GigaDevice Semiconductor Inc.

基于 GD32 MCU 的 Dhryston 移植指南

应用笔记

AN164

1.0 版本

（2023 年 10 月）

目录

图索引

表索引

1. 前言

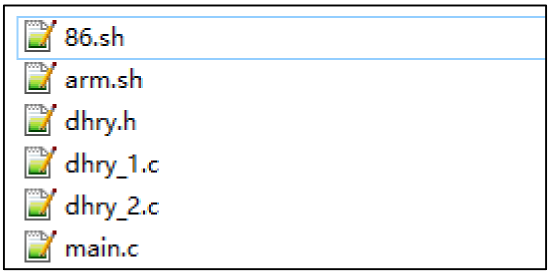
Dhrystone 是测量 MCU 运算能力的最常见的基准测试程序之一，其主要目的是测试 MCU 的整数运算和逻辑运算的性能。

Dhrystone 的测试原理是在单位时间内，测试 MCU 运行了多少次 Dhrystone 程序，测试结果指标用单位 DMIPS/MHz 表示。DMIPS 是 Dhrystone Million Instructions Per Second 的缩写，表示每秒处理的百万级机器语言指令数。

2. Dhrystone 源码

Dhrystone 源代码没有官方的下载渠道，但是网上源程序的下载来源也很多。我们从网上下载到 Version 2.1 的 C 语言版本，在本地解压打开后源文件如*图 2-1. Dhrystone 源文件*所示：

图 2-1. Dhrystone 源文件



3. Dhrystone 移植

3.1. 工程配置

本 AN 以 GD32L23x，keil5 为例，介绍 Dhrystone 的移植与注意事项。由于 GD32L23x 为 ARM Cortex-M23 内核，测试 Dhrystone 的文件为 dhry.h，dhry_1.c，dhry_2.c。

在工程路径下新建一个 Dhrystone 的文件夹，并将 dhry.h，dhry_1.c，dhry_2.c 三个文件拷贝过去，如*图 3-1. 工程目录结构*，*图 3-2. Dhrystone 文件*所示：

图 3-1. 工程目录结构

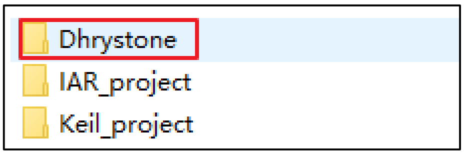
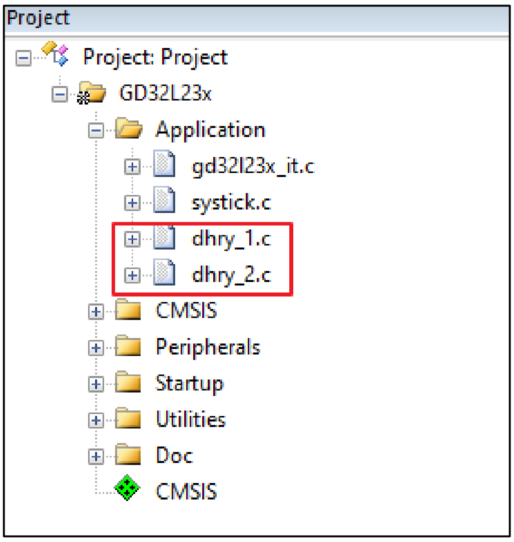


图 3-2. Dhrystone 文件

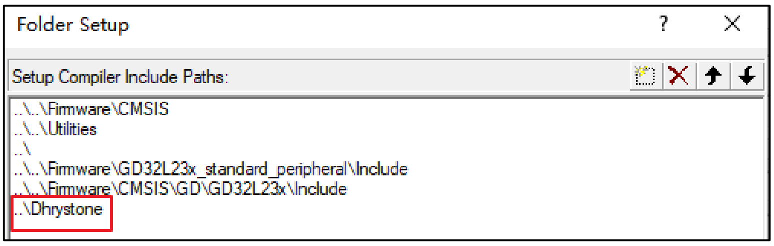
打开工程，将 dhry_1.c，dhry_2.c 添加到工程中，如*图 3-3. Dhrystone 工程结构*所示：

图 3-3. Dhrystone 工程结构



添加文件包含路径，如*图 3-4. Dhrystone 工程文件路径配置*所示：

图 3-4. Dhrystone 工程文件路径配置



3.2. 代码修改

MCU 运行 Dhrystone 测试代码需要计时和打印必要的信息，这里选择 TIMER1，TIMER2 和 USART1（供参考，可根据实际情况进行选择），配置 TIMER，USART 和对应的 GPIO 代码如*表 3-1. 时钟配置*，*表 3-2. USART 和 GPIO 配置*，*表 3-3. TIMER 配置*所示：

表 3-1. 时钟配置

```
void rcu_configuration(void)
{
    rcu_periph_clock_enable(RCU_USART1);
    rcu_periph_clock_enable(RCU_GPIOA);
    rcu_periph_clock_enable(RCU_TIMER1);
    rcu_periph_clock_enable(RCU_TIMER2);
}
```

表 3-2. USART 和 GPIO 配置

```
void usart_config(void)
{
    /* connect port to USARTx_Tx */
    gpio_af_set(GPIOA, GPIO_AF_7, GPIO_PIN_2);

    /* connect port to USARTx_Rx */
    gpio_af_set(GPIOA, GPIO_AF_7, GPIO_PIN_3);

    /* configure USART Tx as alternate function push-pull */
    gpio_mode_set(GPIOA, GPIO_MODE_AF, GPIO_PUPD_PULLUP, GPIO_PIN_2);
    gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_10MHZ, GPIO_PIN_2);

    /* configure USART Rx as alternate function push-pull */
    gpio_mode_set(GPIOA, GPIO_MODE_AF, GPIO_PUPD_PULLUP, GPIO_PIN_3);
    gpio_output_options_set(GPIOA, GPIO_OTYPE_PP, GPIO_OSPEED_10MHZ, GPIO_PIN_3);

    /* USART configure */
    usart_deinit(USART1);
    usart_baudrate_set(USART1, 115200U);
    usart_receive_config(USART1, USART_RECEIVE_ENABLE);
```

```
    usart_transmit_config(USART1, USART_TRANSMIT_ENABLE);
    usart_enable(USART1);
}
```

表 3-3. TIMER 配置

```
void timer_config(void)
{
    timer_parameter_struct timer_initpara;

    /* deinit TIMER */
    timer_deinit(TIMER1);
    timer_deinit(TIMER2);
    /* initialize TIMER init parameter struct */
    timer_struct_para_init(&timer_initpara);
    /* TIMER1 configuration */
    timer_initpara.prescaler            = ((clk/1000000)*2 -1);
    timer_initpara.alignedmode          = TIMER_COUNTER_EDGE;
    timer_initpara.counterdirection = TIMER_COUNTER_UP;
    timer_initpara.period               = 9999;
    timer_initpara.clockdivision        = TIMER_CKDIV_DIV1;
    timer_init(TIMER1, &timer_initpara);

    /* TIMER2 configuration */
    timer_initpara.prescaler            = 0;
    timer_initpara.alignedmode          = TIMER_COUNTER_EDGE;
    timer_initpara.counterdirection = TIMER_COUNTER_UP;
    timer_initpara.period               = 9999;
    timer_initpara.clockdivision        = TIMER_CKDIV_DIV1;
    timer_init(TIMER2, &timer_initpara);

    timer_master_slave_mode_config(TIMER1, TIMER_MASTER_SLAVE_MODE_ENABLE);
    timer_master_output_trigger_source_select(TIMER1, TIMER_TRI_OUT_SRC_UPDATE);

    timer_master_slave_mode_config(TIMER2, TIMER_MASTER_SLAVE_MODE_ENABLE);
    timer_slave_mode_select(TIMER2, TIMER_SLAVE_MODE_EXTERNAL0);
    timer_input_trigger_source_select(TIMER2, TIMER_SMCFG_TRGSEL_ITI0);

    /* enable TIMER */
    timer_enable(TIMER1);
    timer_enable(TIMER2);
}
```

在 dhry_1.c 文件的 main 函数需要做一些修改，main 函数修改后如***表 3-4. main 函数修改***所示：

表 3-4. main 函数修改

```
int main (void)
/*****/

  /* main program, corresponds to procedures        */
  /* Main and Proc_0 in the Ada version             */
{
        One_Fifty       Int_1_Loc;
  REG   One_Fifty       Int_2_Loc;
        One_Fifty       Int_3_Loc;
  REG   char            Ch_Index;
        Enumeration     Enum_Loc;
        Str_30          Str_1_Loc;
        Str_30          Str_2_Loc;
  REG   int             Run_Index;
  REG   int             Number_Of_Runs;

  clk = rcu_clock_freq_get(CK_APB1);

  Next_Ptr_Glob = (Rec_Pointer) malloc (sizeof (Rec_Type));
  Ptr_Glob = (Rec_Pointer) malloc (sizeof (Rec_Type));

  Ptr_Glob->Ptr_Comp                    = Next_Ptr_Glob;
  Ptr_Glob->Discr                       = Ident_1;
  Ptr_Glob->variant.var_1.Enum_Comp     = Ident_3;
  Ptr_Glob->variant.var_1.Int_Comp      = 20;
  strcpy (Ptr_Glob->variant.var_1.Str_Comp,
        "DHRYSTONE PROGRAM, SOME STRING");
  strcpy (Str_1_Loc, "DHRYSTONE PROGRAM, 1'ST STRING");

  Arr_2_Glob [8][7] = 10;

  /* Was missing in published program. Without this statement,    */
  /* Arr_2_Glob [8][7] would have an undefined value.             */
  /* Warning: With 16-Bit processors and Number_Of_Runs > 32000,  */
  /* overflow may occur for this array element.                   */

  rcu_configuration();
  usart_config();

  printf ("Dhrystone Benchmark, Version 2.1 (Language: C)\n\r");

  Number_Of_Runs = 1000000;
```

```
printf ("Execution starts, %d runs through Dhrystone\n\r", Number_Of_Runs);

timer_config();

Begin_Time = timer_counter_read(TIMER2)*10000 + timer_counter_read(TIMER1);
for (Run_Index = 1; Run_Index <= Number_Of_Runs; ++Run_Index)
{

    Proc_5();
    Proc_4();
    /* Ch_1_Glob == 'A', Ch_2_Glob == 'B', Bool_Glob == true */
    Int_1_Loc = 2;
    Int_2_Loc = 3;
    strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
    Enum_Loc = Ident_2;
    Bool_Glob = ! Func_2 (Str_1_Loc, Str_2_Loc);
    /* Bool_Glob == 1 */
    while (Int_1_Loc < Int_2_Loc)   /* loop body executed once */
    {
        Int_3_Loc = 5 * Int_1_Loc - Int_2_Loc;
        /* Int_3_Loc == 7 */
        Proc_7 (Int_1_Loc, Int_2_Loc, &Int_3_Loc);
        /* Int_3_Loc == 7 */
        Int_1_Loc += 1;
    } /* while */
    /* Int_1_Loc == 3, Int_2_Loc == 3, Int_3_Loc == 7 */
    Proc_8 (Arr_1_Glob, Arr_2_Glob, Int_1_Loc, Int_3_Loc);
    /* Int_Glob == 5 */
    Proc_1 (Ptr_Glob);
    for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)
    /* loop body executed twice */
    {
        if (Enum_Loc == Func_1 (Ch_Index, 'C'))
        /* then, not executed */
        {
            Proc_6 (Ident_1, &Enum_Loc);
            strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 3'RD STRING");
            Int_2_Loc = Run_Index;
            Int_Glob = Run_Index;
        }
    }
    /* Int_1_Loc == 3, Int_2_Loc == 3, Int_3_Loc == 7 */
    Int_2_Loc = Int_2_Loc * Int_1_Loc;
```

```
        Int_1_Loc = Int_2_Loc / Int_3_Loc;
        Int_2_Loc = 7 * (Int_2_Loc - Int_3_Loc) - Int_1_Loc;
        /* Int_1_Loc == 1, Int_2_Loc == 13, Int_3_Loc == 7 */
        Proc_2 (&Int_1_Loc);
        /* Int_1_Loc == 5 */

    } /* loop "for Run_Index" */

    /**************/
    /* Stop timer */
    /**************/
    timer_disable(TIMER1);
    timer_disable(TIMER2);
    End_Time = timer_counter_read(TIMER2)*10000 + timer_counter_read(TIMER1);
    User_Time = End_Time - Begin_Time;

    Dhrystones_Per_Second = (double) Number_Of_Runs / (User_Time / 1000000);
    Vax_Mips = Dhrystones_Per_Second / 1757.0;

    printf ("Run time is: %6.6f \n\r", User_Time/1000000);
    printf ("Dhrystones per Second: %6.1f \n\r", Dhrystones_Per_Second);
    printf ("Vax_Mips is: %6.1f \n", Vax_Mips);
    printf ("\n");
    printf("MCU CK_SYS frequency is: %d\n\r%d\n\r", rcu_clock_freq_get(CK_AHB));
    printf("DMIPS/MHz is: %f \n", (double)Vax_Mips / (rcu_clock_freq_get(CK_AHB)/1000000));
    while(1);
}
```

4. 测试结果

本文档以代码运行在 Flash 中为例，分别设置 Number_Of_Runs 为 500000 和 10000000，其中 Dhrystones_Per_Second = 运行次数 / 运行时间，Vax_Mips = Dhrystones_Per_Second / 1757.0， DMIPS/MHz = Vax_Mips / 主频。对应的测试结果如图 4-1. Dhrystone 测试 1，图 4-2. Dhrystone 测试 2所示：

图 4-1. Dhrystone 测试 1

```
Dhrystone Benchmark, Version 2.1 (Language: C)

Execution starts, 500000 runs through Dhrystone

Run time is: 14.289064

Dhrystones per Second: 34991.8

Vax_Mips is: 19.915649

MCU CK_SYS frequency is: 64000000

DMIPS/MHz is: 0.311182
```

图 4-2. Dhrystone 测试 2

```
Dhrystone Benchmark, Version 2.1 (Language: C)

Execution starts, 1000000 runs through Dhrystone

Run time is: 28.578126

Dhrystones per Second: 34991.8

Vax_Mips is: 19.915649

MCU CK_SYS frequency is: 64000000

DMIPS/MHz is: 0.311182
```

5. 版本历史

表 5-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2023 年 10 月 30 日