GigaDevice Semiconductor Inc.

GD32G5x3 Trigonometric Math Unit User Guide

Application Note

AN207

Revision 1.0

(May. 2024)

Table of Contents

List of Tables

1. Preface

This document is specifically designed for engineers developing with GD32 MCUs. It mainly introduces the functional configuration and usage precautions of the trigonometric math unit (TMU) in the GD32G5x3 series devices. The aim is to help GD32 MCU developers use the TMU correctly and quickly, thereby shortening the development cycle.

The TMU is a fully configurable block that execute common trigonometric and arithmetic operations. The TMU can reduce the burden of CPU, and it is usually used in motor control, signal processing and many other applications.

Characteristics of GD32G5x3 series TMU：

- Supports 10 kinds of functions.
- Interrupt and DMA requests.
- The fixed point q1.31 / q1.15 format or IEEE754 32-bit single precision floating-point format is configurable.
- Programmable precision.
- CORDIC-algorithm core: circular system and hyperbolic system, and supports rotation pattern and vectoring pattern.

2. TMU introduction

The TMU in the GD32G5x3 series solves trigonometric and other complex mathematical functions through the CORDIC (Coordinated Rotation Digital Computer) algorithm. The CORDIC algorithm approximates vector rotations iteratively, breaking the rotation operation into a series of small rotation steps, each involving only simple shifts and additions or subtractions, making it very suitable for hardware implementation.

The ITRTNUM[3:0] bit field of the TMU_CS register is used to configure the number of iterations for the CORDIC algorithm core. The more iterations, the higher the calculation accuracy.

2.1. Mode configuration

Under the circular system, the CORDIC algorithm can solve trigonometric functions. The circular system is divided into rotation mode and vectoring mode; the rotation mode is used to calculate cos and sin, while the vectoring mode is used to calculate atan2 and modulus.

Under the hyperbolic system, the CORDIC algorithm can solve a series of transcendental functions. In rotation mode, it can solve cosh and sinh, and thus calculate $e^x = \sinh(x) + \cosh(x)$. In vectoring mode, it can perform inverse $\tanh^{-1}$, logarithmic functions, and square root operations.

The MODE[3:0] bit-field in TMU_CS register is used to configure the mode of the CORDIC algorithm core. Different modes use different systems (circular or hyperbolic) and different patterns (rotation or vectoring). Detailed configuration refer to ***Table 2-1. TMU mode configuration***.

Table 2-1. TMU mode configuration

Mode	The first input data	The second input data	The first output data	The second output data	System and Pattern
Mode 0	θ	m	m*cos(θ)	m*sin(θ)	Circular, Rotation
Mode 1	θ	m	m*sin(θ)	m*cos(θ)	Circular, Rotation
Mode 2	x	y	atan2 (y,x)	$\sqrt{x^2+y^2}$	Circular, Vectoring
Mode 3	x	y	$\sqrt{x^2+y^2}$	atan2 (y,x)	Circular, Vectoring
Mode 4	x	None	$\tan^{-1}(x)$	None	Circular, Vectoring
Mode 5	x	None	cosh (x)	sinh (x)	Hyperbolic, Rotation
Mode 6	x	None	sinh (x)	cosh (x)	Hyperbolic, Rotation
Mode 7	x	None	$\tanh^{-1}(x)$	None	Hyperbolic, Vectoring
Mode 8	x	None	ln (x)	None	Hyperbolic, Vectoring
Mode 9	x	None	$\sqrt{x}$	None	Hyperbolic, Vectoring

Although TMU algorithm can only calculate a small number of functions directly, more functions can be obtained indirectly. For example, $e^x = \sinh(x) + \cosh(x)$.

2.2. Data format and configuration

The input and output data of the TMU in the GD32G5x3 series devices can be configured in q1.31, q1.15 fixed-point format, and IEEE754 32-bit single-precision floating-point format.

When the input data is in fixed point format (IFLTEN=0), the IWIDTH bit in TMU_CS register is used to configure the fixed point format of the input data. When the input data is in floating-point format (IFLTEN=1), the configuration of the IWIDTH bit is invalid. Each mode requires a different number of input data (for example, mode 0 requires two inputs and mode 5 requires only one), and can be configured via the INUM bit of the TMU_CS register. Detailed configuration refer to ***Table 2-2. Input data configuration***.

Note: When the input data is configured in q1.15 format, the TMU_IDATA register only needs to be written once, the first input data in the low half word, the second input data in the high half word. If the mode only needs one input data, only the low half word is used, and the high half word is not used.

Table 2-2. Input data configuration

IWIDTH bit	INUM bit	IFLTEN bit	Data format	Write operation to TMU_IDATA
0	0	0	q1.31 fixed point	Only one write operation
0	1	0	q1.31 fixed point	Two successive write operation
1	0	0	q1.15 fixed point	Only one write operation
1	1	0	q1.15 fixed point	Not available
X	0	1	IEEE754 32-bit single precision floating-point	Only one write operation
X	1	1		Two successive write operation

When the output data is in fixed point format (OFLTEN=0), the OWIDTH bit in TMU_CS register is used to configure the fixed point format of the output data. When the output data is in floating-point format (OFLTEN=1), the configuration of the OWIDTH bit is invalid. Each mode requires a different number of output data (for example, mode 0 has two output datas and mode 8 only has one), and can be configured via the ONUM bit of the TMU_CS register. Detailed configuration refer to ***Table 2-3. Output data configuration***.

Note: When the output data is configured in q1.15 format, the TMU_IDATA register only needs to be read once, the first output data in the low half word, the second output data in the high half word. If the mode only needs one output data, only the low half word is used, and the high half word is not used.

Table 2-3. Output data configuration

OWIDTH bit	ONUM bit	OFLTEN bit	Data format	Read operation to TMU_ODATA
0	0	0	q1.31 fixed point	Only one read operation
0	1	0	q1.31 fixed point	Two successive read operation

OWIDTH bit	ONUM bit	OFLTEN bit	Data format	Read operation to TMU_ODATA
1	0	0	q1.15 fixed point	Only one read operation
1	1	0	q1.15 fixed point	Not available
X	0	1	IEEE754 32-bit single precision floating-point	Only one read operation
X	1	1		Two successive read operation

3. Code configuration

The input and output data of the TMU can be written and read through software and DMA. This chapter introduces how to configure the TMU module for function calculations based on these two methods.

3.1. Software write and read data example

Taking Mode 1 as an example, to calculate the sine value m*sin(θ) when the input and output data are both in q1.31 fixed-point format (IFLTEN=0 and OFLTEN=0), the TMU module configuration is shown in ***Table 3-1. TMU module configuration (q1.31 data format, DMA not used)***.

Customers can configure the number of iterations according to the required accuracy. 24 iterations are configured in this example. In Mode 1, the scaling factor FACTOR[2:0] is not available. When the input and output data are directly written and read through software, both the DMA read and write requests are disabled. Mode 1 has two input data and two output data. When the input data is in q1.31 fixed-point format, it needs to be written twice consecutively. When the output data is in q1.31 fixed-point format, it needs to be read twice consecutively.

Table 3-1. TMU module configuration (q1.31 data format, DMA not used)

```
/* 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);
```

When the datas to be calculated are in floating-point type, data type conversion is needed.

After converting to q1.31 format, write consecutively to the TMU_IDATA register. Similarly, the output data can also be converted to floating-point type.

Additionally, when the input or output data is in fixed-point format and the supported data range is small, if m exceeds the data range, it needs to be scaled down to the [0,1) range before writing to the register, and then the output data is proportionally scaled up.

Table 3-2. Software write and read data (q1.31 data format)

```
/* 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;
```

When the input and output data are in IEEE754 32-bit single precision floating-point format (IFLTEN=1 and OFLTEN=1), the TMU module configuration is basically the same as that in ***Table 3-1. TMU module configuration (q1.31 data format, DMA not used)***, just enable the single-precision floating point format of the input and output data, shown in ***Table 3-3. TMU module configuration (floating-point data format)***. Mode 1 has two input data and two output data. When the input and output data are in single-precision floating-point format, it needs to be written and read twice continuously.

Table 3-3. TMU module configuration (floating-point data format)

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

When the input and output data are both in IEEE754 32-bit single precision floating-point format (IFLTEN=1 and OFLTEN=1), a large range of data is supported. When the input data does not exceed the range described in the user manual, the floating-point data can be directly written to the TMU_IDATA register. Similarly, the calculation results can be obtained by reading the TMU_ODATA register directly. If the input data or the data in the operation is too large according to the specified floating-point format, the OVRF flag bit of the TMU_CS register will be set.

Table 3-4. Software write and read data (floating-point data format)

```
/* 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 write and read data example

In addition to software write, the TMU also supports writing and reading data through the DMA module. When the WDEN bit of the TMU_CS is set to 1, a DMA write request is generated when there is no TMU operation pending. When the RDEN bit is set to 1, a DMA read request is generated after the TMU operation is completed. The DMA module configuration is shown in ***Table 3-5. DMA module configuration***. When the data is in IEEE754 32-bit single precision floating-point format, replace the memory address in the DMA transfer with the floating-point data address.

Mode 1 has two input data and two output data. When the input and output data are in q1.31 fixed-point format or 32-bit single precision floating-point format, they need to be written and read twice consecutively. Therefore, the width of the DMA read and write channels is 32 bits, and the quantity is 2.

Table 3-5. DMA module configuration

```
/* 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);
```

When DMA is used, the TMU module configuration is basically the same as ***Table 3-1. TMU module configuration (q1.31 data format, DMA not used)***, only needs to enable read and write DMA requests.

Table 3-6. TMU module configuration (DMA used)

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

When the input and output data are both in single-precision floating-point format and do not exceed the range described in the user manual, directly write the floating-point data to the TMU_IDATA register and read the TMU_ODATA register to get the calculation result.

After configuring the DMA and TMU modules, wait for the DMA transfer to complete before reading the results and processing them, as shown in ***Table 3-7. DMA write and read data (q1.31 data format)*** and ***Table 3-8. DMA write and read data (floating-point data format)***.

Table 3-7. DMA write and read data (q1.31 data format)

```
/* 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;
```

Table 3-8. DMA write and read data (floating-point data format)

```
/* 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. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	May.25 2024