M-PESTI Initiator Driver API Reference

Technical Note

FPGA-TN-02413-1.2

December 2025

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. 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

Figures

Tables

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviations	Definition
APEN	Active Phase Enable
API	Application Programming Interface
DPEN	Discovery Phase Enable
FPGA	Field Programmable Gate Array
IP	Intellectual Property
SCLK	System Clock
SDRAM	Synchronous Dynamic Random Access Memory
SDK	Software Development Kit
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
CPU	Central Processing Unit
CMD	Command
GPIO	General Purpose I/O
GUI	Graphical User Interface
M-PESTI	Modular Peripheral Sideband Tunneling Interface
RAM	Random-Access Memory
OCP	Open Compute Project
SoC	System-on-a-Chip
SRAM	Static Random-Access Memory
UART	Universal Asynchronous Receiver Transmitter

1. Introduction

The Modular Peripheral Sideband Tunneling Interface (M-PESTI) Initiator IP core provides early peripheral presence detection and attribute collection before system boot up. This IP core supports bidirectional communication with initiator command and target response structure. It is designed to comply with the Modular-Peripheral Sideband Tunneling Interface (M-PESTI) Base Specification Version 1.0 Release Candidate 2 and Modular-Peripheral Sideband Tunneling Interface (M-PESTI) Base Specification Version 1.2 Release Candidate 2. This IP is OCP Ready™.

For further information about the IP core and its register descriptions, refer to the M-PESTI Initiator IP User Guide (FPGA-IPUG-02258).

Also, The M-PESTI Initiator Reference Design provides a solution template that uses the M-PESTI Initiator IP core, Initiator pattern generator block, and M-PESTI Target test component. The Reference Design is compliant with the M-PESTI Base Specification (part of the DC-MHS version 1.0 specification). Refer to the M-PESTI Initiator Reference Design (FPGA-RD-02312).

1.1. Purpose

M-PESTI and its SDK are a set of application programming interfaces (APIs) that provide access to specific Lattice hardware and software capabilities. This document serves as a reference guide for developers by providing details of the C language driver APIs and function call flows.

1.2. Audience

This document is intended for embedded system designers and software developers using Lattice MachXO5™-NX, MachXO3™, and MachXO3D™ devices. This technical guide assumes readers have expertise in embedded systems and FPGA technologies.

1.3. Driver and IP Compatibility

Table 1.1. Driver and IP Version

Driver version	IP version
25.02.00	2.0.1

Note: Refer to the M-PESTI IP Release Note (FPGA-RN-02036) for more information on the driver and IP versions.

Table 1.2. Quick Facts of Driver Tested Environment

Hardware Device	Tool Version
MachXO5™-NX	Lattice Propel™ Builder 2025.1.1
	Lattice Propel SDK 2025.1.1
	Lattice Radiant™ Software 2025.1.1
	Radiant Programmer 2025.1.1

2. API Description

2.1. mpesti_initiator_init()

This API takes base address of M-PESTI IP and initializes the M-PESTI Initiator IP.

```
uint32_t mpesti_initiator_init(mpesti_initiator_handle_t *handle, uint32_t base_addr);
```

Table 2.1. mpesti_initiator_init()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	base_addr	Base address of the M-PESTI IP.	

2.2. mpesti_initiator_reset()

This API performs a comprehensive reset of the M-PESTI initiator IP. Refer to the M-PESTI Initiator IP User Guide (FPGA-IPUG-02258) for register description or user commands.

```
uint32_t mpesti_initiator_reset(mpesti_initiator_handle_t *handle);
```

Table 2.2. mpesti_initiator_reset()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success

2.3. mpesti_initiator_init_cpu_timer()

This API initializes the CPU timer to enable the timeout functionality. The timer is part of the CPU and is not included in the M-PESTI IP. Use this API only when the LATTICE RISC-V core is present in the SoC design and the CPU timer is not allocated for any other purpose.

To enable timeout feature, add *_TIMEOUT_ENABLE_* under Project > Properties > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C Compiler > Preprocessor > Defined symbols (-D).

```
uint32_t mpesti_initiator_init_cpu_timer(mpesti_initiator_handle_t *handle, uint32_t
cpu_timer_base_addr, uint64_t system_frequency_mhz);
```

Table 2.3. mpesti_initator_init_cpu_timer()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	cpu_timer_base_addr	Base address for CPU timer.	
In	system_frequency_mhz	Frequency which is fed to CPU timer in MHz.	

2.4. mpesti_initiator_get_cpu_timer_value()

This API gets the current CPU timer value. The timer is part of the CPU and is not included in the M-PESTI IP. Use this API only when the LATTICE RISC-V core is present in the SoC design and the CPU timer is not allocated for any other purpose.

To enable timeout feature, add *_TIMEOUT_ENABLE_* under Project > Properties > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C Compiler > Preprocessor > Defined symbols (-D).

```
uint32_t mpesti_initiator_get_cpu_timer_value(mpesti_initiator_handle_t *handle,
uint64_t *timer_value);
```

Table 2.4. mpesti_initiator_get_cpu_timer_value()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	timer_value	Pointer to store the 64-bits timer value.	

2.5. mpesti_initiator_clear_user_cmd_status()

This API clears the user command status register.

```
uint32_t mpesti_initiator_clear_user_cmd_status(mpesti_initiator_handle_t *handle);
```

Table 2.5. mpesti_initiator_clear_user_cmd_status()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success

2.6. mpesti_initiator_get_user_cmd_status()

This API retrieves the value of the user command status register.

```
uint32_t mpesti_initiator_get_user_cmd_status(mpesti_initiator_handle_t *handle,
uint32_t *user_cmd_status);
```

Table 2.6. mpesti_initiator_get_user_cmd_status()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	user_cmd_status	Pointer to store the user command status.	

2.7. mpesti_initiator_poll_user_cmd_status()

This API polls the user command status until matching bits are set or timeout. The timer used is part of the CPU and is not included in the M-PESTI IP. Use this API only when the LATTICE RISC-V core is present in the SoC design and the CPU timer is not allocated for any other purpose.

To enable the timeout feature, add *_TIMEOUT_ENABLE_* under Project > Properties > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C Compiler > Preprocessor > Defined symbols (-D).

```
uint32_t mpesti_initiator_poll_user_cmd_status(mpesti_initiator_handle_t *handle,
uint32_t poll_mask, uint32_t *user_cmd_status);
```

Table 2.7. mpesti_initiator_poll_user_cmd_status()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	poll_mask	Bitmask of status bits to poll for. Function returns when ALL bits in poll_mask are set.	
Out	user_cmd_status	Pointer to store the user command status.	

2.8. mpesti_initiator_set_user_wr_data()

This API sets the data to be transmitted in the user_wr_data register.

```
uint32_t mpesti_initiator_set_user_wr_data(mpesti_initiator_handle_t *handle, uint32_t
wr_data);
```

Table 2.8. mpesti_initiator_set_user_wr_data()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	wr_data	8-bit data value to be written to user_wr_data register.	

2.9. mpesti_initiator_get_user_rd_data()

This API reads the value of the user_rd_data register.

```
uint32_t mpesti_initiator_get_user_rd_data(mpesti_initiator_handle_t *handle, uint32_t
*rd_data);
```

Table 2.9. mpesti_initiator_get_user_rd_data()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	rd_data	Pointer to store the value read from the user_rd_data register.	

2.10. mpesti_initiator_set_user_cmd()

This API provides an interface for configuring the user_command_register.

```
uint32_t mpesti_initiator_set_user_cmd(mpesti_initiator_handle_t *handle, uint32_t
is_send, uint32_t is_broadcast, uint32_t is_expect_abort, unint32_t read_byte_length,
uint32_t target_select);
```

Table 2.10. mpesti_initiator_set_user_cmd()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	is_send	Set 1 to send data, Set 0 to prepare for receiving.	
In	is_broadcast	Set 1 for broadcast, Set 0 for unicast.	
In	is_expect_abort	Set 1 to expect data abort response (for broadcast) or expect data read (for unicast), Set 0 otherwise.	
In	read_byte_length	Number of expected bytes to read when is_send is 0 or is_expect_abort is 1 for unicast.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.11. mpesti_initiator_send_broadcast_command()

This API sends a broadcast command through the MPESTI initiator. Only one data is sent at a time.

```
uint32_t mpesti_initiator_send_broadcast_command(mpesti_initiator_handle_t *handle,
uint32_t command, uint32_t is_abort);
```

Table 2.11. mpesti_initiator_send_broadcast_command()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	command	The broadcast command to be sent to all devices	
In	is_abort	Set 1 to abort existing transmission, Set 0 to wait for current transmission to finish.	

2.12. mpesti_initiator_send_target_command()

This API sends a command to a specific target.

```
uint32_t mpesti_initiator_send_target_command(mpesti_initiator_handle_t *handle,
uint32_t command, uint32_t expect_resp, uint32_t read_byte_length, uint32_t
target_select);
```

Table 2.12. mpesti_initiator_send_target_command()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	command	Command to be sent to the target.	
In	expect_resp	Set 1 if response is expected. Set 0 otherwise.	
In	read_byte_length	Number of expected bytes to read when expect_resp is 1.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.13. mpesti_initiator_send_target_commands()

This API sends multiple commands to a specific target sequentially.

```
uint32_t mpesti_initiator_send_target_commands(mpesti_initiator_handle_t *handle,
uint32_t *commands, uint32_t size, uint32_t expect_resp, uint32_t read_byte_length,
uint32_t target_select);
```

Table 2.13. mpesti_initiator_send_target_commands()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	commands	Pointer to array of commands to be sent to the target.	
In	size	Number of commands in the commands array to be sent.	
In	expect_resp	Set 1 if response is expected. Set 0 otherwise.	
In	read_byte_length	Number of expected bytes to read when expect_resp is 1.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.14. mpesti_initiator_receive_target_commands()

This API receives command data from a specific target.

```
uint32_t mpesti_initiator_receive_target_commands(mpesti_initiator_handle_t *handle,
uint32_t expected_size, uint32_t *cmd_out, uint32_t *actual_size, uint32_t
target_select);
```

Table 2.14. mpesti_initiator_receive_target_commands()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	expected_size	Expected number of bytes to receive.	
Out	cmd_out	Pointer to buffer to store received command data.	
Out	actual_size	Pointer to store the actual number of bytes received.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.15. mpesti_initiator_set_vwire_out()

This API writes virtual wire output data for a specific target. The data is transferred during the Virtual Wire Exchange in the active phase. The output size for the virtual wire is configured through the IP GUI.

```
uint32_t mpesti_initiator_set_vwire_out(mpesti_initiator_handle_t *handle, uint8_t
*vwire_out, uint32_t size, uint32_t target_select);
```

Table 2.15. mpesti_initiator_set_vwire_out()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	vwire_out	Pointer to buffer containing virtual wire output data.	
In	size	Size of the virtual wire data in bytes	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.16. mpesti_initiator_get_vwire_in()

This API reads virtual wire input data from the specified target connected to the MPESTI initiator interface. The data is received during the Virtual Wire Exchange in the active phase. The input size for the virtual wire is configured through the IP GUI.

```
uint32_t mpesti_initiator_get_vwire_in(mpesti_initiator_handle_t *handle, uint8_t
*vwire_in, uint32_t size, uint32_t target_select);
```

Table 2.16. mpesti_initiator_get_vwire_in()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	vwire_in	Pointer to buffer containing virtual wire input data.	
In	size	Number of virtual wire entries to read.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.17. mpesti_initiator_manual_virtual_wire_exchange()

This API performs a manual virtual wire exchange with a specific target.

```
uint32_t mpesti_initiator_manual_virtual_wire_exchange(mpesti_initiator_handle_t
*handle, uint32_t *data_out, uint32_t data_out_size, uint32_t *data_in, uint32_t
data_in_size, uint32_t target_select);
```

Table 2.17. mpesti_initiator_manual_virtual_wire_exchange()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	data_out	Pointer to buffer containing data to send.	
In	data_out_size	Size of the output data buffer.	
Out	data_in	Pointer to buffer to store received data.	
In	data_in_size	Expected size of the input data.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.18. mpesti_initiator_configure_discovery_phase()

This API configures the discovery phase enable bit (DISCOVERY_PHASE_ENABLE) for a specific target. When enabled, the discovery phase allows the initiator to identify the connected device type and determine device features. The discovery phase must occur before the active phase and is essential for the initiator to learn about connected peripherals.

```
uint32_t mpesti_initiator_configure_discovery_phase(mpesti_initiator_handle_t *handle,
uint32_t is_enable, uint32_t target_select);
```

Table 2.18. mpesti_initiator_configure_discovery_phase()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	is_enable	Set 1 to enable, Set 0 to disable.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.19. mpesti_initiator_configure_active_phase()

This API sets the APEN (Active Phase Enable) bit of specific target, controlling whether the initiator enters the virtual wire exchange phase. When enabled (1), the initiator enters the active phase if the discovery payload has been successfully received (DSTAT[1:0] = 10b). The active phase enables bidirectional virtual wire data exchange between the initiator and target using the Virtual Wire Exchange (VWE), allowing device-specific virtual wire communication after successful discovery.

```
uint32_t mpesti_initiator_configure_active_phase(mpesti_initiator_handle_t *handle,
uint32_t is_enable, uint32_t target_select);
```

Table 2.19. mpesti_initiator_configure_active_phase()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	is_enable	Set 1 to enable, Set 0 to disable.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.20. mpesti_initiator_get_target_phase_status()

This API retrieves the phase status of a specific target. Value of 1 indicates that the target is in the active phase, while value of 0 signifies the discovery phase.

```
uint32_t mpesti_initiator_get_target_phase_status(mpesti_initiator_handle_t *handle,
uint32_t *is_active_phase, uint32_t target_select);
```

Table 2.20. mpesti_initiator_get_target_phase_status()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	is_active_phase	Pointer to store the active phase status.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.21. mpesti_initiator_get_target_discovery_status()

This API reads the discovery status of a specific target.

```
uint32_t mpesti_initiator_get_target_discovery_status(mpesti_initiator_handle_t *handle,
uint32_t *target_status, uint32_t target_select);
```

Table 2.21. mpesti_initiator_get_target_discovery_status()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	target_status	Pointer to store the target status. 0b00: Absent 0b01: Presence 0b10: M-PESTI is present with good payload 0b11: M-PESTI is present without successful payload	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.22. mpesti_initiator_get_target_error_status_control()

This API retrieves the target error status control register value for a specified target.

```
uint32_t mpesti_initiator_get_target_error_status_control(mpesti_initiator_handle_t
*handle, mpesti_target_error_status_control_u *mpesti_target_error_status_control_u,
uint32_t target_select);
```

Table 2.22. mpesti_initiator_get_target_error_status_control()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	mpesti_target_error_status_control_u	Pointer to union structure to store the error status control value.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.23. mpesti_initiator_set_target_error_status_control()

This API sets the target error status control register for a specified target.

```
uint32_t mpesti_initiator_set_target_error_status_control(mpesti_initiator_handle_t
*handle, mpesti_target_error_status_control_u mpesti_target_error_status_control_u,
uint32_t target_select);
```

Table 2.23. mpesti_initiator_set_target_error_status_control()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	mpesti_target_error_status_control_u	Union structure containing the error status control value to write.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.24. mpesti_initiator_get_target_payload()

This API reads the target payload for a specific target.

```
uint32_t mpesti_initiator_get_target_payload(mpesti_initiator_handle_t *handle, uint32_t *payload_data, uint32_t size, uint32_t target_select);
```

Table 2.24. mpesti_initiator_get_target_payload()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	payload_data	Pointer to store the actual payload data read.	
In	size	Size of the maximum static discovery payload size which is configured in Propel Builder UI.	
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.25. mpesti_initiator_clear_target_error_status()

This API clears all error status bits for a specific target.

```
uint32_t mpesti_initiator_clear_target_error_status(mpesti_initiator_handle_t *handle, uint32_t target_select);
```

Table 2.25. mpesti_initiator_clear_target_error_status()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	target_select	Selects the Target number. Maximum allowable number is set in the IP GUI.	

2.26. mpesti_initiator_configure_interrupt_enable()

This API disables or enables the interrupt for M-PESTI Initiator.

```
uint32_t mpesti_initiator_configure_interrupt_enable(mpesti_initiator_handle_t *handle, uint32_t is_enable, uint32_t interrupt_enable);
```

Table 2.26. mpesti_initiator_configure_interrupt_enable()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	is_enable	Flag to enable (1) or disable (0) the interrupt.	
In	interrupt_enable	Bitmask for the specific interrupts to enable or disable.	

2.27. mpesti_initiator_set_interrupt_set()

This API set interrupt set register which triggers interrupt for specific interrupt.

```
uint32_t mpesti_initiator_set_interrupt_set(mpesti_initiator_handle_t *handle, uint32_t int_set);
```

Table 2.27. mpesti_initiator_set_interrupt_set()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
In	int_set	Input data that sets the interrupt bit fields of the Interrupt Set register.	

2.28. mpesti_initiator_get_interrupt_status()

This API retrieves the current status of the interrupt status register.

```
uint32_t mpesti_initiator_get_interrupt_status(mpesti_initiator_handle_t *handle,
uint32_t *interrupt_status);
```

Table 2.28. mpesti_initiator_get_interrupt_status()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	interrupt_status	Pointer to store the retrieved interrupt status.	

2.29. mpesti_initiator_get_ip_info()

This API retrieves the IP information from the M-PESTI initiator.

```
uint32_t mpesti_initiator_get_ip_info(mpesti_initiator_handle_t *handle, uint32_t
*ip_block_version, uint32_t *ip_block_id);
```

Table 2.29. mpesti_initiator_get_ip_info()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	ip_block_version	Pointer to store the extracted IP block version.	
Out	ip_block_id	Pointer to store the extracted IP block ID.	

2.30. mpesti_initiator_get_max_supported_targets()

This API retrieves the maximum number of supported targets from the M-PESTI initiator.

```
uint32_t mpesti_initiator_get_max_supported_targets(mpesti_initiator_handle_t *handle,
uint32_t *max_supported_targets);
```

Table 2.30. mpesti_initiator_get_max_supported_targets()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	max_supported_targets	Pointer to store the maximum number of supported targets.	

2.31. mpesti_initiator_get_max_supported_payload_size()

This API get the maximum supported payload size for the MPESTI initiator.

```
uint32_t mpesti_initiator_get_max_supported_payload_size(mpesti_initiator_handle_t
*handle, uint32_t *max_supported_payload_size);
```

Table 2.31. mpesti_initiator_get_max_supported_payload_size()

In/Out	Parameter	Description	Returns
In	handle	Handle of the M-PESTI Initiator instance.	0: failure 1: success
Out	max_supported_payload_size	Pointer to store the maximum supported payload size.	

3. Function Call Flow Diagrams

3.1. mpesti_initiator_init()

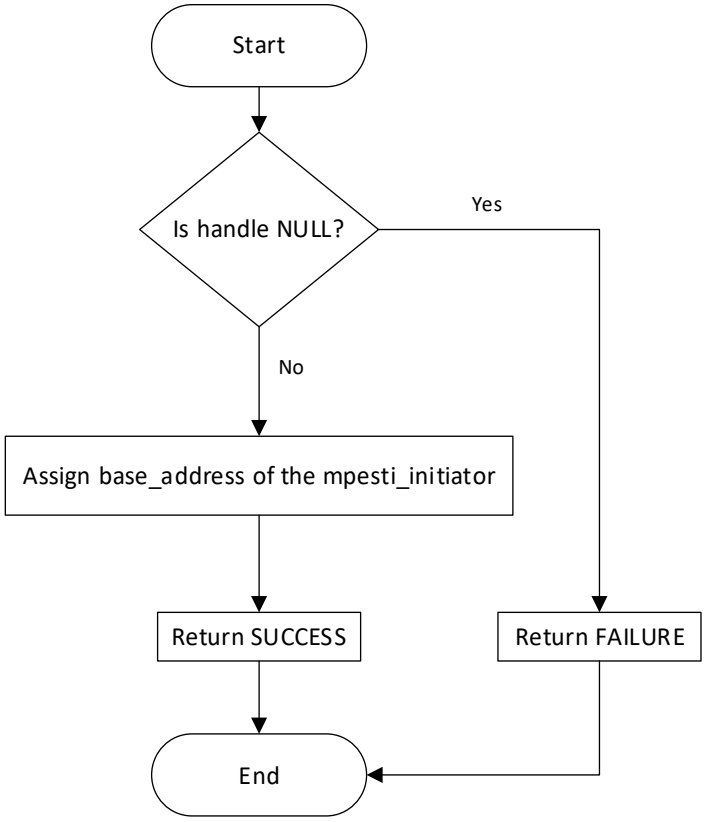

Figure 3.1. mpesti_initiator_init()

3.2. mpesti_initiator_reset()

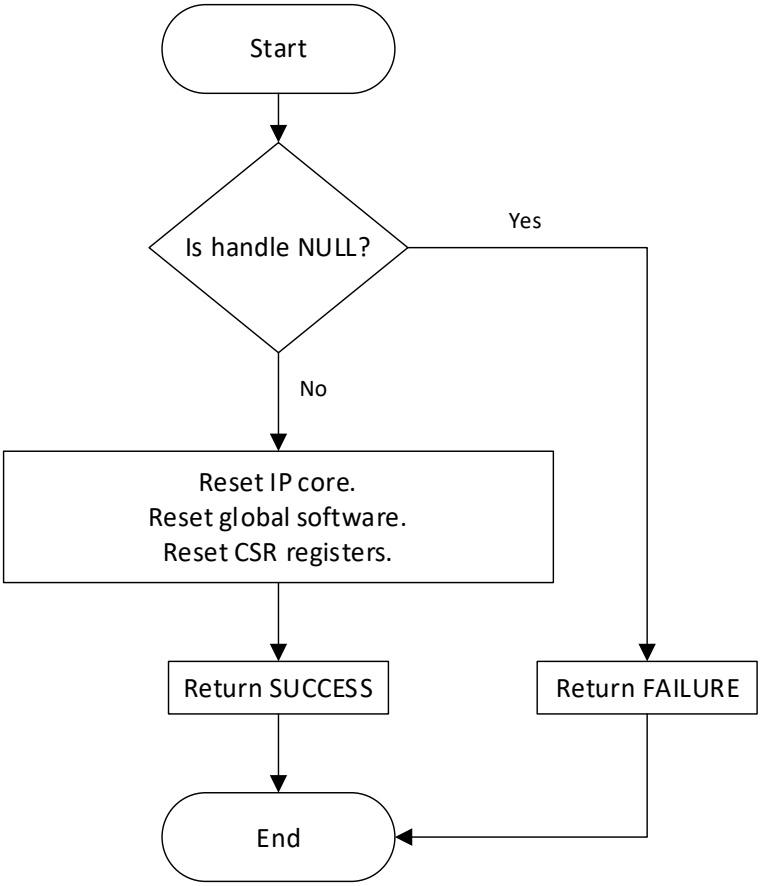

Figure 3.2. mpesti_initiator_reset()

3.3. mpesti_initiator_init_cpu_timer()

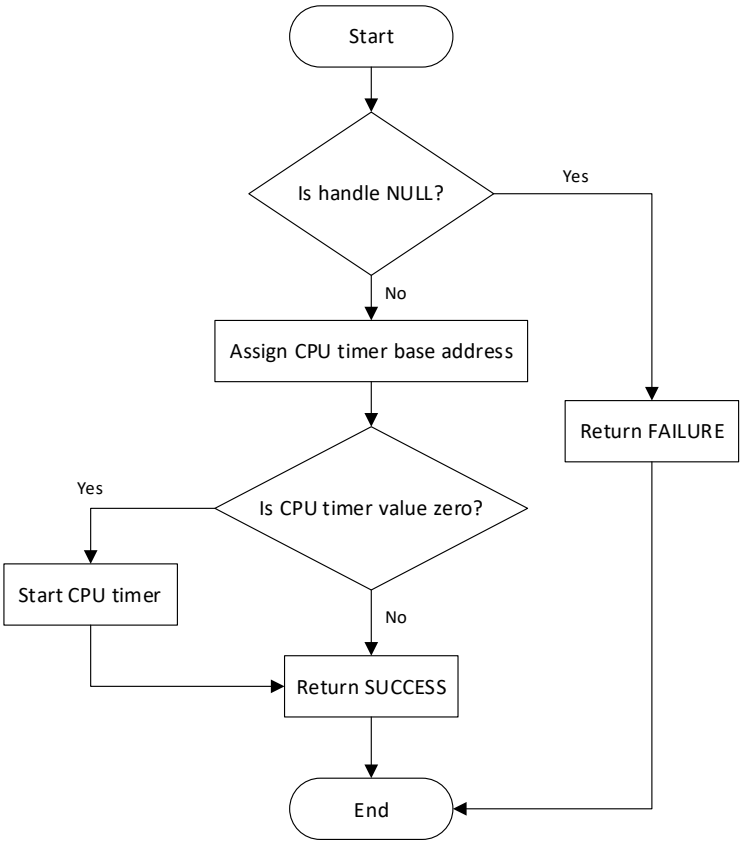

Figure 3.3. mpesti_initiator_init_cpu_timer()

3.4. mpesti_initiator_get_cpu_time_value()

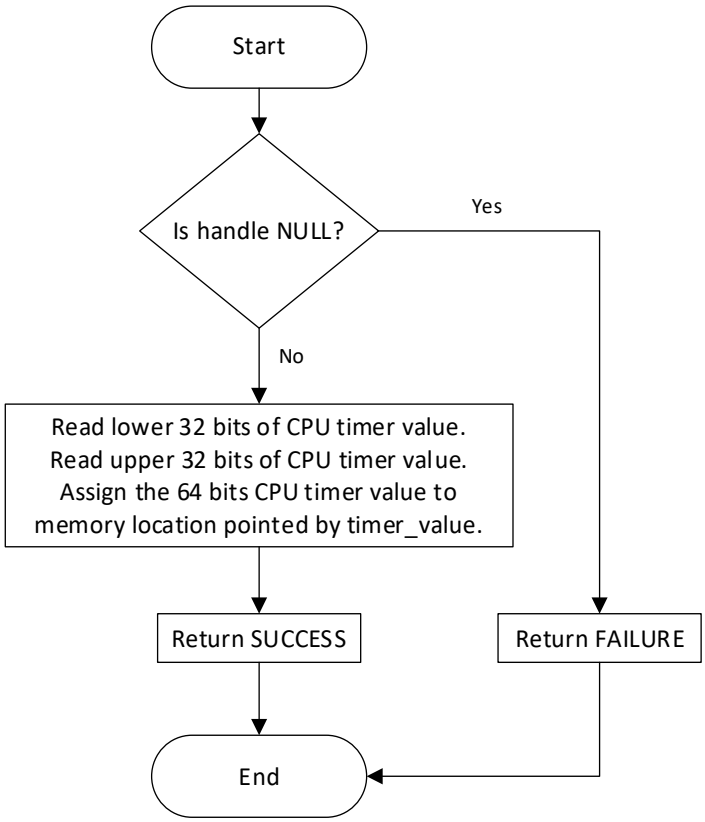

Figure 3.4. mpesti_initiator_get_cpu_time_value()

3.5. mpesti_initiator_clear_user_cmd_status()

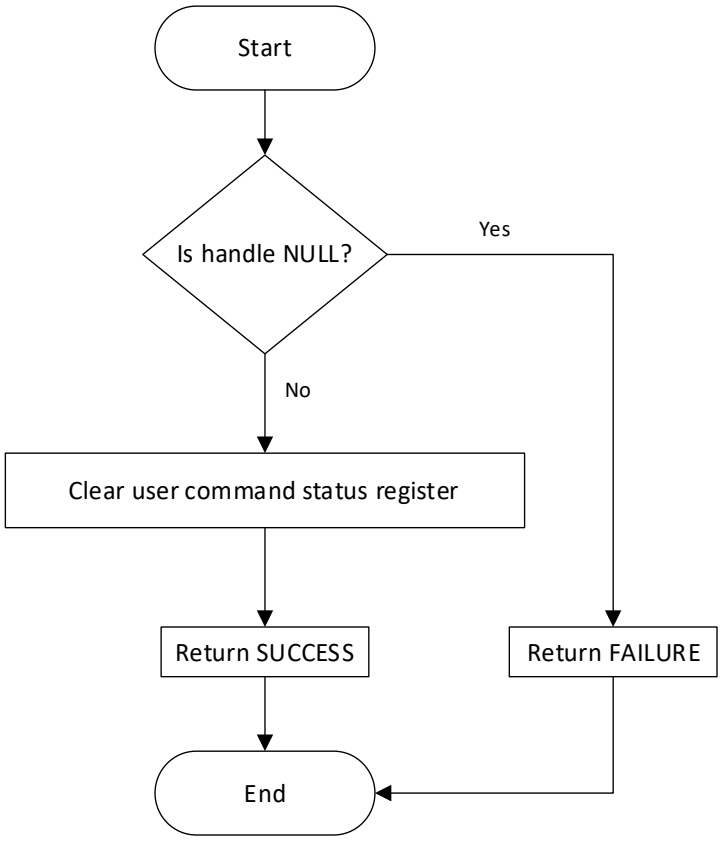

Figure 3.5. mpesti_initiator_clear_user_cmd_status()

3.6. mpesti_initiator_get_user_cmd_status()

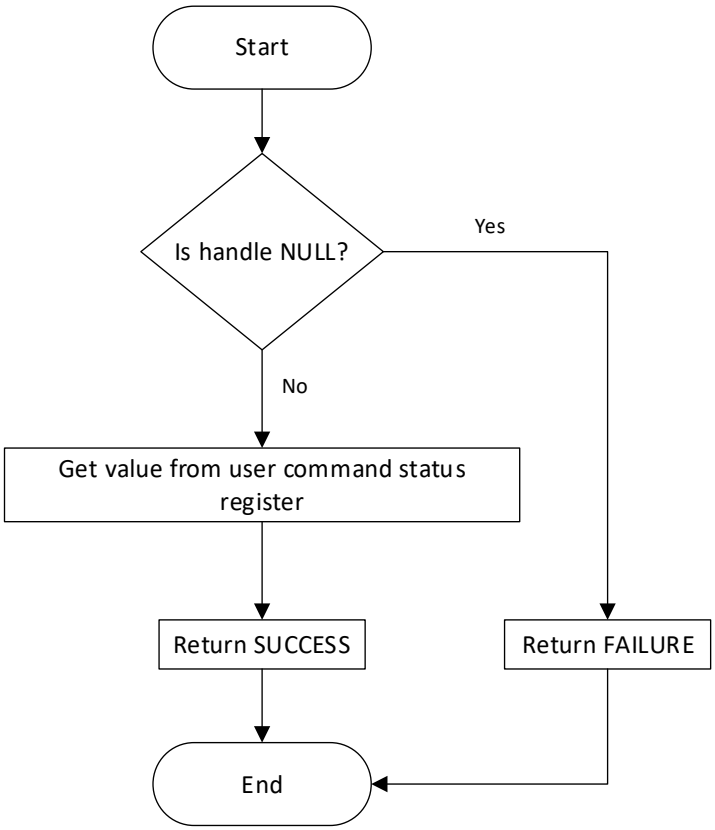

Figure 3.6. mpesti_initiator_get_user_cmd_status()

3.7. mpesti_initiator_poll_user_cmd_status()

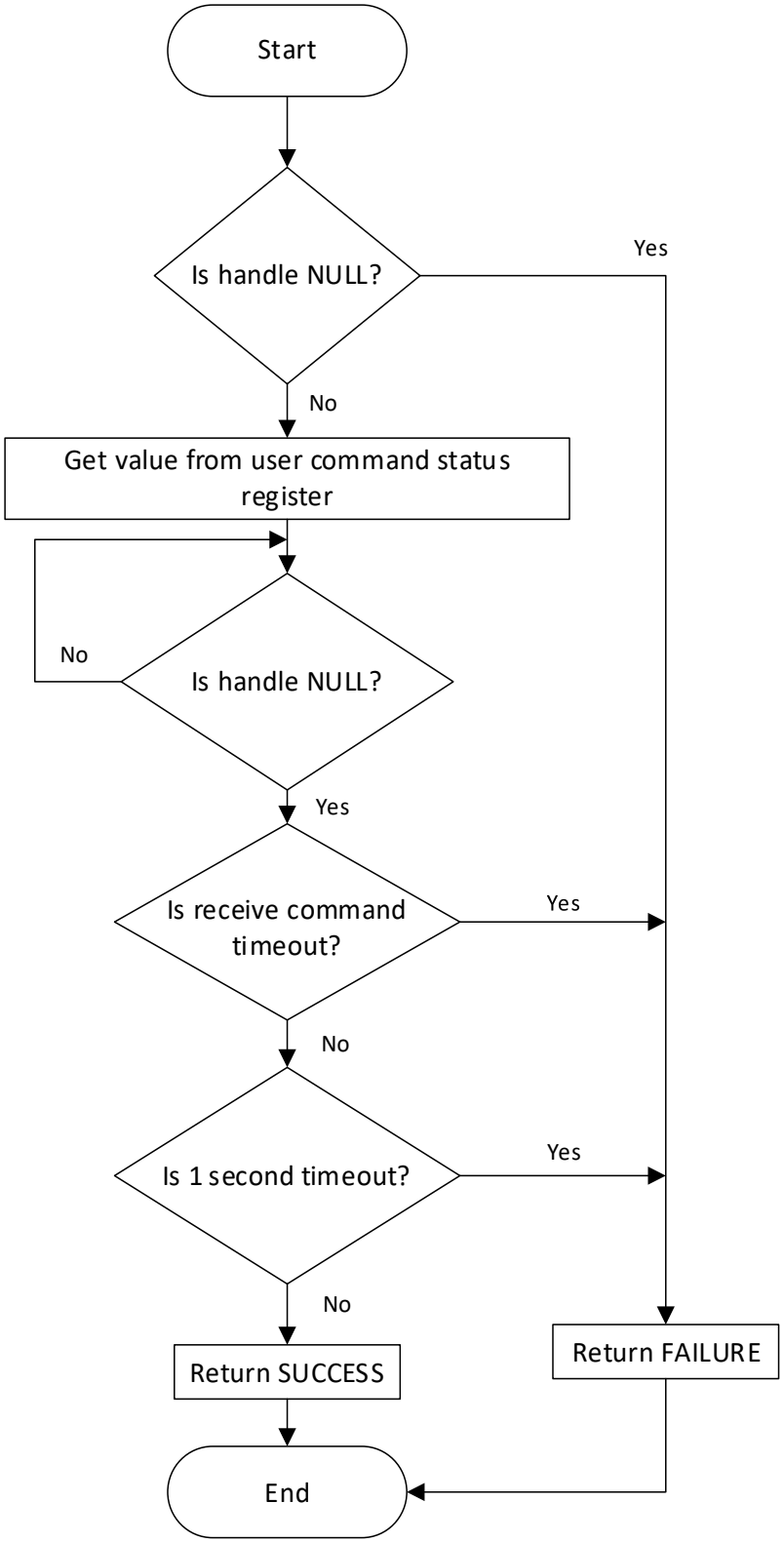

Figure 3.7. mpesti_initiator_poll_user_cmd_status()

3.8. mpesti_initiator_set_user_wr_data()

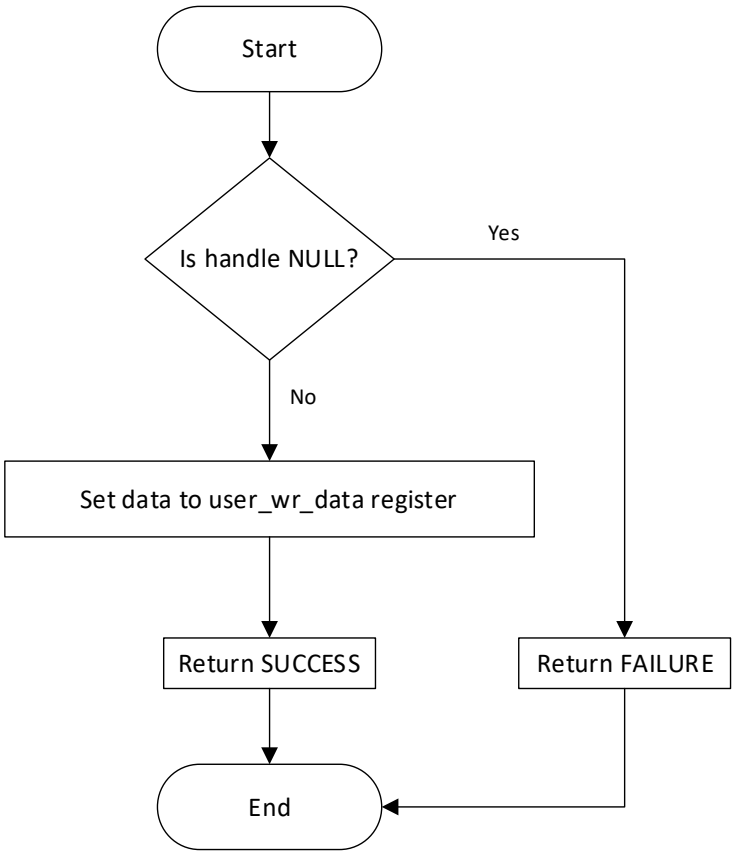

Figure 3.8. mpesti_initiator_set_user_wr_data()

3.9. mpesti_initiator_get_user_rd_data()

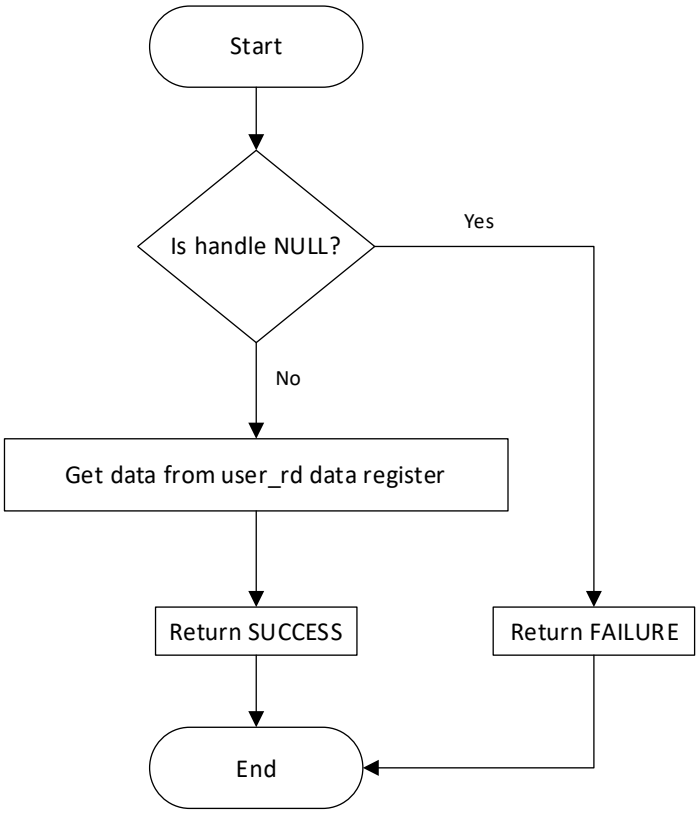

Figure 3.9. mpesti_initiator_get_user_rd_data()

3.10. mpesti_initiator_set_user_cmd()

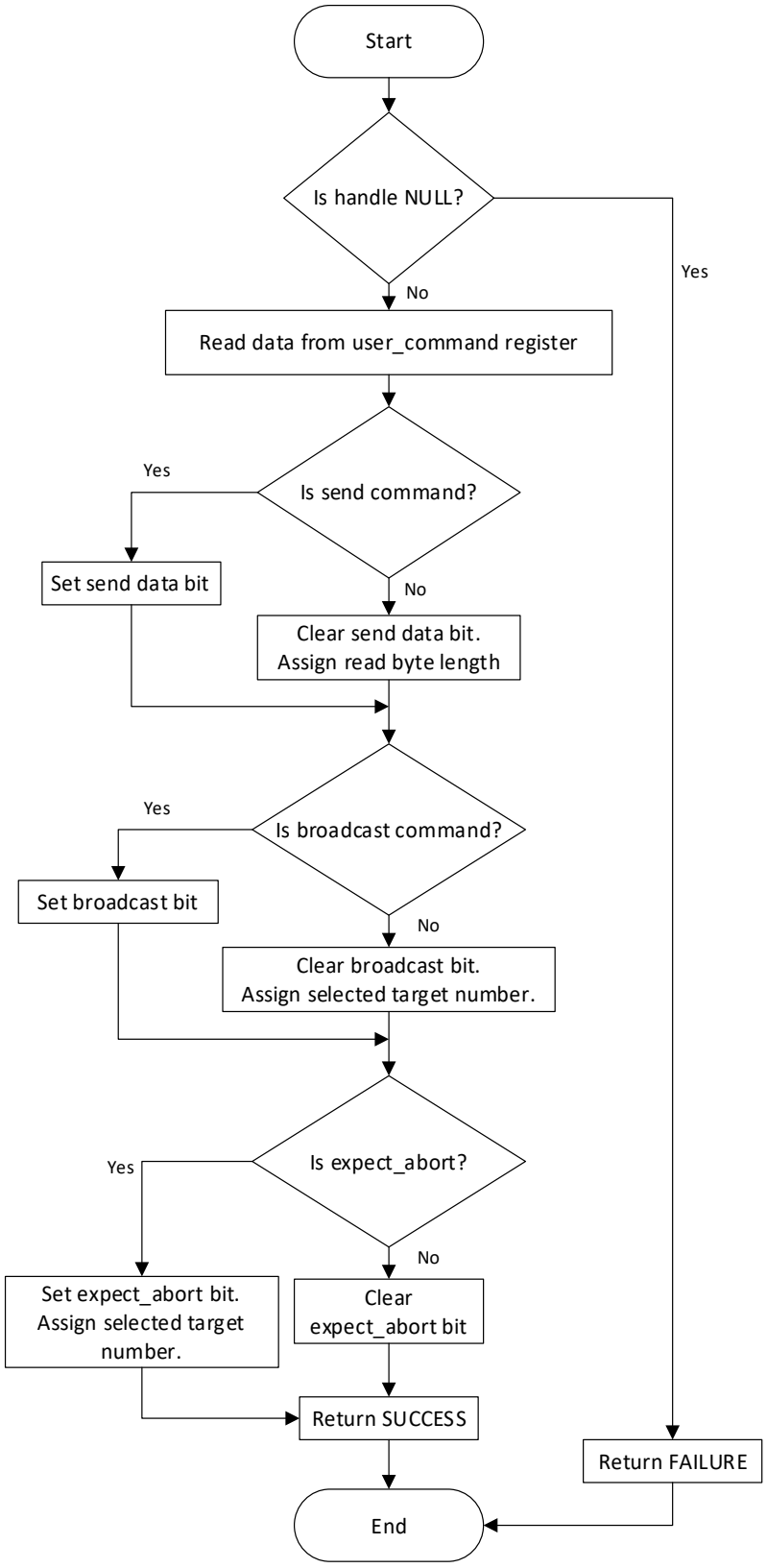

Figure 3.10. mpesti_initiator_set_user_cmd()

3.11. mpesti_initiator_send_broadcast_command()

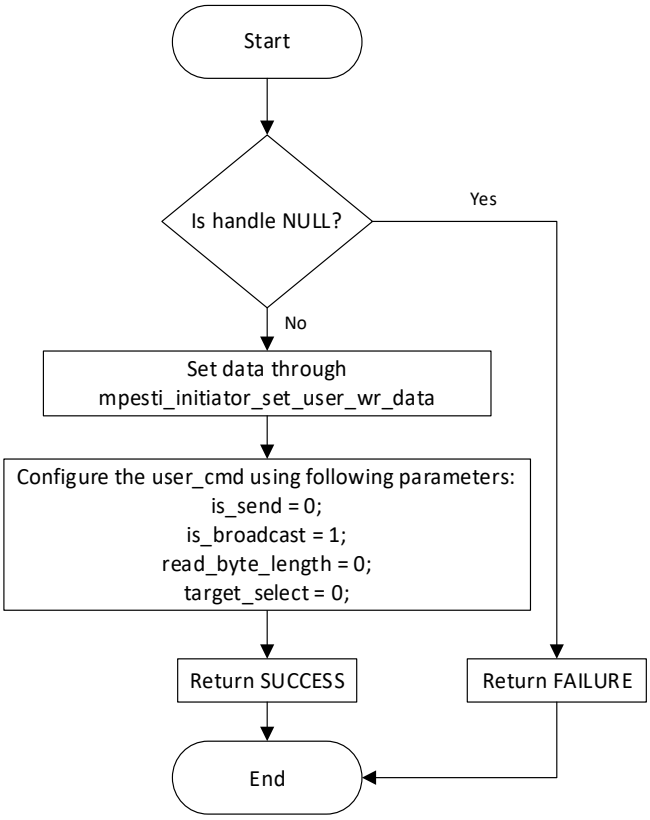

Figure 3.11. mpesti_initiator_send_broadcast_command()

3.12. mpesti_initiator_send_target_command()

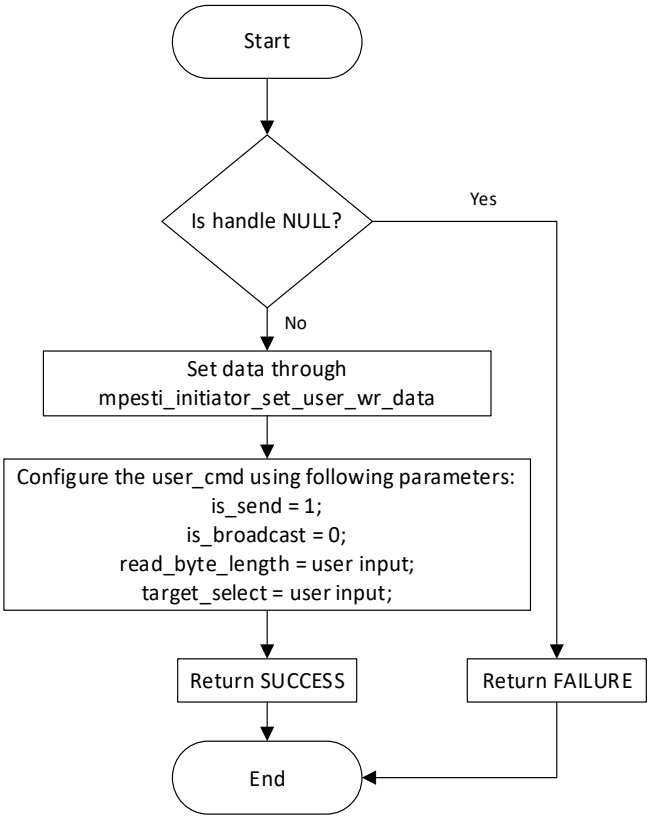

Figure 3.12. mpesti_initiator_send_target_command()

3.13. mpesti_initiator_send_target_commands()

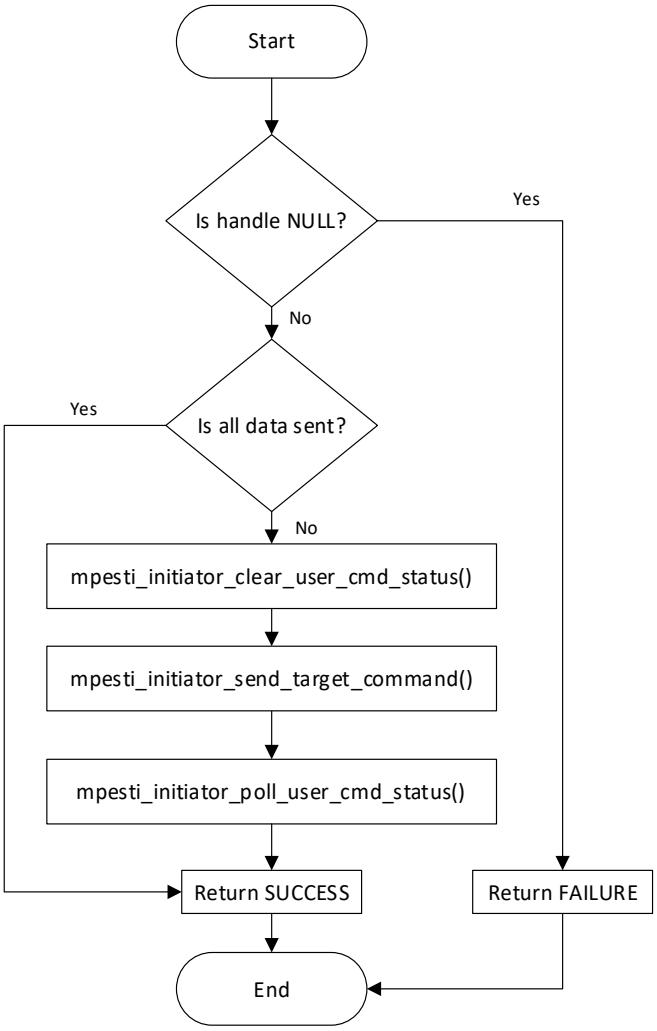

Figure 3.13. mpesti_initiator_send_target_commands()

3.14. mpesti_initiator_receive_target_commands()

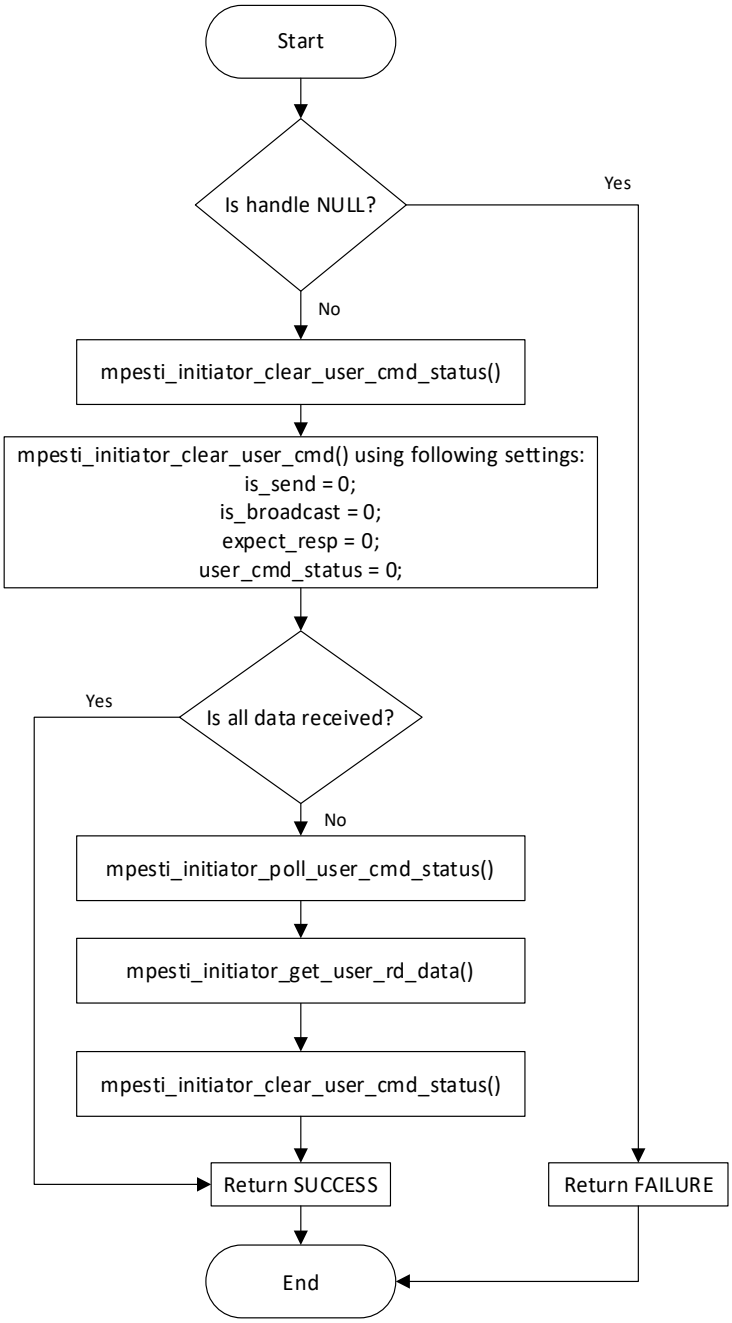

Figure 3.14. mpesti_initiator_receive_target_commands()

3.15. mpesti_initiator_set_vwire_out()

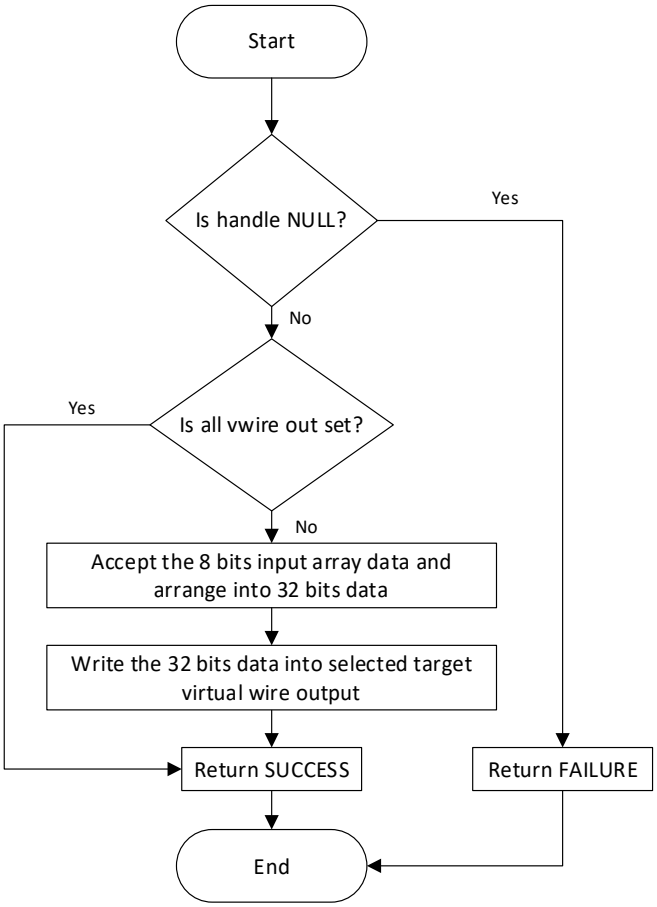

Figure 3.15. mpesti_initiator_set_vwire_out()

3.16. mpesti_initiator_get_vwire_in()

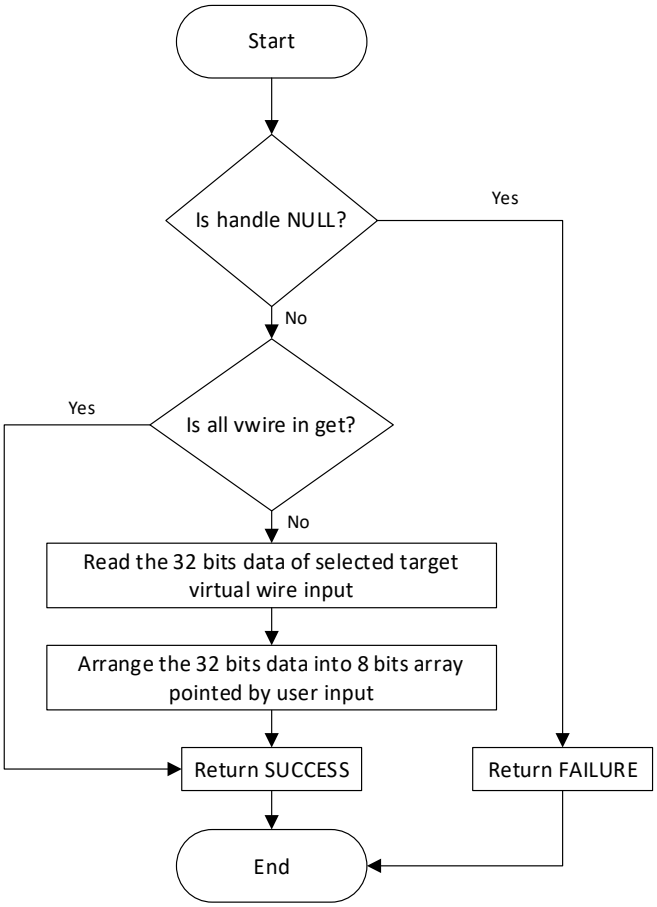

Figure 3.16. mpesti_initiator_get_vwire_in()

3.17. mpesti_initiator_manual_virtual_wire_exchange()

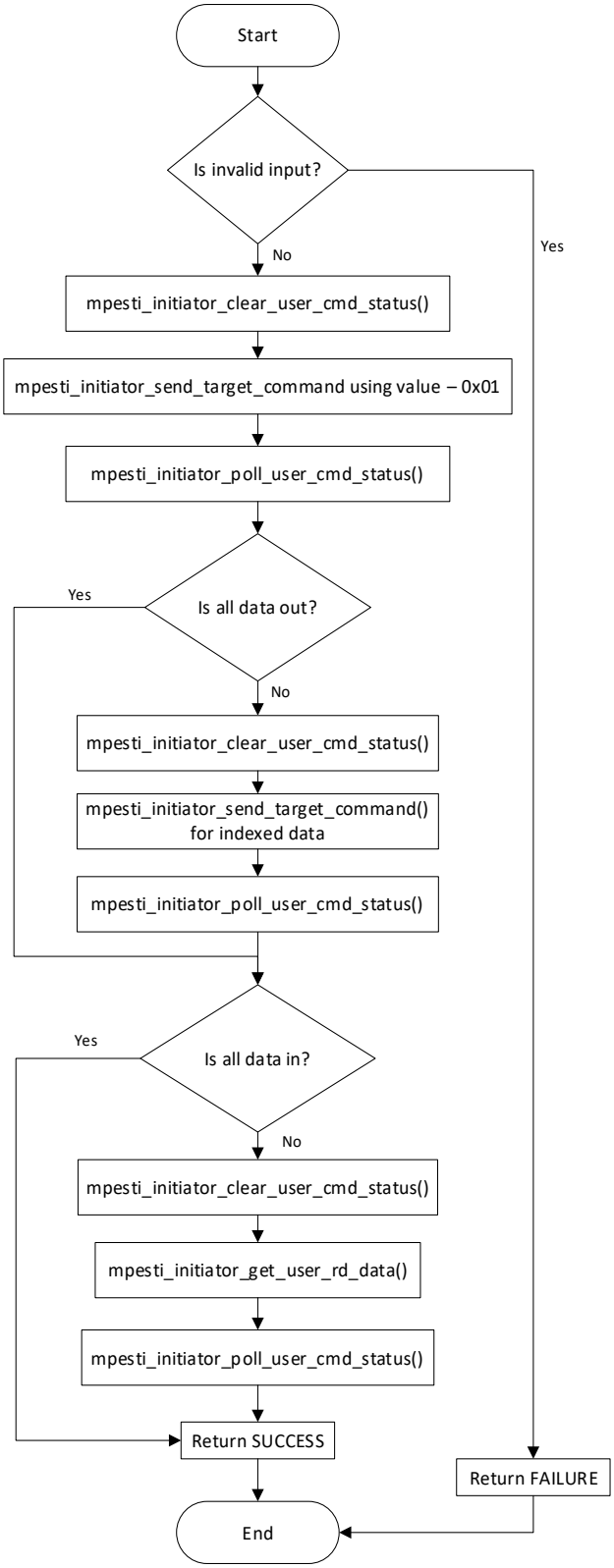

Figure 3.17. mpesti_initiator_manual_virtual_wire_exchange()

3.18. mpesti_initiator_configure_discovery_phase()

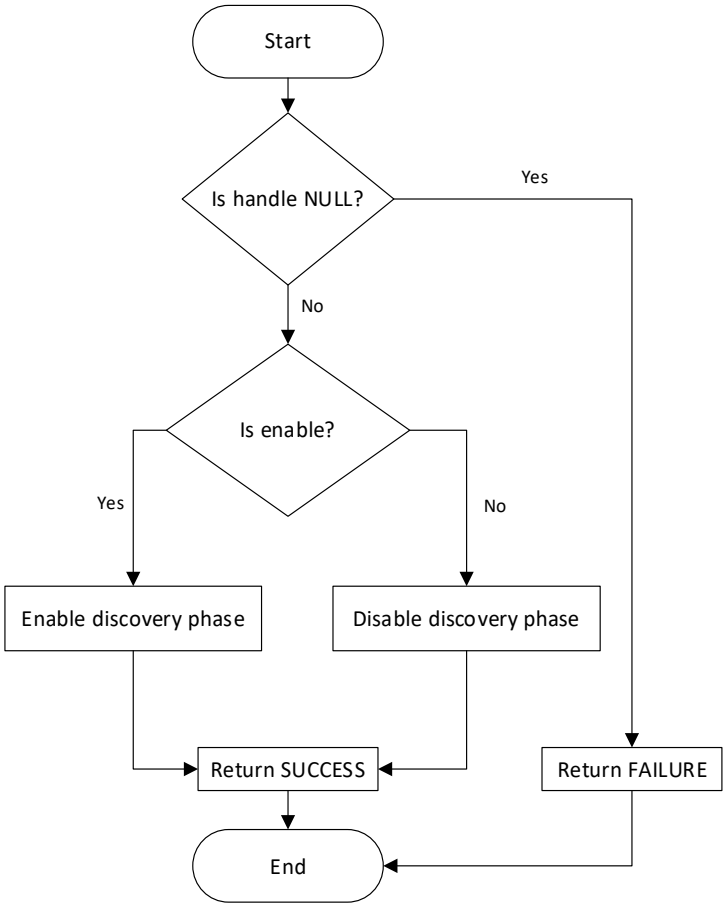

Figure 3.18. mpesti_initiator_configure_discovery_phase()

3.19. mpesti_initiator_configure_active_phase()

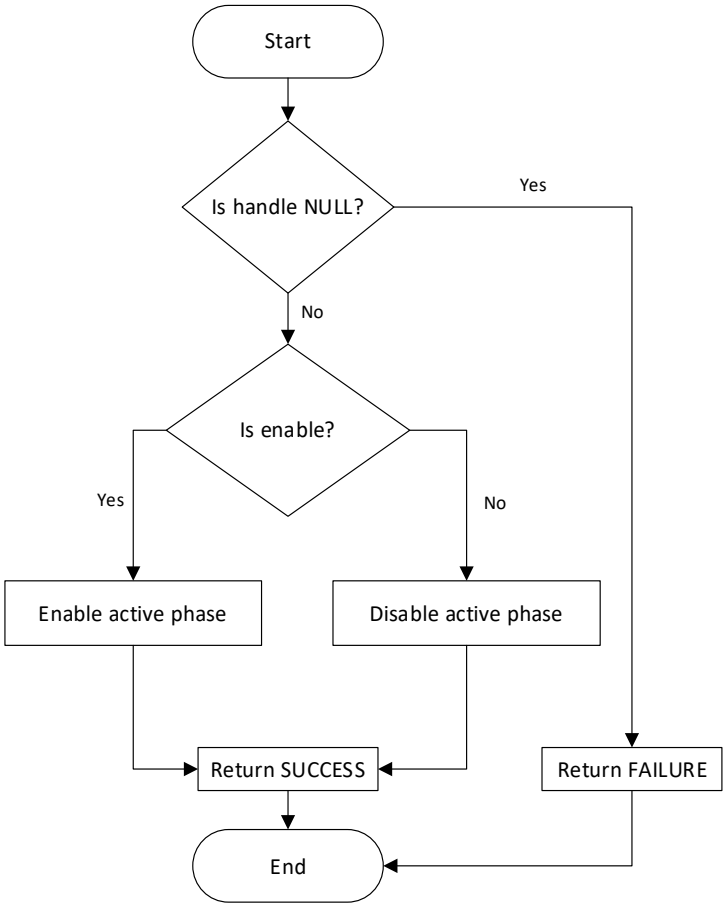

Figure 3.19. mpesti_initiator_configure_active_phase()

3.20. mpesti_initiator_get_target_phase_status()

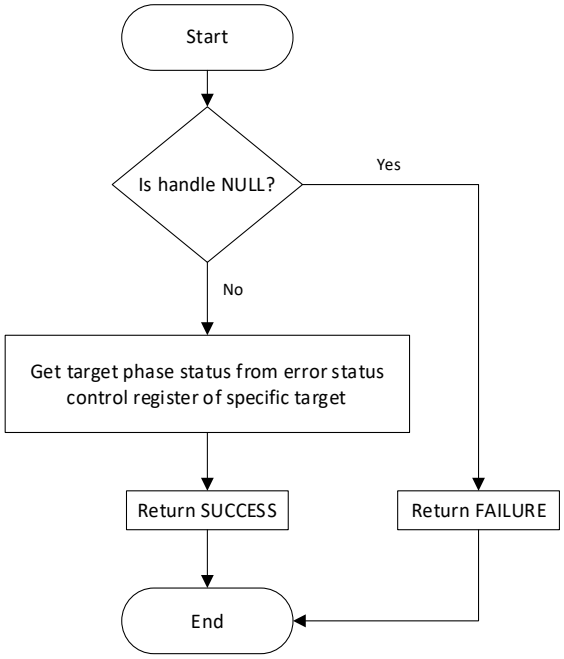

Figure 3.20. mpesti_initiator_get_target_phase_status()

3.21. mpesti_initiator_get_target_discovery_status()

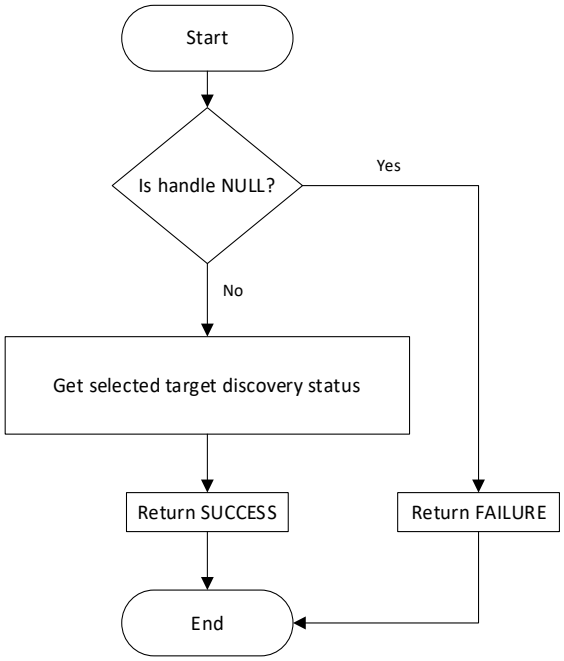

Figure 3.21. mpesti_initiator_get_target_discovery_status()

3.22. mpesti_initiator_get_target_error_status_control()

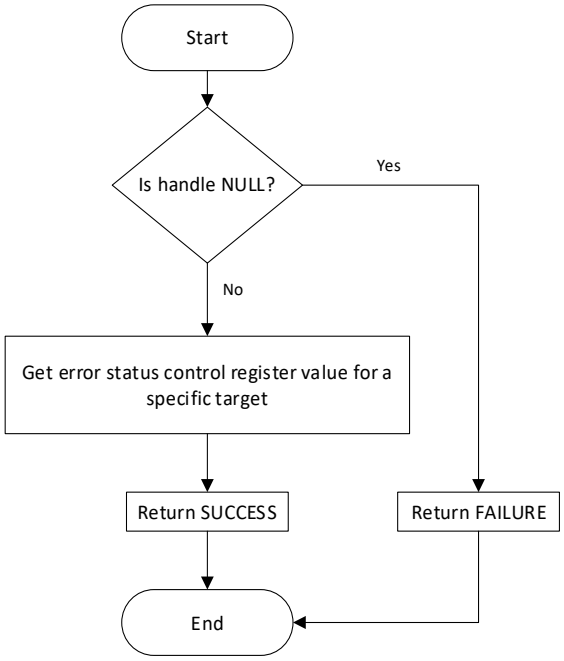

Figure 3.22. mpesti_initiator_get_target_error_status_control()

3.23. mpesti_initiator_set_target_error_status_control()

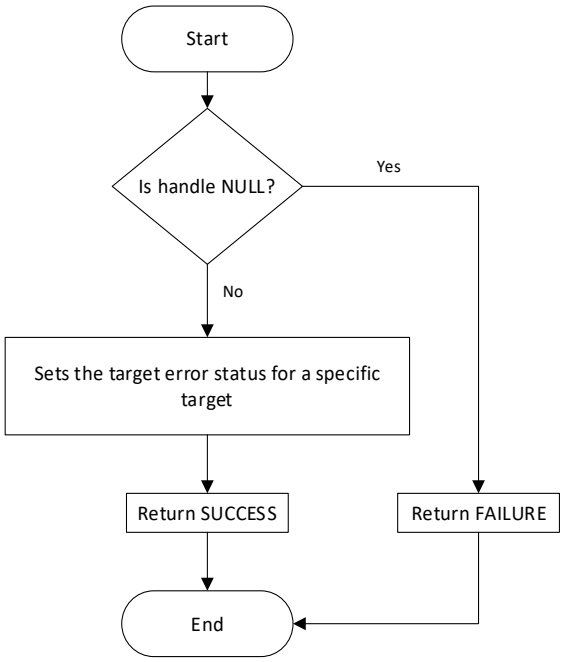

Figure 3.23. mpesti_initiator_set_target_error_status_control()

3.24. mpesti_initiator_get_target_payload()

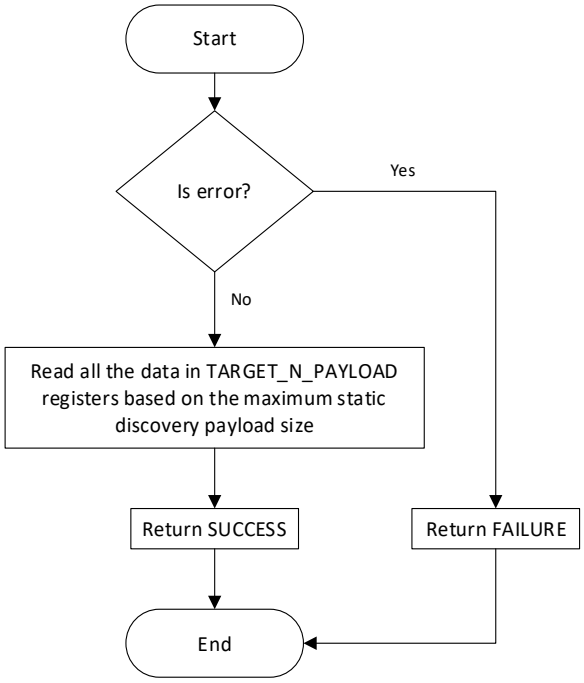

Figure 3.24. mpesti_initiator_get_target_payload()

3.25. mpesti_initiator_clear_target_error_status()

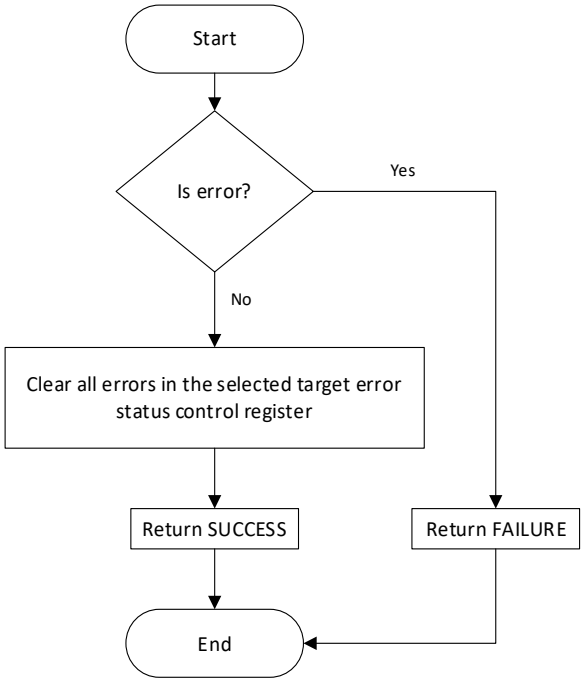

Figure 3.25. mpesti_initiator_clear_target_error_status()

3.26. mpesti_initiator_configure_interrupt_enable()

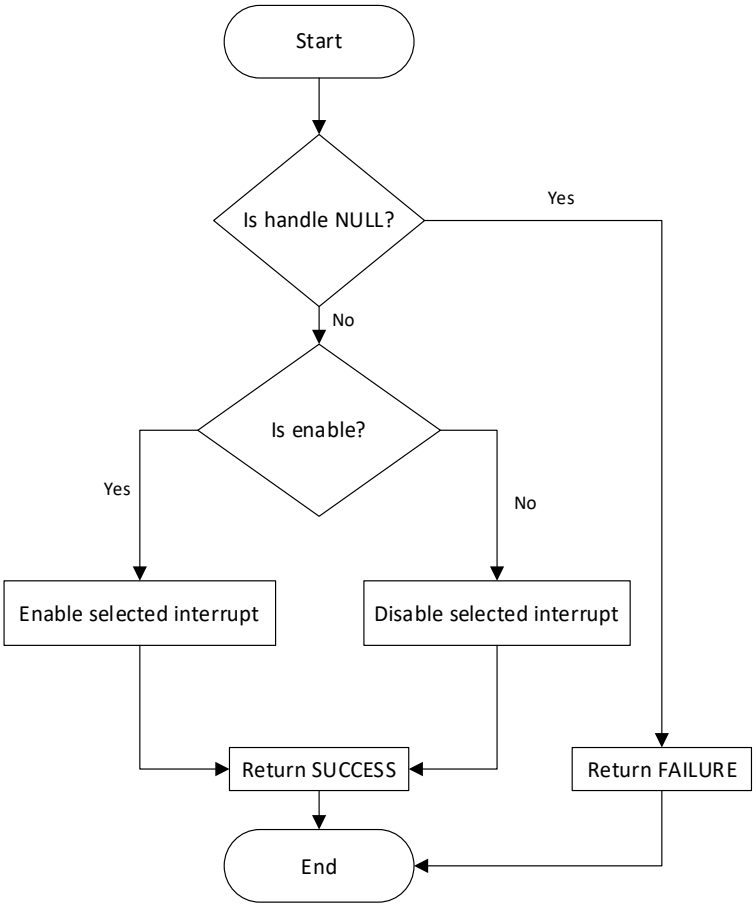

Figure 3.26. mpesti_initiator_configure_interrupt_enable()

3.27. mpesti_initiator_set_interrupt_set()

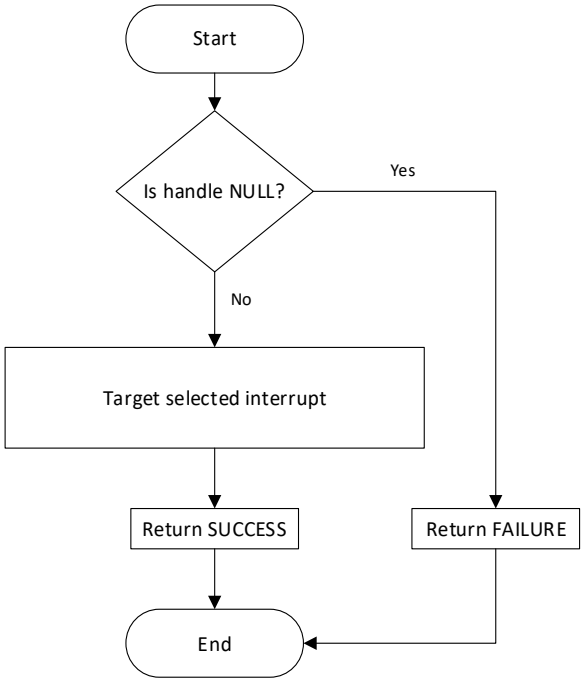

Figure 3.27. mpesti_initiator_set_interrupt_set()

3.28. mpesti_initiator_get_interrupt_status()

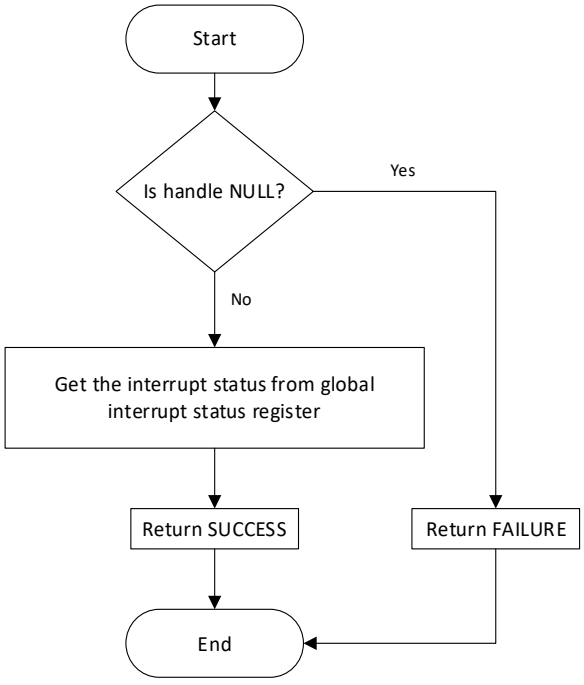

Figure 3.28. mpesti_initiator_get_interrupt_status()

3.29. mpesti_initiator_get_ip_info()

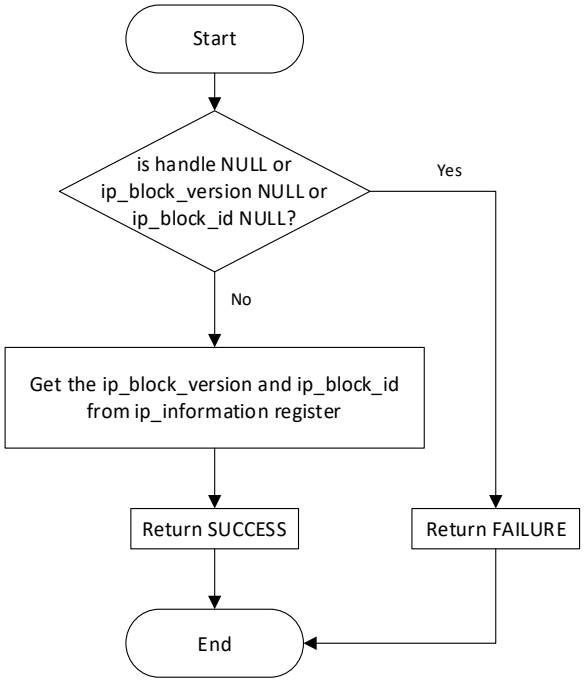

Figure 3.29. mpesti_initiator_get_ip_info()

3.30. mpesti_initiator_get_max_supported_targets()

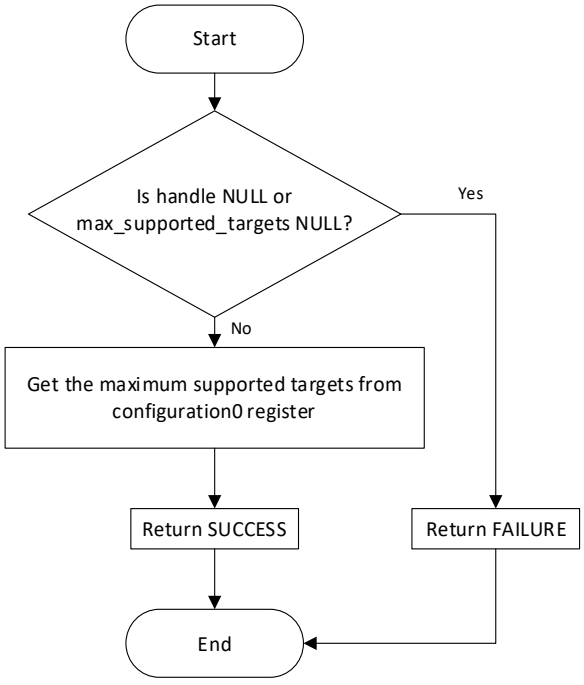

Figure 3.30. mpesti_initiator_get_max_supported_targets()

3.31. mpesti_initiator_get_max_supported_payload_size()

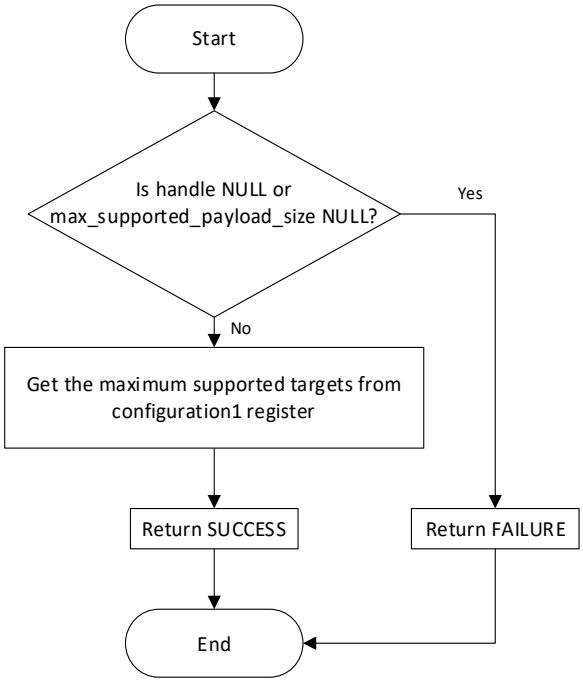

Figure 3.31. mpesti_initiator_get_max_supported_payload_size()

4. API Data Structures

4.1. Struct mpesti_initiator_handle_t

Table 4.1. mpesti_initiator_handle_t Parameters

Data Type	Struct Member	Description
uint32_t	base_addr	Base address of the M-PESTI IP.
uint32_t	cpu_timer_base_addr	Base address of CPU mtimer.
uint32_t	system_frequency_mhz	Frequency which is fed to CPU mtimer.

4.2. Struct mpesti_target_error_status_control_t

Refer to M-PESTI Initiator IP User Guide (FPGA-IPUG-02258) IP user guide for the detailed register description.

Table 4.2. mpesti_target_error_status_control_t Parameters

Data Type	Fields Struct Members	Description
uint32_t	discovery_status : 2	Discovery status of target.
uint32_t	active_phase_status : 1	0 : Discovery phase 1 : Active phase
uint32_t	active_phase_error : 1	0 : No active phase error 1 : Active phase error
uint32_t	active_phase_receive_timeout : 1	0 : Active phase received no timeout error 1 : Active phase received timeout error
uint32_t	presence_detection_timeout : 1	0 : No presence timeout error 1 : Presence timeout occurred
uint32_t	discovery_phase_error : 1	0 : No discovery phase error 1 : Discovery phase error
uint32_t	discovery_retry_error : 1	0 : Has not reached three discovery retries 1 : Exceeded three unsuccessful discovery retries
uint32_t	parity_error : 1	0 : No parity error 1 : Parity error
uint32_t	discovery_phase_crc_error : 1	0 : No CRC error during discovery phase 1 : CRC error during discovery phase
uint32_t	active_phase_crc_error : 1	0 : No CRC error during active phase 1 : CRC error during active phase
uint32_t	reserved1 : 5	Reserved.
uint32_t	discovery_phase_enable : 1	When set, initiator sends discovery phase request command.
uint32_t	discovery_phase_bypass : 1	When set, active phase is allowed on target without successful M-PESTI payload received.
uint32_t	active_phase_enable : 1	When set, initiator sends active phase command to start virtual wire exchange.
uint32_t	active_phase_pec_enable : 1	When set, initiator will be on active phase PEC during active phase.
uint32_t	reserved2 : 4	Reserved.
uint32_t	broadcast_subcription : 1	When set, allows target to receive broadcast message.
uint32_t	enable_presence_timeout_detection_interrupt_status : 1	When set, presence timeout detection interrupt status is enabled.
uint32_t	enable_discovery_phase_error_interrupt_status : 1	When set, discovery phase error interrupt status is enabled.

uint32_t	enable_active_phase_error_interrupt_status : 1	When set, active phase error interrupt status is enabled.
uint32_t	enable_target_presence_interrupt_status : 1	When set, target presence interrupt status is enabled.
uint32_t	enable_target_mpesti_ready_interrupt_status : 1	When set, target ready interrupt status is enabled.
uint32_t	enable_target_payload_good_interrupt_status : 1	When set, target payload good interrupt status is enabled.
uint32_t	enable_target_vwire_input_update_interrupt_status : 1	When set, target VWIRE input update interrupt status is enabled.

4.3. mpesti_target_error_status_control_u

Table 4.3. mpesti_target_error_status_control_u Parameters

Data Type	Struct Member	Description
uint32_t	value	Value of specific target error status control register.
mpesti_target_error_status_control_t	bit	Bit access of each specific bit in error status control register.

5. API Macros

Table 5.1. API Macros Description

Macro	Description
#define MPESTI_INITIATOR_DRV_VER	M-PESTI Initiator Driver Version.
#define IP_INFORMATION	IP information.
#define CONFIGURATION0	Information on maximum targets supported.
#define CONFIGURATION1	Information on maximum payload size supported.
#define GLOBAL_SOFTWARE_RESET	Global software reset.
#define DISCOVERY_PAYLOAD	Discovery payload page select.
#define USER_COMMAND_STATUS	User command status.
#define USER_COMMAND	User command.
#define USER_WR_DATA	User write data.
#define USER_RD_DATA	User read data.
#define GLOBAL_OPTIONAL_REGISTER_OFFSET	Global optional register offset which is 0x1000.
#define CLOCK_CONFIGURATION	Clock configuration.
#define SOFTWARE_RESET	Software reset.
#define GLOBAL_INTERRUPT_STATUS	Global interrupt status.
#define GLOBAL_INTERRUPT_ENABLE	Global interrupt enable.
#define GLOBAL_INTERRUPT_SET	Global interrupt set.
#define USER_COMMAND_INTERRUPT_ENABLE	User command interrupt enable.
#define USER_COMMAND_INTERRUPT_SET	User command interrupt set.
#define SECONDARY_WIRE_CONTROL_STATUS	Secondary wire control status.
#define SECONDARY_WIRE_SELECT	Secondary wire select.
#define TARGET_REGISTER_OFFSET	Target register offset which is 0x10000.
#define TARGET_N_ERROR_STATUS_CONTROL(N)	Specific target error status control register.
#define TARGET_PAYLOAD_OFFSET	Target payload offset which is 0x20000.
#define TARGET_N_PAYLOAD(SIZE, N)	Specific target payload register.
#define TARGET_VIRTUAL_WIRE_REGISTER_OFFSET	Target virtual wire register offset which is 0x30000.
#define TARGET_VIRTUAL_N_WIRE_INPUT(N)	Specific target virtual wire input register.
#define TARGET_VIRTUAL_N_WIRE_OUTPUT(N)	Specific target virtual wire output register.
#define TARGET_OPTIONAL_REGISTER_OFFSET	Target optional register offset which is 0x40000.
#define TARGET_VIRTUAL_N_WIRE_CONFIGURATION(N)	Specific target optional virtual wire configuration.
#define ENABLE_SW_RESET	Enable software reset.
#define USER_COMMAND_RECEIVE_TIMEOUT	User command receive timeout.
#define USER_COMMAND_DONE	User command done.
#define USER_COMMAND_RECEIVE_DATA_AVAILABLE	User command receive data available.
#define SEND_DATA	Configuration bit to send data.
#define EXPECT_DATA_ABORT	Configuration bit to expect return data when in sending move or abort current transmission in broadcast mode.
#define BROADCAST	Configuration bit to enable broadcast.
#define BYTE_LENGTH_START_POSITION	Offset to configure the byte length for receive.
#define BYTE_LENGTH_SIZE	Byte length for receive.
#define TARGET_SELECT_START_POSITION	Offset to configure the selected target.
#define TARGET_SELECT_SIZE	Maximum configurable selected target.
#define NO_AUTO_CLEAR	No auto clear.
#define CSR_RESET	Reset control status register.
#define IP_CORE_RESET	Reset IP core.
#define VIRTUAL_WIRE_INPUT_UPDATE_DETECTED	Virtual wire input update detected.

Macro	Description
#define PAYLOAD_GOOD_DETECTED	Payload good detected.
#define MPESTI_READY_DETECTED	M-PESTI ready detected.
#define TARGET_PRESENT_DETECTED	Target present detected.
#define ACTIVE_PHASE_ERROR_DETECTED	Active phase error detected.
#define DISCOVERY_PHASE_ERROR_DETECTED	Discovery phase error detected.
#define PRESENCE_TIMEOUT_DETECTED	Presence timeout detected.
#define VIRTUAL_WIRE_INPUT_UPDATE_ENABLE	Virtual wire input update enable.
#define PAYLOAD_GOOD_ENABLE	Payload good enable.
#define MPESTI_READY_ENABLE	M-PESTI ready enable.
#define TARGET_PRESENT_ENABLE	Target present enable.
#define ACTIVE_PHASE_ERROR_ENABLE	Active phase error enable.
#define DISCOVERY_PHASE_ERROR_ENABLE	Discovery phase error enable.
#define PRESENCE_TIMEOUT_ENABLE	Presence timeout enable.
#define VIRTUAL_WIRE_INPUT_UPDATE_SET	Virtual wire input update set.
#define PAYLOAD_GOOD_SET	Payload good set.
#define MPESTI_READY_SET	M-PESTI ready set.
#define TARGET_PRESENT_SET	Target present set.
#define ACTIVE_PHASE_ERROR_SET	Active phase error set.
#define DISCOVERY_PHASE_ERROR_SET	Discovery phase error set.
#define PRESENCE_TIMEOUT_SET	Presence timeout set.
#define USER_COMMAND_RECEIVE_TIMEOUT_INTERRUPT_ENABLE	User command receive timeout interrupt enable.
#define USER_COMMAND_DONE_INTERRUPT_ENABLE	User command done interrupt enable.
#define USER_COMMAND_RECEIVE_DATA_AVAILABLE_INTERRUPT_ENABLE	User command receive data available interrupt enable.
#define USER_COMMAND_RECEIVE_TIMEOUT_INTERRUPT_SET	User command receive timeout interrupt set.
#define USER_COMMAND_DONE_INTERRUPT_SET	User command done interrupt set.
#define USER_COMMAND_RECEIVE_DATA_AVAILABLE_INTERRUPT_SET	User command receive data available interrupt set.
#define LOWER_BITS_WIRE_SELECT_START_POSITION	Lower bits wire select start position.
#define LOWER_BITS_WIRE_SELECT_SIZE	Lower bits wire select size.
#define SECONDARY_WIRE	Secondary wire.
#define SECONDARY_WIRE_STIMULAS	Secondary wire stimulas.
#define ENABLE_TARGET_VWIRE_INPUT_UPDATE_INTERRUPT_STATUS	Enable target vwire input update interrupt status.
#define ENABLE_TARGET_PAYLOAD_GOOD_INTERRUPT_STATUS	Enable target payload good interrupt status.
#define ENABLE_TARGET_MPESTI_READY_INTERRUPT_STATUS	Enable target M-PESTI ready interrupt status.
#define ENABLE_TARGET_PRESENCE_INTERRUPT_STATUS	Enable target presence interrupt status.
#define ENABLE_ACTIVE_PHASE_ERROR_INTERRUPT_STATUS	Enable active phase error interrupt status.
#define ENABLE_DISCOVERY_PHASE_ERROR_INTERRUPT_STATUS	Enable discovery phase error interrupt status.
#define ENABLE_PRESENCE_TIMEOUT_DETECTION_INTERRUPT_STATUS	Enable presence timeout detection interrupt status.
#define BROADCAST_SUBCRIPTION	Broadcast subscription.
#define ACTIVE_PHASE_PEC_ENABLE	Active phase PEC enable.
#define ACTIVE_PHASE_ENABLE	Active phase enable.
#define DISCOVERY_PHASE_BYPASS	Discovery phase bypass.
#define DISCOVERY_PHASE_ENABLE	Discovery phase enable.
#define ACTIVE_PHASE_CRC_ERROR	Active phase CRC error.
#define DISCOVERY_PHASE_CRC_ERROR	Discovery phase CRC error.
#define PARITY_ERROR	Parity error.

Macro	Description
#define DISCOVERY_RETRY_ERROR	Discovery retry error.
#define DISCOVERY_PHASE_ERROR	Discovery phase error.
#define PRESENCE_DETECTION_TIMEOUT	Presence detection timeout.
#define ACTIVE_PHASE_RECEIVE_TIMEOUT	Active phase receive timeout.
#define ACTIVE_PHASE_ERROR	Active phase error.
#define ACTIVE_PHASE_STATUS	Active phase status.
#define DISCOVERY_STATUS1	Discovery status 1.
#define DISCOVERY_STATUS0	Discovery status 0.
#define DISCOVERY_STATUS_ABSENT	Discovery status absent.
#define DISCOVERY_STATUS_PRESENT	Discovery status present.
#define DISCOVERY_STATUS_GOOD_PAYLOAD	Discovery status good payload.
#define DISCOVERY_STATUS_NO_PAYLOAD	Discovery status no payload.
#define FAILURE	0
#define SUCCESS	1

References

- M-PESTI Initiator IP User Guide (FPGA-IPUG-02258)
- M-PESTI Initiator IP Release Notes (FPGA-RN-02005)
- M-PESTI Initiator Reference Design (FPGA-RD-02312)
- M-PESTI Initiator Reference Design web page
- Modular-Peripheral Sideband Tunneling Interface (M-PESTI) Base Specification Version 1.0 Release Candidate 2
- Modular-Peripheral Sideband Tunneling Interface (M-PESTI) Base Specification Version 1.2 Release Candidate 2
- Open Compute Project® web page
- Lattice Propel 2024.1 Builder User Guide (FPGA-UG-02212)
- Lattice Solutions IP Cores web page
- Lattice Propel Design Environment web page
- Lattice Radiant Software 2025.1 User Guide
- Lattice Radiant Software 2024.2 User Guide
- Lattice Radiant FPGA design software
- Lattice Insights 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 1.2, December 2025

Section	Change Summary
Introduction	Updated the following in the Driver and IP Compatibility section: • Driver Version to 25.02.00 • IP Version to 2.0.1 • Tested hardware device to MachXO5-NX • Lattice Propel Builder, Lattice Propel SDK, Lattice Radiant Software, and Radiant Programmer to 2025.1.1
API Description	Updated these sections including section titles, tables, and figures to reflect on Driver and IP version changes.
Function Call Flow Diagrams	
API Data Structures	
API Variables	Removed the following section.
API Macros	Updated Table 5.1. API Macros Description to reflect on Driver and IP version changes.
References	Added Modular-Peripheral Sideband Tunneling Interface (M-PESTI) Base Specification Version 1.2 Release Candidate 2.

Revision 1.1, September 2025

Section	Change Summary
All	Added the OCP logo to the cover page.
Abbreviations in This Document	Added *Open Compute Project (OCP)*.
Introduction	Added a statement indicating that the IP is OCP Ready.
References	Added the Open Compute Project web page.

Revision 1.0, July 2025

Section	Change Summary
All	Initial release.

www.latticesemi.com