STM32 Cordic 运算速度评估

1. 引言

客户在使用 Cordic 进行运算时候，对 Cordic 打断 CPU 的时间存有疑问，认为时间不是按照芯片手册中所描述的时钟周期，本文针对 Cordic 时间测试用于澄清计算周期，同时可作为客户评估使用，本文以 STM32G431 作为示例。

2. 执行周期

在 STM32G431 参考手册中有关于 Cordic 的精度及迭代次数描述，另外还有执行周期的对应值，从图上我们了解到设定精度数据等同于指令执行时间，因为 Cordic 执行时会自动耗损 CPU 执行周期，因此我们可以了解执行速度等同于精度数据设定。

图 1. Cordic 精度与执行周期

Function	Number of iterations	Number of cycles	Max residual error(1)	
			q1.31 format	q1.15 format
Sin, Cos, Phase(2), Mod, Atan(4)	4	1	2^{-3}	2^{-3}
	8	2	2^{-7}	2^{-7}
	12	3	2^{-11}	2^{-11}
	16	4	2^{-15}	2^{-15}
	20	5	2^{-18}	2^{-16}
	24	6	2^{-19}	2^{-16}

3. 测试方法

我们分两种写法：

1.单次执行测试

2.多次测试取平均值

测试代码中 hcordic.Instance->也可以写为 CORDIC->；读取 TIM3 的寄存器数据就可以知道这段代码执行的时间，其中 TIM3 配置时钟为 170MHz 等同于 CPU 执行时间；TIM3 的 CNT 数据即指令执行周期。

3.1. 单次执行测试

当使用单次测试时，我们使用 16 次迭代测试，精度配置字为 4；

图 2.单次执行时间测试代码

```
hcordic.Instance->CSR = 0x00000041;

TIM3->CNT = 0;
TIM3->CR1 |= 0x01; /* START TIM3 */

hcordic.Instance->WDATA = Angle;    /* Write angle */
sine_temp = hcordic.Instance->RDATA; /* Read result */

TIM3->CR1 &= 0xFFFFFFFE;   /* STOP TIM3 */

/* Get Cycle number*/
Cycle[Count++] = TIM3->CNT;
```

图 3.单次执行时间汇编代码

```
Disassembly
0x10000014 6808      LDR           r0,[r1,#0x00]
0x10000016 F0400001  ORR           r0,r0,#0x01
0x1000001A 6008      STR           r0,[r1,#0x00]
   334:          hcordic.Instance->WDATA = Angle;        /* Write angle */
0x1000001C 4812      LDR           r0,[pc,#72]  ; @0x10000068
0x1000001E 6816      LDR           r6,[r2,#0x00]
0x10000020 6885      LDR           r5,[r0,#0x08]
0x10000022 6075      STR           r5,[r6,#0x04]
   335:          sine_temp = hcordic.Instance->RDATA; /* Read result */
   336:
0x10000024 6812      LDR           r2,[r2,#0x00]
0x10000026 6892      LDR           r2,[r2,#0x08]
   337:          TIM3->CR1 &= 0xFFFFFFFE;        /* STOP TIM3 */
   338:
   339:          /* Get Cycle number*/
0x10000028 680D      LDR           r5,[r1,#0x00]
0x1000002A F0250501  BIC           r5,r5,#0x01
0x1000002E 600D      STR           r5,[r1,#0x00]
```

图 4.单次指令执行周期观测

Watch 1

Name	Value	Type
Cycle	0x200000D8 Cycle	ushort[20]
[0]	24	ushort
[1]	24	ushort
[2]	24	ushort
[3]	24	ushort
[4]	24	ushort
[5]	24	ushort
[6]	24	ushort

图 5.单次指令执行分析：

```
 334:        hcordic.Instance->WDATA = Angle;       /* Write angle */
0x1000001C 4812      LDR          r0,[pc,#72]  ; @0x10000068 --- 3 Cycle
0x1000001E 6816      LDR          r6,[r2,#0x00] --- 2 Cycle
0x10000020 6885      LDR          r5,[r0,#0x08] --- 2 Cycle
0x10000022 6075      STR          r5,[r6,#0x04] --- 1 Cycle
 335:        sine_temp = hcordic.Instance->RDATA; /* Read result */
 336:
0x10000024 6812      LDR          r2,[r2,#0x00] --- 2 Cycle
0x10000026 6892      LDR          r2,[r2,#0x08] --- 3 Cycle
 337:        TIM3->CR1 &= 0xFFFFFFFE;       /* STOP TIM3 */
 338:
 339:        /* Get Cycle number*/
0x10000028 680D      LDR          r5,[r1,#0x00] --- 5 Cycle
0x1000002A F0250501  BIC          r5,r5,#0x01 --- 1 Cycle
0x1000002E 600D      STR          r5,[r1,#0x00] --- 1 Cycle
```

去除数据 load 设定时间以及 TIM3 执行停止指令时间，可以看到真正 Cordic 运算正弦运算的指令执行周期为：

24cycle -（3+2+2+1+2+3+5+1+1）cycle = 4 cycle；

也就是在写入 WDATA 和读取 RDATA 之间的 CPU 占用时间为 4 个 CPU 时间。

3.2. 多次执行测试

我们使用 24 次迭代测试，精度配置字为 6；循环计算 100 次，计算平均值：

图 6. 多次执行测试 C 代码

```
__attribute__((section (".ccmram")))
void Cordic_SinTime_Test1(void)
{
  static int32_t sine_temp[100];
  hcordic.Instance->CSR = 0x00000061;
  TIM3->CNT = 0;
  TIM3->CR1 |= 0x01; /* START TIM3 */

  for(int16_t i = 0; i<100;i++)
  {
    hcordic.Instance->WDATA = i;  /* Write angle */
    sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
  }

  TIM3->CR1 &= 0xFFFFFFFE;  /* STOP TIM3 */

  /* Get Cycle number*/
  Cycle[Count++] = TIM3->CNT;
```

图 7. 多次执行汇编代码

```
   334:          for(int16_t i = 0; i<100;i++)
   335:          {
   336:              hcordic.Instance->WDATA = i;    /* Write angle */
0x1000001C 4618       MOV           r0,r3
   337:              sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
   338:          }
   339:
0x1000001E F5A174C8   SUB           r4,r1,#0x190
   336:              hcordic.Instance->WDATA = i;    /* Write angle */
   337:              sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
   338:          }
   339:
0x10000022 680E       LDR           r6,[r1,#0x00]
   336:              hcordic.Instance->WDATA = i;    /* Write angle */
0x10000024 6070       STR           r0,[r6,#0x04]
   337:              sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
0x10000026 680E       LDR           r6,[r1,#0x00]
   337:              sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
0x10000028 68B6       LDR           r6,[r6,#0x08]
0x1000002A F8446020   STR           r6,[r4,r0,LSL #2]
0x1000002E 1C40       ADDS          r0,r0,#1
   334:          for(int16_t i = 0; i<100;i++)
0x10000030 B200       SXTH          r0,r0
   334:          for(int16_t i = 0; i<100;i++)
0x10000032 2864       CMP           r0,#0x64
   334:          for(int16_t i = 0; i<100;i++)
0x10000034 DBF5       BLT           0x10000022
```

图 8. 多次执行周期观测

Watch 1

Name	Value	Type
Cycle	0x20000264 Cycle	ushort[20]
[0]	2008	ushort
[1]	2008	ushort
[2]	2008	ushort
[3]	2008	ushort
[4]	2008	ushort
[5]	2008	ushort
[6]	2008	ushort
[7]	2008	ushort
[8]	2008	ushort
[9]	2008	ushort
[10]	2008	ushort

图 9. 多次执行指令分析

```
  334:        for(int16_t i = 0; i<100;i++)
  335:        {
  336:              hcordic.Instance->WDATA = i;    /* Write angle */
0x1000001C 4618      MOV          r0,r3  ---1Cycle
  337:              sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
  338:        }
  339:
0x1000001E F5A174C8  SUB          r4,r1,#0x190 ---1Cycle
  336:              hcordic.Instance->WDATA = i;    /* Write angle */
  337:              sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
  338:        }
  339:
0x10000022 680E      LDR          r6,[r1,#0x00]---2Cycle
  336:              hcordic.Instance->WDATA = i;    /* Write angle */
0x10000024 6070      STR          r0,[r6,#0x04] ---1Cycle
  337:              sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
0x10000026 680E      LDR          r6,[r1,#0x00] ---2Cycle
  337:              sine_temp[i] = hcordic.Instance->RDATA; /* Read result */
0x10000028 68B6      LDR          r6,[r6,#0x08]---3Cycle
0x1000002A F8446020  STR          r6,[r4,r0,LSL #2]---2Cycle
0x1000002E 1C40      ADDS         r0,r0,#1---1Cycle
  334:        for(int16_t i = 0; i<100;i++)
0x10000030 B200      SXTH         r0,r0---1Cycle
  334:        for(int16_t i = 0; i<100;i++)
0x10000032 2864      CMP          r0,#0x64---1Cycle
  334:        for(int16_t i = 0; i<100;i++)
0x10000034 DBF5      BLT          0x10000022---1Cycle
  340:        TIM3->CR1 &= 0xFFFFFFFE;       /* STOP TIM3 */
  341:
  342:        /* Get Cycle number*/
0x10000036 6810      LDR          r0,[r2,#0x00]---5Cycle
  340:        TIM3->CR1 &= 0xFFFFFFFE;       /* STOP TIM3 */
0x10000038 F0200001  BIC          r0,r0,#0x01---1Cycle
0x1000003C 6010      STR          r0,[r2,#0x00]
```

循环体 for 执行周期为 2+1+2+3+2+1+1+1+1=14，TIM3 执行停止指令 8Cycle

这样单次的 Cordic 运算周期为：

$$\frac{2008 - 8 - 14 * 100}{100} = 6\ cycles$$

4. 结论

从上面的测试可很清楚看到 Cordic 指令执行时间严格遵循精度设定，执行时间可以说非常的短。当客户使用 Cordic 时候往往需要考虑的是数据的 load 以及 set 时间，以及 C 语言的写法问题，这边还是推荐直接操作寄存器的方式时间最短。

参考文献

RM0440 — STM32G4 Reference manual（V6）

文档中所用到的工具及版本

CubeMx V6.1
MDK ARM V5.32

版本历史

日期	版本	变更
2022 年 04 月 11 日	1.0	首版发布

重要通知 – 请仔细阅读

重要通知 **-** 请仔细阅读 意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。若需 ST 商标的更多信息，请参考 *www.st.com/trademarks*。所有其他产品或服务名称均为其 各自所有者的财产。

本LAT文档是ST本地团队在支持客户时的经验总结，是对官方文档的一个补充以供参考，如与ST.COM官网的内容相冲突，则以官网内容为准。如有任何因为此文档（包括代码例子，电路图例子）而造成生产、服务等各个环节的损失恕不负责。

本文档中的信息取代本文档所有早期版本中提供的信息。

© 2020 STMicroelectronics - 保留所有权利