# GigaDevice Semiconductor Inc.

# GD32G5x3 三角函数加速器 TMU 的使用指南

应用笔记

**AN207**

1.0 版本

（2024 年 5 月）

# 目录

# 表索引

# 1. 前言

本文是专门为基于GD32 MCU开发的工程设计人员提供，主要介绍了GD32G5x3系列器件内三角函数加速器（TMU）的内部结构、功能配置、以及使用时的注意事项，旨在帮助GD32 MCU开发者正确快速的使用三角函数加速器，缩短开发周期。

三角函数加速器（TMU）是一个完全可配置的单元，可执行常见的三角运算和算术运算操作。TMU 可以减轻 CPU 的负担，通常应用于电机控制，信号处理和很多其他应用场景。

GD32G5x3 系列三角函数加速器（TMU）主要特征有：

- 支持 10 种函数；
- 具有中断和 DMA 请求；
- 可配置的数据格式：q1.31、q1.15 定点格式和 IEEE754 32 位单精度浮点格式；
- 可编程的计算精度；
- CORDIC算法核：支持圆周系统和双曲线系统，支持旋转模式和向量模式。

# 2. TMU 简介

GD32G5x3 系列器件的三角函数加速器（TMU）通过 CORDIC（Coordinated Rotation Digital Computer，坐标旋转数字计算方法）求解三角函数和其他复杂的数学函数。CORDIC 算法通过迭代逼近的方式旋转向量，将旋转操作分解为一系列微小的旋转步骤，每一步都只涉及简单的移位和加减运算，非常适合硬件实现。

TMU_CS 寄存器的 ITRTNUM[3:0]位域用来配置 CORDIC 算法核的迭代次数。迭代次数越高，计算精度越高。

## 2.1. 模式配置

在圆周系统下，CORDIC 算法可以求解三角函数，圆周系统又分为旋转模式和向量模式，旋转模式用于计算 cos 和 sin，向量模式用于计算 atan2 和模长值。

在双曲系统下，CORDIC 算法可以求解一系列超越函数，在旋转模式下，可以求解 cosh 和 sinh，进而可以计算$e^x = \sinh(x) + \cosh(x)$。在向量模式下，可以实现$\tanh^{-1}$、对数函数和开方运算。

TMU_CS 寄存器的 MODE[3:0]位域用来配置 CORDIC 算法核模块的运行模式。不同的模式的输入和输出数据参考*表 2-1. TMU 模式配置*。

**表 2-1. TMU 模式配置**

| 模式 | 第一个输入数据 | 第二个输入数据 | 第一个输出数据 | 第二个输出数据 | 使用的系统和模式 |
|---|---|---|---|---|---|
| 模式 0 | θ | m | m*cos(θ) | m*sin(θ) | 圆周系统，旋转模式 |
| 模式 1 | θ | m | m*sin(θ) | m*cos(θ) | 圆周系统，旋转模式 |
| 模式 2 | x | y | atan2 (y,x) | $\sqrt{x^2+y^2}$ | 圆周系统，向量模式 |
| 模式 3 | x | y | $\sqrt{x^2+y^2}$ | atan2 (y,x) | 圆周系统，向量模式 |
| 模式 4 | x | 无 | $\tan^{-1}(x)$ | 无 | 圆周系统，向量模式 |
| 模式 5 | x | 无 | cosh (x) | sinh (x) | 双曲线系统，旋转模式 |
| 模式 6 | x | 无 | sinh (x) | cosh (x) | 双曲线系统，旋转模式 |
| 模式 7 | x | 无 | $\tanh^{-1}(x)$ | 无 | 双曲线系统，向量模式 |
| 模式 8 | x | 无 | ln (x) | 无 | 双曲线系统，向量模式 |
| 模式 9 | x | 无 | $\sqrt{x}$ | 无 | 双曲线系统，向量模式 |

尽管 TMU 算法仅能够直接计算少量的函数，但更多的函数可以通过间接的方法来获得。

## 2.2. 数据格式配置

GD32G5x3 系列器件的三角函数加速器（TMU）的输入、输出数据可配置为 q1.31、q1.15 定点格式和 IEEE754 32 位单精度浮点格式。

当输入数据格式为定点时（IFLTEN=0），可以通过 TMU_CS 寄存器中的 IWIDTH 位配置输入数据的定点格式。当输入数据格式为浮点时（IFLTEN=1），IWIDTH 位的配置无效。每个模式需要的输入数据的个数有所不同（例如，模式 0 需要两个输入数据，模式 5 只需要一个），可以通过 TMU_CS 寄存器的 INUM 位配置输入数据的数量。详细配置参考*表 2-2. 输入数据配置*。

**注意：**当输入数据配置为 q1.15 格式，如果所配模式需要两个输入数据，只需要写一次 TMU_IDATA 寄存器，第一个输入数据在低半字，第二个输入数据在高半字。如果所配模式只需要一个输入数据，则只使用低半字，高半字的不使用。

**表 2-2. 输入数据配置**

| IWIDTH 位 | INUM 位 | IFLTEN 位 | 数据格式 | 写 TMU_IDATA 寄存器 |
|---|---|---|---|---|
| 0 | 0 | 0 | q1.31 定点 | 写一次 |
| 0 | 1 | 0 | q1.31 定点 | 连续写两次 |
| 1 | 0 | 0 | q1.15 定点 | 写一次 |
| 1 | 1 | 0 | q1.15 定点 | 不可用 |
| X | 0 | 1 | IEEE754 32 位单精度浮点 | 写一次 |
| X | 1 | 1 | | 连续写两次 |

当输出数据格式为定点时（OFLTEN=0），可以通过 TMU_CS 寄存器的 OWIDTH 位配置输出数据的定点格式。当输出数据格式为浮点时（OFLTEN=1），OWIDTH 位的配置无效。每个模式的输出数据的个数有所不同（例如，模式 0 有两个输出数据，模式 8 只有一个），可以通过 TMU_CS 寄存器的 ONUM 位配置输出数据的数量。详细配置参考*表 2-3. 输出数据配置*。

**注意：**当输出数据配置为 q1.15 格式，如果所配模式有两个输出数据，只需要读一次 TMU_ODATA 寄存器。第一个输出数据在低半字，第二个输出数据在高半字。如果所配模式只有一个输出，则只使用低半字，不使用高半字。

**表 2-3. 输出数据配置**

| OWIDTH 位 | ONUM 位 | OFLTEN 位 | 数据格式 | 读 TMU_ODATA 寄存器 |
|---|---|---|---|---|
| 0 | 0 | 0 | q1.31 定点 | 读一次 |
| 0 | 1 | 0 | q1.31 定点 | 连续读两次 |
| 1 | 0 | 0 | q1.15 定点 | 读一次 |
| 1 | 1 | 0 | q1.15 定点 | 不可用 |
| X | 0 | 1 | IEEE754 32 位单精度浮点 | 读一次 |
| X | 1 | 1 | | 连续读两次 |

# 3. 代码配置

三角函数加速器（TMU）的输入、输出数据可以通过软件和 DMA 两种方式进行写入和读取。本章将基于这两种方式，分别介绍如何配置 TMU 模块，进行函数运算。

## 3.1. 软件写入、读取数据例程

以模式 1 为例，计算角度的正弦值m*sin(θ)，当输入、输出数据都为 q1.31 定点格式（IFLTEN=0 且 OFLTEN=0）时，TMU 模块配置见*表 3-1. TMU 模块配置（q1.31 数据格式，不使用 DMA）*所示。

客户可以根据需要的精度配置迭代次数，这里配置了 24 次；模式 1 中缩放因子 FACTOR[2:0]不可用；当输入、输出数据均通过软件直接写入和读取时，DMA 读、写请求均配置为禁能；模式 1 具有两个输入数据，两个输出数据，当输入数据为 q1.31 定点格式时，需要连续写入两次，当输出数据为 q1.31 定点格式时，需要连续读取两次。

**表 3-1. TMU 模块配置（q1.31 数据格式，不使用 DMA）**

```
/* enable the clock */
rcu_periph_clock_enable(RCU_TMU);

tmu_parameter_struct tmu_init_struct;
tmu_struct_para_init(&tmu_init_struct);
/* reset the TMU */
tmu_deinit();
/* configure TMU peripheral */
tmu_init_struct.mode = TMU_MODE_SIN;
tmu_init_struct.iterations_number = TMU_ITERATION_STEPS_24;
tmu_init_struct.scale = TMU_SCALING_FACTOR_1;
tmu_init_struct.output_floating = TMU_OUTPUT_FLOAT_DISABLE;
tmu_init_struct.input_floating = TMU_INPUT_FLOAT_DISABLE;
tmu_init_struct.dma_read = TMU_READ_DMA_DISABLE;
tmu_init_struct.dma_write = TMU_WRITE_DMA_DISABLE;
tmu_init_struct.read_times = TMU_READ_TIMES_2;
tmu_init_struct.write_times = TMU_WRITE_TIMES_2;
tmu_init_struct.output_width = TMU_OUTPUT_WIDTH_32;
tmu_init_struct.input_width = TMU_INPUT_WIDTH_32;
tmu_init(&tmu_init_struct);
```

当输入、输出数据为定点格式，而需要计算的原始数据为浮点类型时，需要进行数据类型的转换，转化为 q1.31 格式的数据后，连续写入 TMU_IDATA 寄存器。同样的，输出数据也可以转换为浮点类型。

另外，当输入或输出数据为定点格式，支持的数据范围较小，当 m 超出了数据范围时，需要先缩小到[0,1)范围内，再写入到寄存器中，最后对输出数据进行相应比例的放大。

**表 3-2. 软件写入、读取数据（q1.31 数据格式）**

```
/* the first input data: angle */
float theta = DEMO_PI / 2.0f;
/* the second input data: modulus */
float m = 100.0f;
/* the calculation results: result=m*sin(theta)*/
float result = 0;
/* TMU input data in q31 format */
uint32_t in_data_q31[2] = {0};
/* TMU output data in q31 format */
uint32_t out_data_q31[2] = {0};

uint16_t scaling_factor = 128U;

/* software processes the input data */
in_data_q31[0] = (uint32_t)((int32_t)(theta / DEMO_PI * 0x80000000U));
in_data_q31[1] = (uint32_t)((int32_t)(m / scaling_factor * 0x80000000U));

/* configure TMU input and output data in q31 format */
tmu_config_q31();
/* write data to start TMU */
tmu_two_q31_write(in_data_q31[0], in_data_q31[1]);
/* read two output data */
tmu_two_q31_read(&out_data_q31[0], &out_data_q31[1]);

/* software processes the output data */
result = scaling_factor * (float)((int32_t)out_data_q31[0]) / 0x80000000U;
```

当输入、输出数据都为 IEEE754 32 位单精度浮点格式（IFLTEN=1 且 OFLTEN=1）时，TMU 模块配置与*表 3-1. TMU 模块配置（q1.31 数据格式，不使用 DMA）*基本一致，只需使能输入、输出数据的单精度浮点格式即可，见*表 3-3. TMU 模块配置（单精度浮点数据格式）*所示。模式 1 具有两个输入数据，两个输出数据，当输入、输出数据为单精度浮点格式时，需要连续写入、读取两次。

**表 3-3. TMU 模块配置（单精度浮点数据格式）**

```
tmu_init_struct.output_floating = TMU_OUTPUT_FLOAT_ENABLE;
tmu_init_struct.input_floating = TMU_INPUT_FLOAT_ENABLE;
```

当输入和输出数据均为单精度浮点格式时，支持的数据范围较大，当输入数据不超过用户手册中描述的范围时，直接写入浮点数据到 TMU_IDATA 寄存器中即可，同样的，直接读取 TMU_ODATA 寄存器即可得到计算结果。

当输入和输出数据均为单精度浮点格式时，如果输入数据或者运算中的数据按照既定浮点格式显得过大会产生上溢，在这种情景下，TMU_CS 寄存器的 OVRF 标志位会置位。

**表 3-4. 软件写入、读取数据（单精度浮点数据格式）**

```
/* the first input data: angle */
float theta = DEMO_PI / 2.0f;
/* the second input data: modulus */
float m = 100.0f;
/* the calculation results: result=m*sin(theta)*/
float result = 0;
/* TMU output data in floating point format */
float out_data_f32[2] = {0};

/* configure TMU input and output data in floating point format */
tmu_config_f32();
/* write data to start TMU */
tmu_two_f32_write(theta / DEMO_PI, m);
/* read two output data */
tmu_two_f32_read(&out_data_f32[0], &out_data_f32[1]);
```

## 3.2. DMA 写入、读取数据例程

除了软件写入外，TMU 还支持通过 DMA 模块写入和读取数据。当 TMU_CS 的 WDEN 位置 1 时，无 TMU 运算挂起时，将会产生 DMA 写请求，当 RDEN 位置 1 时，在 TMU 运算结束后，将会产生 DMA 读请求。DMA 模块的配置见*表 3-5. DMA 模块配置*所示，当数据均为单精度浮点格式时，使用浮点数据的地址替换 DMA 传输中的内存地址即可。

模式 1 具有两个输入数据，两个输出数据，当输入、输出数据为 q1.31 定点格式或者单精度浮点格式时，需要连续写入、读取两次两次，因此 DMA 读、写通道传输的宽度为 32bit，数量均为 2。

**表 3-5. DMA 模块配置**

```
/* enable DMA1 clock */
rcu_periph_clock_enable(RCU_DMA1);
/* enable DMAMUX clock */
rcu_periph_clock_enable(RCU_DMAMUX);

tmu_parameter_struct tmu_init_struct;
/* TMU write request: DMA1 channel */
dma_struct_para_init(&dma_init_struct);
dma_deinit(DMA1, DMA_CH1);
dma_init_struct.request = DMA_REQUEST_TMU_INPUT;
dma_init_struct.direction = DMA_MEMORY_TO_PERIPHERAL;
dma_init_struct.memory_addr = (uint32_t)in_data_q31;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = 2;
```

```
dma_init_struct.periph_addr = TMU_WRITE_ADDRESS;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_DISABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_32BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA1, DMA_CH1, &dma_init_struct);
dma_circulation_disable(DMA1, DMA_CH1);

/* TMU read request: DMA1 channe2 */
dma_struct_para_init(&dma_init_struct);
dma_deinit(DMA1, DMA_CH2);
dma_init_struct.request = DMA_REQUEST_TMU_OUTPUT;
dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)out_data_q31;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_32BIT;
dma_init_struct.number = 2;
dma_init_struct.periph_addr = TMU_READ_ADDRESS;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_DISABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_32BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA1, DMA_CH2, &dma_init_struct);
/* configure DMA mode */
dma_circulation_disable(DMA1, DMA_CH2);

/* enable DMA transfer */
dma_flag_clear(DMA1, DMA_CH1, DMA_FLAG_FTF);
dma_flag_clear(DMA1, DMA_CH2, DMA_FLAG_FTF);
dma_channel_enable(DMA1, DMA_CH1);
dma_channel_enable(DMA1, DMA_CH2);
```

使用 DMA 时，TMU 模块配置与*软件写入、读取数据例程*基本一致，只需使能读、写 DMA 请求即可。

**表 3-6. TMU 模块配置（使用 DMA）**

```
tmu_init_struct.dma_read = TMU_READ_DMA_ENABLE;
tmu_init_struct.dma_write = TMU_WRITE_DMA_ENABLE;
```

当输入和输出数据均为单精度浮点格式时，数据范围较大，当输入数据不超过用户手册中描述的范围时，直接写入浮点数据到 TMU_IDATA 寄存器中即可，同样的，直接读取 TMU_ODATA 寄存器即可得到计算结果。

使用 DMA 时，在配置好 DMA 和 TMU 模块后，等待 DMA 传输完成后读取结果然后处理，见*表 3-7. DMA 写入、读取数据（q1.31 数据格式）*和*表 3-8. DMA 写入、读取数据（单精度浮点数据格式）*所示。

**表 3-7. DMA 写入、读取数据（q1.31 数据格式）**

```
/* the first input data: angle */
float theta = DEMO_PI / 2.0f;
/* the second input data: modulus */
float m = 100.0f;
/* the calculation results: result=m*sin(theta)*/
float result = 0;
/* TMU input data in q31 format */
uint32_t in_data_q31[2] = {0};
/* TMU output data in q31 format */
uint32_t out_data_q31[2] = {0};

uint16_t scaling_factor = 128U;

/* software processes the input data */
in_data_q31[0] = (uint32_t)((int32_t)(theta / DEMO_PI * 0x80000000U));
in_data_q31[1] = (uint32_t)((int32_t)(m / scaling_factor * 0x80000000U));

/* configure TMU input and output data in q31 format */
tmu_config_q31();
/* configure DMA when TMU in q31 format */
dma_config_q31();
/* wait for the DMA transfer complete */
while(RESET == dma_flag_get(DMA1, DMA_CH2, DMA_FLAG_FTF));
result = scaling_factor * (float)((int32_t)out_data_q31[0]) / 0x80000000U;
```

**表 3-8. DMA 写入、读取数据（单精度浮点数据格式）**

```
/* the first input data: angle */
float theta = DEMO_PI / 2.0f;
/* the second input data: modulus */
float m = 100.0f;
/* the calculation results: result=m*sin(theta)*/
float result = 0;
/* TMU output data in floating point format */
float out_data_f32[2] = {0};

/* software processes the input data */
in_data_f32[0] = theta / DEMO_PI;
in_data_f32[1] = m;
/* configure TMU input and output data in floating point format */
tmu_config_f32();
/* configure DMA when TMU in floating point format */
dma_config_f32();
/* wait for the DMA transfer complete */
```

```
while(RESET == dma_flag_get(DMA1, DMA_CH2, DMA_FLAG_FTF));
```

# 4. 版本历史

表 4-1. 版本历史

| 版本号. | 说明 | 日期 |
| --- | --- | --- |
| 1.0 | 首次发布 | 2024 年 5 月 25 日 |

## Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.