0x6f

这里我们可以发现，编译器会把 16 位的 int 型数据分开为两个 8 位数据类型进行存储，从而占用了两个字的 Flash 资源，5 个字的数据总共用了 10 个字的地址空间。KF8 MCU 的 Flash 为 16 位长，可以实现一个字的 Flash 地址存储一个 16 位数据。

相比于上面的使用方式，充分利用芯片的结构，我们采用以下方式定义常数数组，如下：

```
void table_name()
{
    __asm
        .dw 0x063f
        .dw 0x4F5B
        .dw 0x6D66
        .dw 0x077D
        .dw 0x6F7F
    __endasm;
}
```

数值元素的值可以使用 16 进制形式描述，也可直接使用 10 进制数。编译后 Flash 中内存分配示例如下：

00583	.DW 0x063F
00584	.DW 0x4F5B
00585	.DW 0x6D66
00586	.DW 0x077D
00587	.DW 0x6F7F

DW 为汇编的伪指令，声明对应的地址空间为放置的数据。当需要查表时，我们可以 KF8 MCU 的内部资源读取对应地址的数据到程序中。从而实现查表的目的。读取时我们只需要知道数组的首地址和相应元素的下标，这样就实现了类似于数组的操作，读取样例如下：

```
select_tab0(); // 获取数组首地址，并存储到对应的变量中。
Read_out_int= loopuptab_small(2); //定位数组下偏移结果获取，下标 0-255。
或
Read_out_int= loopuptab_large(257); //支持更大数值，下标 0-65535
```

附件：实例程序

```
#include "main.h"
#define uchar unsigned char
#define uint unsigned int
uchar Read_out_char;
uint Read_out_int;

/*
*KF8XXX系列对应          #define LOOKUPTAB_VERSION    0
*KF8XXXX/KF8TS/KF8V系列对应  #define LOOKUPTAB_VERSION    1
*KF8S系列具体请咨询相关人员
*
*/

#define LOOKUPTAB_VERSION 0 /*芯片类型选择，

//常规整数数组的定义
uint const tabx[] = {
    0x063F,    // 16进制
    0x4F5B,
    0x6D66,
    1000,      // 10进制
    0x6F7F,
};
// 本文介绍的整型数组实现形式 数组1
void tab0()
{
    __asm
```

```
.dw 0x063f
.dw 0x4F5B
.dw 0x6D66
.dw 0x077D
.dw 0x6F7F
__endasm;
}

void select_tab0()
{
/*ChipON IDE函数传递参数使用R0 STK00-STK12，因此要求调用该函数前后的STK11 STK12不影响，否则
需要建立单独的变量，部分程序下传参仅STK00-STK11可用。
C语言默认建立 STK00 -STK12 的变量，部分代码仅到STK11，可见map表，例如参数为三个long型数据，使
用 R0, STK00-STK10,其中R0, STK00, STK01, STK02分别为第一个参数的高到低位字节结果。
*/
__asm
BANKSEL STK11 // 数组的首地址分别存放在STK11和STK12中
MOV R0,#high(_tab0) //函数数组首地址高位，C语言到汇编追加前面的“_”。
MOV STK11,R0
BANKSEL STK12
MOV R0,#low(_tab0) //函数数组首地址低位
MOV STK12,R0
__endasm;
}
// 数组2
void tab1()
{
__asm
.DW 3
.DW 2
.DW 1
.DW 0
__endasm;
}
void select_tab1()
{
/* 见选择表1说明*/
__asm
BANKSEL STK11
MOV R0, #high(_tab1) //表名函数到编译器中做了下划线处理
MOV STK11,R0
BANKSEL STK12
MOV R0, #low(_tab1) //表名函数到编译器中做了下划线处理
MOV STK12,R0
__endasm;
}
//查表函数，但查找长度限制为0-255，number为表中元素的索引，参数：R0中
unsigned int loopuptab_small(unsigned char number)
{
__asm
```

```
BANKSEL STK12
ADD R0,STK12
BANKSEL STK11
MOV R1,STK11
JNB PSW,0
INC R1

BANKSEL 0x3A
MOV 0x3A,R1
MOV 0x3B,R0
MOV R0,#0x81
MOV 0x3C,R0
NOPZ
NOPZ
NOPZ
NOPZ
#if LOOKUPTAB_VERSION == 0
    BANKSEL STK00
    MOV STK00,R6
    MOV R0,R7
#elif LOOKUPTAB_VERSION == 1
    BANKSEL 0x39
    MOV R0,0x39
    BANKSEL STK00
    MOV STK00,R0
    BANKSEL 0x38
    MOV R0,0x38
#else
    #error "unknow LOOKUPTAB_VERSION"
#endif
__endasm;
}
//查表函数，但查找长度支持0-65535，number为表中元素的索引，参数：高位R0和低位STK00
unsigned int loopuptab_large(unsigned int number)
{
    __asm
        BANKSEL STK00
        MOV R1,STK00
        BANKSEL STK12
        ADD R1,STK12
        JNB PSW,0
        INC R0
        BANKSEL STK11
        ADD R0, STK11

        BANKSEL 0x3A
        MOV 0x3A,R1
        MOV 0x3B,R0
        MOV R0,#0x81
        MOV 0x3C,R0
        NOPZ
```

```
NOPZ
NOPZ
NOPZ
#if LOOKUPTAB_VERSION == 0
    BANKSEL STK00
    MOV     STK00,R6
    MOV     R0,R7
#elif LOOKUPTAB_VERSION == 1
    BANKSEL 0x39
    MOV     R0,0x39
    BANKSEL STK00
    MOV     STK00,R0
    BANKSEL 0x38
    MOV     R0,0x38
#else
    #error "unknow LOOKUPTAB_VERSION"
#endif
    __endasm;
}
// 初始化函数
void Init_fun()
{
    OSCCTL = 0x60; //选择工作时钟
    /*****端口初始化*****/
    TR0 = 0x0B;
    TR1 = 0x00;
    TR2 = 0x00;
    TR3 = 0x00;
    P0 = 0;
    P1 = 0;
    P2 = 0;
    P3 = 0;
}
// 主循环
void main()
{
    Init_fun();
    {
        // uint 常规查表结果实现形式
        Read_out_int= tabx[1];
        //本文介绍的特殊处理模式 节省代码存在空间的实现形式
        select_tab1();
        Read_out_int= loopuptab_small(2);// 作用Read_out_int= tab1[2];

        select_tab0();
        Read_out_int= loopuptab_small(4);// 作用Read_out_int= tab0[4];
    }
    while(1);
}
```