Timer/Counter Driver API Reference

Technical Note

FPGA-TN-02425-1.0

June 2026

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor's inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice's inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Contents 3
Abbreviations in This Document 5
1. Introduction 6
1.1. Purpose 6
1.2. Audience 6
1.3. Driver Versioning 6
1.3.1. Driver Version 6
1.4. Driver and IP Compatibility 6
2. API Description 7
2.1. gp_timer_init() 7
2.2. gp_timer_config() 7
2.3. gp_timer_get_status() 8
2.4. gp_timer_get_snapshot() 8
2.5. gp_timer_enable_interrupt() 8
2.6. gp_timer_disable_interrupt() 8
2.7. gp_timer_start() 8
2.8. gp_timer_stop() 9
2.9. gp_timer_isr() 9
3. Function Call Flow Diagrams 10
3.1. gp_timer_init() 10
3.2. gp_timer_config() 11
3.3. gp_timer_get_status() 12
3.4. gp_timer_get_snapshot() 13
3.5. gp_timer_enable_interrupt() 14
3.6. gp_timer_disable_interrupt() 15
3.7. gp_timer_start() 16
3.8. gp_timer_stop() 17
3.9. gp_timer_isr() 18
4. API Data Structures 19
4.1. struct gp_timer_callback_entry 19
4.2. struct gp_timer_instance 19
5. API Macros 20
References 22
Technical Support Assistance 23
Revision History 24

Figures

Figure 3.1. uint8_t gp_timer_init() 10
Figure 3.2. uint8_t gp_timer_config() 11
Figure 3.3. uint8_t gp_timer_get_status() 12
Figure 3.4. uint8_t gp_timer_get_snapshot() 13
Figure 3.5. uint8_t gp_timer_enable_interrupt() 14
Figure 3.6. uint8_t gp_timer_disable_interrupt() 15
Figure 3.7. uint8_t gp_timer_start() 16
Figure 3.8. uint8_t gp_timer_stop() 17
Figure 3.9. void gp_timer_isr() 18

Tables

Table 4.1. gp_timer_callback_entry Parameters 19
Table 4.2. gp_timer_instance Parameters 19
Table 5.1. API Macros Description 20

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviations	Definition
API	Application Programming Interface
CPU	Central Processing Unit
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface
GP	General Purpose
IP	Intellectual Property
ISR	Interrupt Service Routine

1. Introduction

The Timer/Counter IP provides timeout tracking mechanism for the system and triggers a CPU interrupt whenever a timeout event occurs. Implemented in Verilog, the Timer/Counter IP can be configured and generated through the Lattice Propel™ Builder software. It supports all Lattice device families and is built using the Lattice Radiant™ or Lattice Diamond™ toolchains, with synthesis handled by the integrated Synopsys Synplify Pro® tool.

Refer to the Timer/Counter IP User Guide (FPGA-IPUG-02139) for more details about the IP.

1.1. Purpose

This document is intended to act as a reference guide for developers by providing details of the C language driver APIs and function call flows.

1.2. Audience

The intended audience for this document includes embedded system designers and embedded software developers using Lattice devices. The technical guide assumes readers have expertise in embedded systems and FPGA technologies.

1.3. Driver Versioning

1.3.1. Driver Version

GP_TIMER_DRV_VER “26.01.00”

1.4. Driver and IP Compatibility

Driver Version	IP Version
26.01.00	1.5.0

Refer to the Timer/Counter IP Release Notes (FPGA-RN-02022) for more information on the driver and IP versions.

2. API Description

The GP Timer driver API provides initialization, configuration, control, and interrupt handling for hardware timers. The gp_timer_init() function sets up a timer instance with its base address, clock source, and available timer count, while gp_timer_config() allows per-timer configuration of prescaler options, counting mode, direction, and period. Runtime information is retrieved using gp_timer_get_status() for status flags and gp_timer_get_snapshot() for the current counter value. Interrupt behavior is controlled through gp_timer_enable_interrupt() and gp_timer_disable_interrupt(). Timer operation is managed by gp_timer_start(), which also registers a timeout callback and user context, and gp_timer_stop(), which halts the timer. All interrupt-driven events are handled through the gp_timer_isr() routine, which dispatches callbacks upon timeout.

2.1. gp_timer_init()

This API initializes the hardware base address and the number of timer channels implemented in the IP.

```
uint8_t gp_timer_init(struct gp_timer_instance *this_timer, uint32_t base_address,
uint32_t number_of_timers)
```

In/Out	Parameter	Description	Returns
In	this_timer	Pointer to the timer instance structure.	1: Success 0: Failure
In	base_addr	Base address of the timer peripheral.	
In	number_of_timers	Specifies the number of timers configured in the IP GUI. The module supports a minimum of 1 timer and a maximum of 8 timers.	

2.2. gp_timer_config()

This API configures a specific timer channel using the provided parameters, including the selected timer channel number (ranging from 1 up to the number of timers implemented), and operational settings such as prescaler behavior, where disabling the prescaler applies no clock scaling, and enabling it divides the clock according to the specified prescaler_value (0–31). The function also sets the timer mode, either continuous or one-shot, the counting direction (up or down), and the period/reload value. It returns SUCCESS (1) if the configuration is applied successfully or FAILURE (0) if an invalid parameter or configuration error occurs.

```
uint8_t gp_timer_config(struct gp_timer_instance *this_timer, uint32_t timer, bool
is_prescaler_disable, bool is_continuous, uint32_t prescaler_value, bool dir, uint32_t
period)
```

In/Out	Parameter	Description	Returns
In	this_timer	Pointer to the timer instance structure.	1: Success 0: Failure
In	timer	Timer channel number (1 to number_of_timers).	
In	is_prescaler_disable	If true, no scaling; if false, timer is divided accordingly to prescaler_value.	
In	is_continuous	If true, timer runs continuously; if false, one-shot mode.	
In	prescaler_value	Clock prescaler value (0–31), where the clock divisor value is ($2^{prescaler\_value} \times 2$).	
In	dir	Count direction is either IS_COUNT_UP (1) or IS_COUNT_DOWN (0).	
In	period	Timer period or reload value.	

2.3. gp_timer_get_status()

This API gets the status register value for a specific timer channel.

```
uint8_t gp_timer_get_status(struct gp_timer_instance *this_timer, uint8_t timer,
uint32_t *timer_status)
```

In/Out	Parameter	Description	Returns
In	this_timer	Pointer to the timer instance structure.	1: Success 0: Failure
In	timer	Timer channel number.	
Out	timer_status	Pointer to store the status register value.	

2.4. gp_timer_get_snapshot()

This API gets the current snapshot (counter value) of a specific timer channel.

```
uint8_t gp_timer_get_snapshot(struct gp_timer_instance *this_timer, uint8_t timer,
uint32_t *snapshot)
```

In/Out	Parameter	Description	Returns
In	this_timer	Pointer to the timer instance structure.	1: Success 0: Failure
In	timer	Timer channel number.	
Out	snapshot	Pointer to store the current counter value.	

2.5. gp_timer_enable_interrupt()

This API enables the interrupt for a specific timer channel.

```
uint8_t gp_timer_enable_interrupt(struct gp_timer_instance *this_timer, uint8_t timer)
```

In/Out	Parameter	Description	Returns
In	this_timer	Pointer to the timer instance structure.	1: Success 0: Failure
In	timer	Timer channel number.	

2.6. gp_timer_disable_interrupt()

This API disables the interrupt for a specific timer channel.

```
uint8_t gp_timer_disable_interrupt(struct gp_timer_instance *this_timer, uint8_t timer)
```

In/Out	Parameter	Description	Returns
In	this_timer	Pointer to the timer instance structure.	1: Success 0: Failure
In	timer	Timer channel number.	

2.7. gp_timer_start()

This API starts a specific timer channel with a callback function. For more details on the callback function structure, refer to the struct gp_timer_callback_entry section.

```
uint8_t gp_timer_start(struct gp_timer_instance *this_timer, uint8_t timer, void
(*timer_callback)(void *), void *context)
```

In/Out	Parameter	Description	Returns
In	this_timer	Pointer to the timer instance structure.	1: Success

In/Out	Parameter	Description	Returns
In	timer	Timer channel number.	0: Failure
In	void (*timer_callback)(void *)	Function pointer to be called on timer timeout.	
In	void *context	User-defined context pointer passed to callback.	

2.8. gp_timer_stop()

This API stops a specific timer channel.

```
uint8_t gp_timer_stop(struct gp_timer_instance *this_timer, uint8_t timer)
```

In/Out	Parameter	Description	Returns
In	this_timer	Pointer to the timer instance structure.	1: Success 0: Failure
In	timer	Timer channel number.	

2.9. gp_timer_isr()

This API implements the Interrupt Service Routine (ISR) for the GP Timer module and executes the callback function that is registered during the gp_timer_start() call whenever a timer interrupt occurs.

```
uint8_t gp_timer_isr(void *ctx)
```

In/Out	Parameter	Description	Returns
In	ctx	Pointer to the timer instance (passed as void* for ISR compatibility).	None

3. Function Call Flow Diagrams

3.1. gp_timer_init()

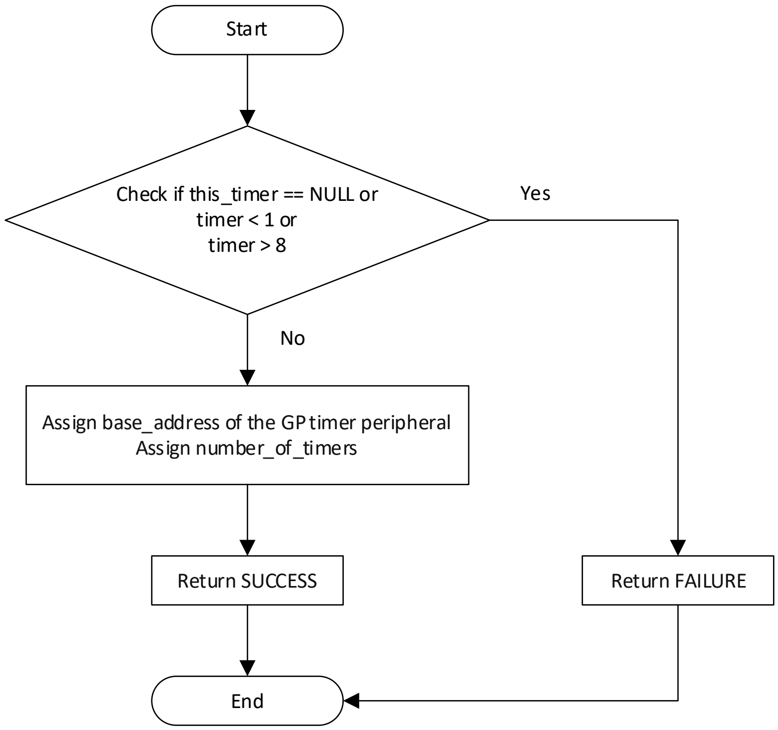

Figure 3.1. uint8_t gp_timer_init()

3.2. gp_timer_config()

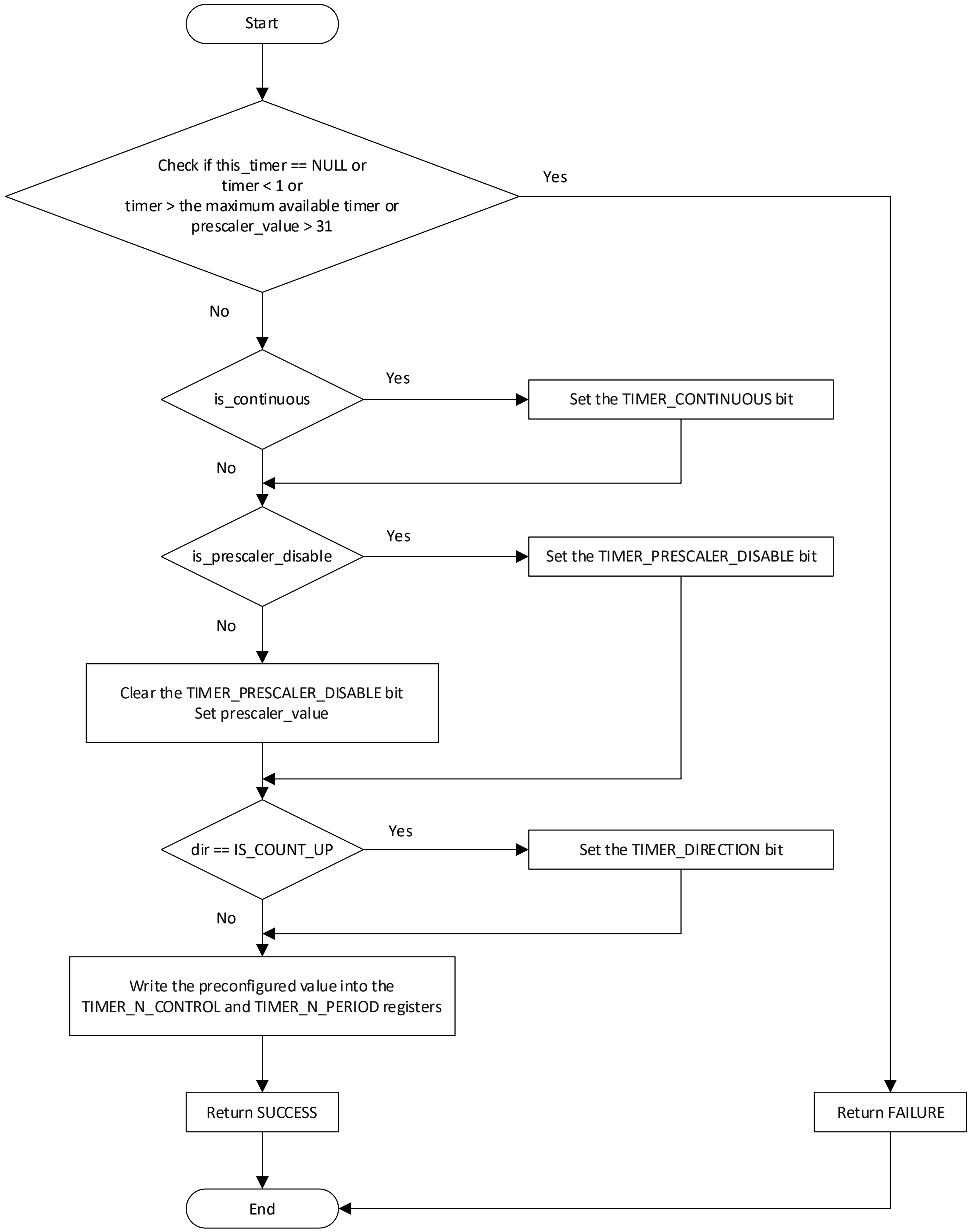

Figure 3.2. uint8_t gp_timer_config()

3.3. gp_timer_get_status()

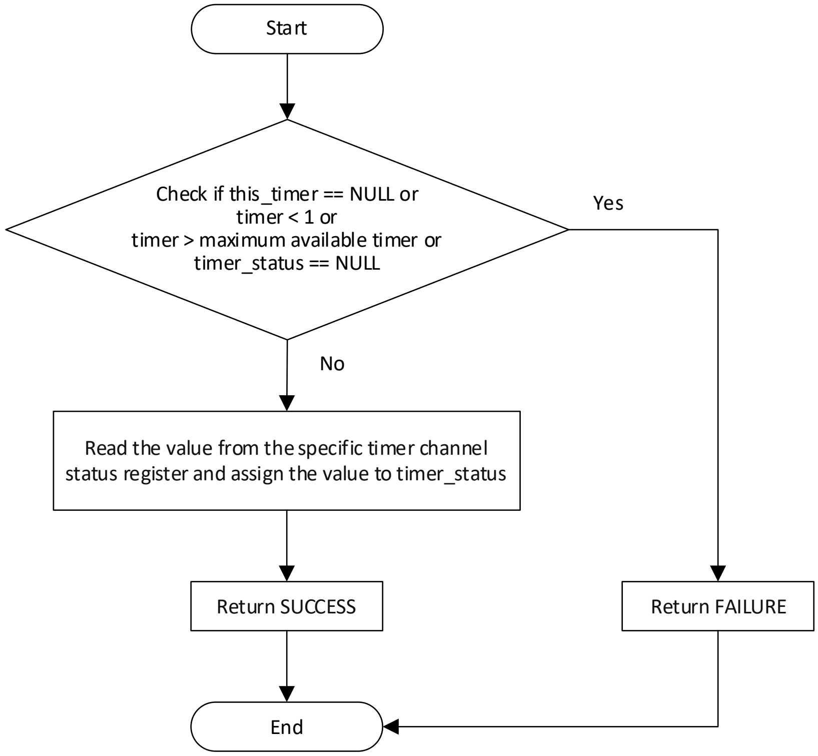

Figure 3.3. uint8_t gp_timer_get_status()

3.4. gp_timer_get_snapshot()

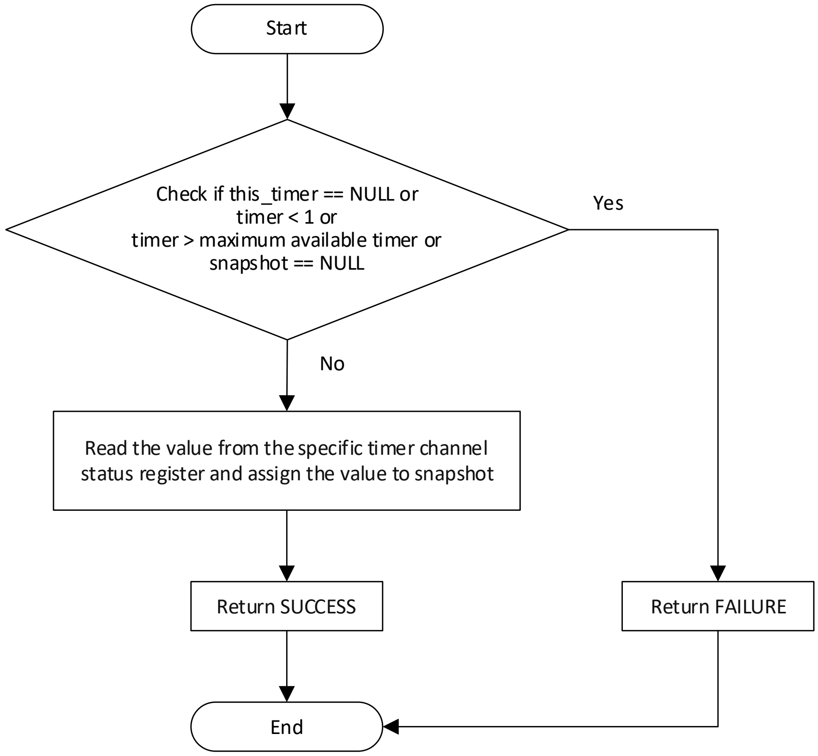

Figure 3.4. uint8_t gp_timer_get_snapshot()

3.5. gp_timer_enable_interrupt()

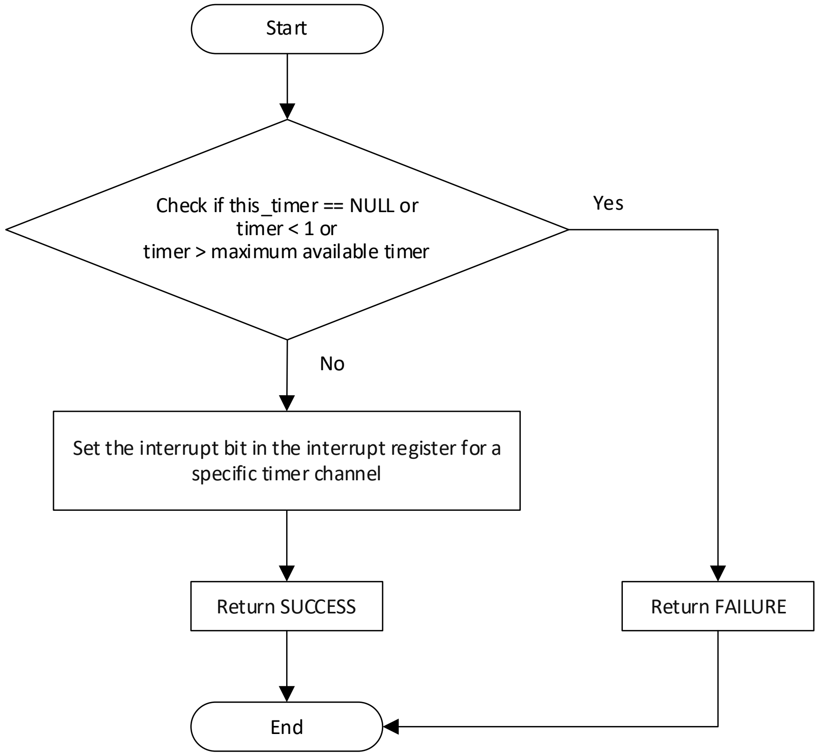

Figure 3.5. uint8_t gp_timer_enable_interrupt()

3.6. gp_timer_disable_interrupt()

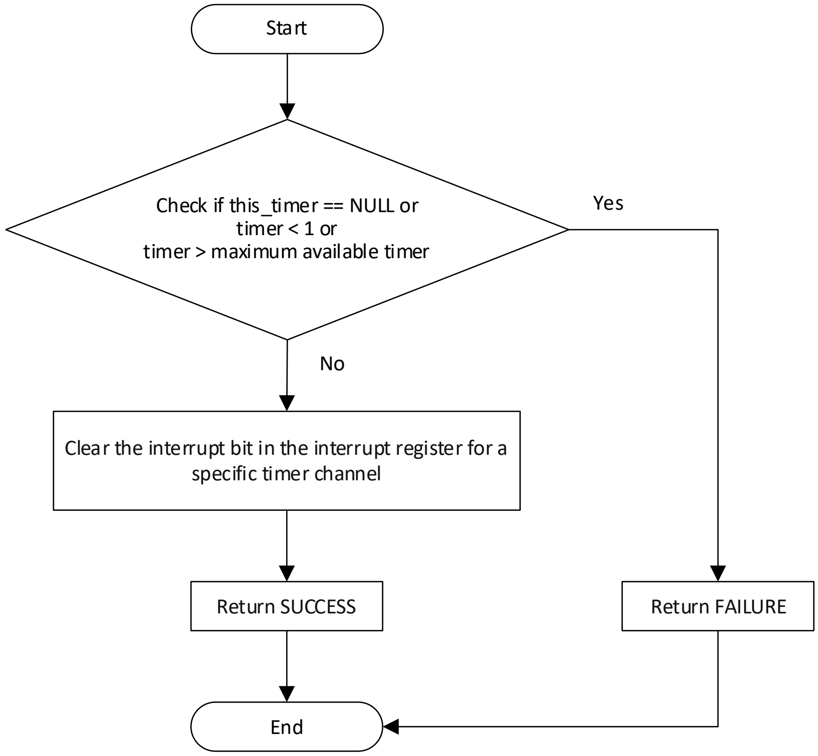

Figure 3.6. uint8_t gp_timer_disable_interrupt()

3.7. gp_timer_start()

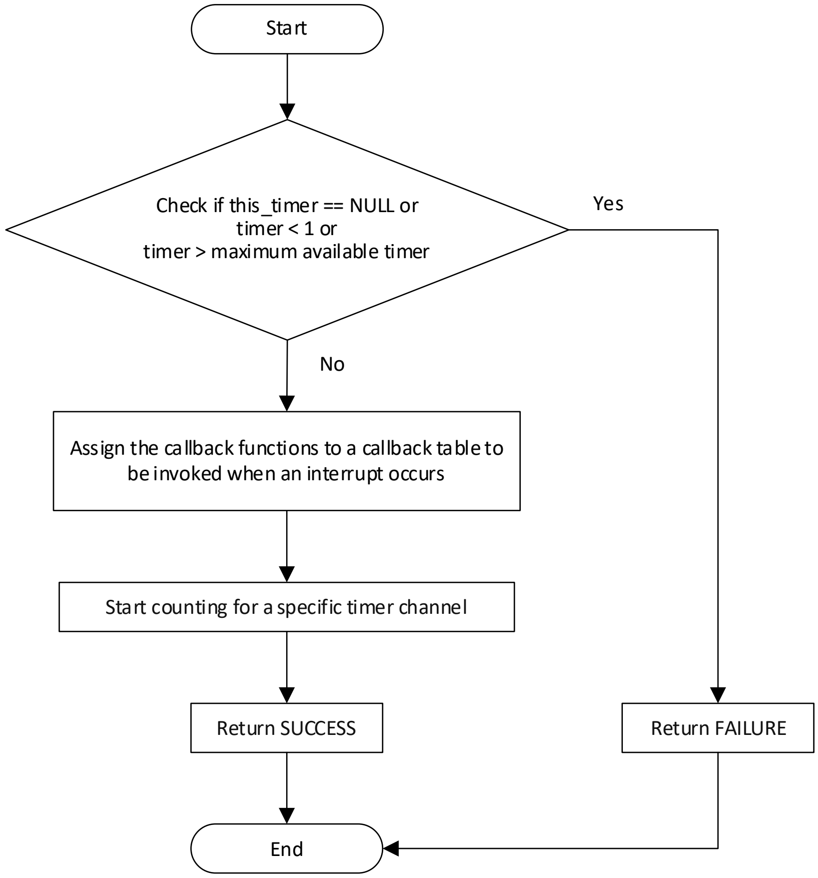

Figure 3.7. uint8_t gp_timer_start()

3.8. gp_timer_stop()

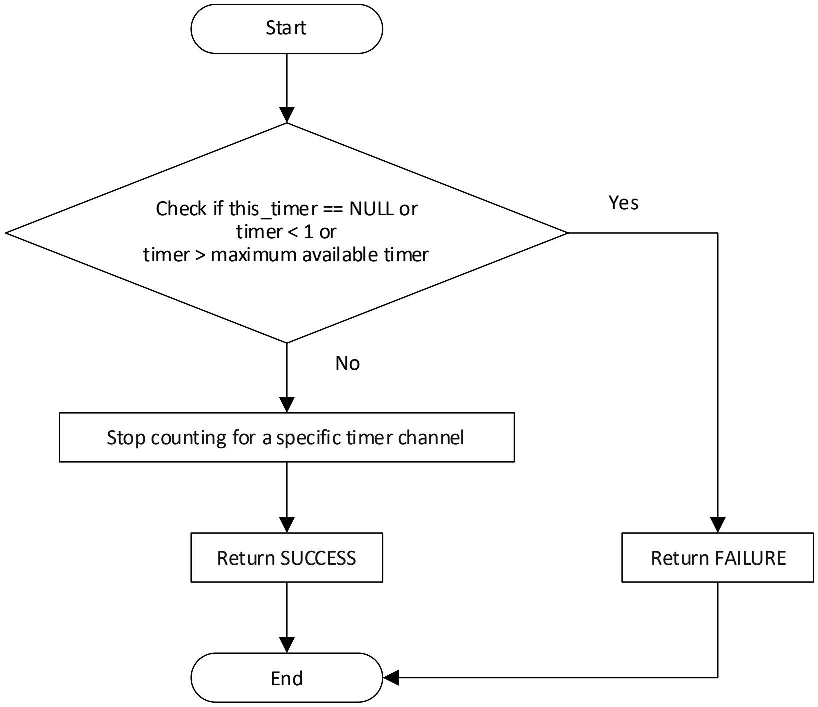

Figure 3.8. uint8_t gp_timer_stop()

3.9. gp_timer_isr()

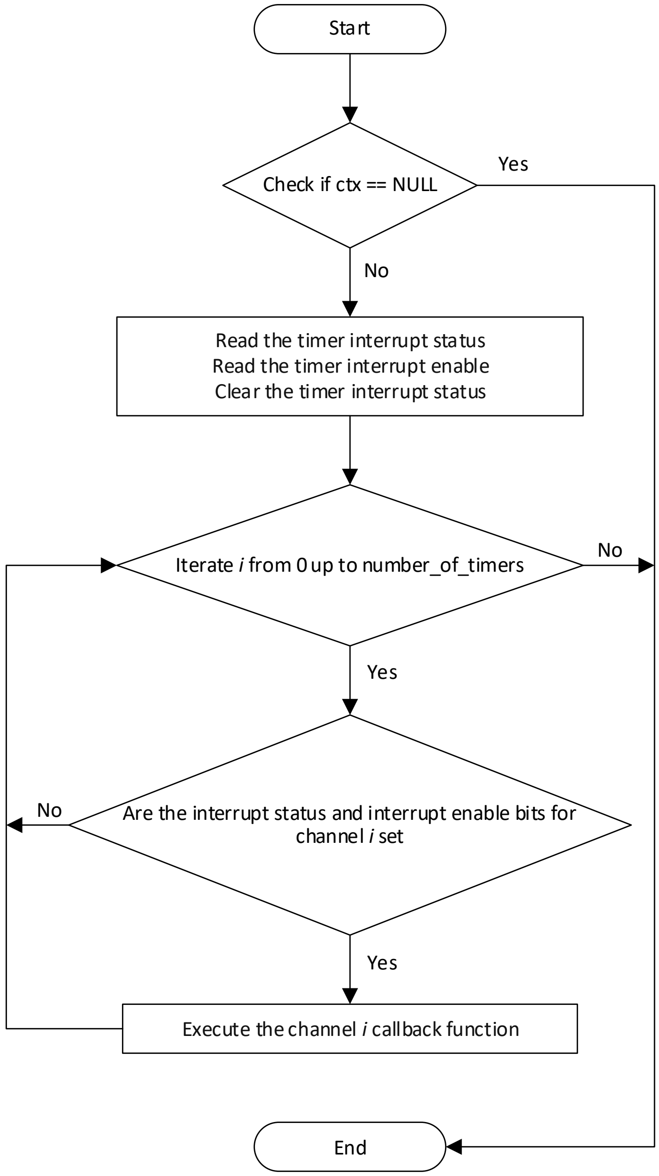

Figure 3.9. void gp_timer_isr()

4. API Data Structures

4.1. struct gp_timer_callback_entry

Table 4.1. gp_timer_callback_entry Parameters

Data Type	Struct Member	Description
void (*)(void *)	timer_callback	Callback function to call.
void *	context	User data passed to callback.

4.2. struct gp_timer_instance

Table 4.2. gp_timer_instance Parameters

Data Type	Struct Member	Description
uint32_t	base_addr	Base address of timer peripheral.
uint32_t	number_of_timers	Number of available timers.
struct gp_timer_callback_entry	callback_table[MAX_TIMERS]	Callback function entry. *MAX_TIMER* is set to 8.

5. API Macros

Table 5.1. API Macros Description

Macro	Description
#define GP_TIMER_DRV_VER	GP Timer driver version.
#define MAX_TIMERS	Maximum number of timers that can be configured in IP GUI.
#define TIMER_INT_STATUS	Timer interrupt status register offset.
#define TIMER_INT_ENABLE	Timer interrupt enable register offset.
#define TIMER_INT_SET	Timer interrupt set register offset.
#define GLB_CTRL	Global control register offset.
#define TIMER_N_STATUS	Status register for timer N.
#define TIMER_N_CONTROL	Control register for timer N.
#define TIMER_N_PERIOD	Period register for timer N.
#define TIMER_N_SNAPSHOT	Snapshot register for timer N.
#define TIMER_1_TIMEOUT_INTERRUPT	Timer 1 timeout interrupt bit.
#define TIMER_2_TIMEOUT_INTERRUPT	Timer 2 timeout interrupt bit.
#define TIMER_3_TIMEOUT_INTERRUPT	Timer 3 timeout interrupt bit.
#define TIMER_4_TIMEOUT_INTERRUPT	Timer 4 timeout interrupt bit.
#define TIMER_5_TIMEOUT_INTERRUPT	Timer 5 timeout interrupt bit.
#define TIMER_6_TIMEOUT_INTERRUPT	Timer 6 timeout interrupt bit.
#define TIMER_7_TIMEOUT_INTERRUPT	Timer 7 timeout interrupt bit.
#define TIMER_8_TIMEOUT_INTERRUPT	Timer 8 timeout interrupt bit.
#define TIMER_1_TIMEOUT_ENABLE	Timer 1 timeout enable bit.
#define TIMER_2_TIMEOUT_ENABLE	Timer 2 timeout enable bit.
#define TIMER_3_TIMEOUT_ENABLE	Timer 3 timeout enable bit.
#define TIMER_4_TIMEOUT_ENABLE	Timer 4 timeout enable bit.
#define TIMER_5_TIMEOUT_ENABLE	Timer 5 timeout enable bit.
#define TIMER_6_TIMEOUT_ENABLE	Timer 6 timeout enable bit.
#define TIMER_7_TIMEOUT_ENABLE	Timer 7 timeout enable bit.
#define TIMER_8_TIMEOUT_ENABLE	Timer 8 timeout enable bit.
#define TIMER_1_TIMEOUT_SET	Timer 1 timeout set bit.
#define TIMER_2_TIMEOUT_SET	Timer 2 timeout set bit.
#define TIMER_3_TIMEOUT_SET	Timer 3 timeout set bit.
#define TIMER_4_TIMEOUT_SET	Timer 4 timeout set bit.
#define TIMER_5_TIMEOUT_SET	Timer 5 timeout set bit.
#define TIMER_6_TIMEOUT_SET	Timer 6 timeout set bit.
#define TIMER_7_TIMEOUT_SET	Timer 7 timeout set bit.
#define TIMER_8_TIMEOUT_SET	Timer 8 timeout set bit.
#define TIMER_1_ENABLE	Timer 1 enable bit.
#define TIMER_2_ENABLE	Timer 2 enable bit.
#define TIMER_3_ENABLE	Timer 3 enable bit.
#define TIMER_4_ENABLE	Timer 4 enable bit.
#define TIMER_5_ENABLE	Timer 5 enable bit.
#define TIMER_6_ENABLE	Timer 6 enable bit.
#define TIMER_7_ENABLE	Timer 7 enable bit.
#define TIMER_8_ENABLE	Timer 8 enable bit.
#define TIMER_CONTINUOUS	Timer continuous bit.
#define TIMER_DIRECTION	Timer direction bit.
#define TIMER_START	Start timer bit.

Macro	Description
#define TIMER_STOP	Stop timer bit.
#define TIMER_PRESCALER_DISABLE	Timer prescaler disable bit.
#define TIMER_PRESCALER_WIDTH	Timer prescaler width.
#define TIMER_PRESCALER_POS	Timer prescaler position.
#define IS_COUNT_UP	Constant for counting up operation.
#define IS_COUNT_DOWN	Constant for counting down operation.
#define IS_PRESCALER_DISABLE	Constant for prescaler disable.
#define IS_PRESCALER_ENABLE	Constant for prescaler enable.
#define IS_CONTINUOUS	Constant for continuous mode.
#define IS_ONE_SHOT	Constant for one-shot mode.
#define SUCCESS	Success.
#define FAILURE	Failure.

References

- Timer/Counter IP User Guide (FPGA-IPUG-02139)
- Timer/Counter IP Release Notes (FPGA-RN-02022)
- Avant-E web page
- Avant-G web page
- Avant-X web page
- Certus-N2 web page
- Certus-NX web page
- CertusPro-NX web page
- CrossLink-NX web page
- MachXO4 web page
- MachXO5-NX web page
- Lattice Diamond Software web page
- Lattice Propel Design Environment web page
- Lattice Radiant Software web page
- Lattice Solutions IP Cores web page
- Lattice Insights web page for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.0, June 2026

Section	Change Summary
All	Initial release.