I3C Controller Driver API Reference

Technical Note

FPGA-TN-02342-1.0

December 2023

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 REDUDANCIES, 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

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
ACK	Acknowledgement
AHB	Advanced High-Performance Bus
APB	Advanced Peripheral Bus
API	Application Programming Interfaces
BCR	Bus Characteristic Register
CCC	Common Control Code
CRC	Cyclic Redundancy Check
DAA	Dynamic Address Assignment
DCR	Device Characteristic Register
DDR	Double Data Rate
DWORD	Double Word
FIFO	First In First Out
FPGA	Field Programmable Gate Array
HDR	High Data Rate
HJ	Hot Join
IBI	In Band Interrupt
IP	Intellectual Property
I2C	Inter-Integrated Circuit
I3C	Improved Inter-Integrated Circuit
LMMI	Lattice Memory Mapped Interface
NACK	No Acknowledgement
PID	Provisional ID
RISC	Reduced Instruction Set Computer
SCL	Serial Clock
SDK	Software Development Kit
SDR	Single Data Rate

1. Introduction

The I3C IP is a two-wire bi-directional serial bus, optimized for multiple sensor secondary devices and controlled by only one I3C controller device at a time. The I3C protocol is backward compatible with many legacy I2C devices. The protocol supports significantly higher speeds, new communication modes, and new device roles, including an ability to change device roles over time when connected to I3C devices. For more details on the I3C Controller IP core, refer to the I3C Controller IP Core User Guide (FPGA-IPUG-02228).

1.1. Purpose

I3C Controller and its SDK is a set of application programming interfaces (APIs) that provides access to Lattice-specific hardware and software capabilities. This document is intended to act as a reference guide for developers by providing details of the I3C controller C-language driver API description, function call flow diagrams, API data structures, API variables, and API macros.

1.2. Audience

The intended audience for this document includes embedded system designers and embedded software developers using Lattice MachXO2™, MachXO3D™, MachXO3L™, MachXO3LF™, CrossLink™-NX, Certus™-NX, CertusPro™-NX, Mach™-NX, MachXO5™-NX, and Lattice Avant™ devices. The technical guidelines assume readers have expertise in the embedded system area and FPGA technologies.

1.3. Driver Versioning

1.3.1. Driver Version

Driver version: 1.0.0

1.3.2. IP Version

IP version: 3.2.0

1.4. Driver and IP Compatibility

The following IPs are used for the co-verification testing:

Table 1.1. IP and IP's version

IP	IP version
I3C Controller	3.2.0
I3C Target	3.2.0
Osc0	1.4.0
PLL	1.7.0
AHB Lite Interconnect	1.3.0
System Memory	2.0.0
AHB Lite to APB bridge	1.1.0
APB interconnect	1.2.0
RISC-V MC	2.4.0

2. API Description

The I3C controller can control peripheral devices or target devices by writing to and reading from the configuration registers. The APIs are listed below:

2.1. i3c_controller_init()

This API initializes the base address of the I3C controller. This function enables clock signal, interrupt signal, and set mode of transfer.

```
unsigned int i3c_controller_init(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	mode	• Member of the i3c_controller_handle_t structure. • Sets the mode to I3C or I2C.	

2.2. i3c_controller_daa_1tgt()

This API is used for dynamic address assignment. This function is used to change the target address from static to dynamic using BCR, DCR, and Provisional ID.

```
unsigned int i3c_controller_daa_1tgt(i3c_controller_handle_t *handle,i3c_dev_info_t *info)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
Out	info	• Member of the i3c_dev_info structure. • Stores the PID, DCR, and BCR values of each target.	
In	target_nums	• Member of the i3c_controller_handle_t structure. • Assigns the count of targets which is *1* for this API.	
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Assigns a dynamic address to the target.	

2.3. i3c_controller_entdaa_multitargets()

This API is used for dynamic address assignment. This function is used to change the target address from static to dynamic using BCR, DCR, and Provisional ID.

```
unsigned int i3c_controller_entdaa_multitargets(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_nums	• Member of the i3c_controller_handle_t structure. • Assigns the count of targets.	
In	target_dyn_addr_multi	• Member of the i3c_controller_handle_t structure. • Assigns a dynamic address to the target.	

2.4. i3c_controller_private_i3c_write()

This API is used to write in the I3C private transfer mode. It follows the defined user packet frame which writes to Tx FIFO through the LMMI interface.

```
unsigned int i3c_controller_private_i3c_write(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C target.	
In	wr_buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned.	
In	wr_len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data.	

2.5. i3c_controller_private_i3c_read()

This API is used to read in the I3C private transfer mode. The read data response from the Target device is stored in Rx FIFO and read through the LMMI interface.

```
unsigned int i3c_controller_private_i3c_read(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C target.	
Out	rd_buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is collected.	
In	rd_len	• Member of the i3c_controller_handle_t structure. • Buffer length for read data.	

2.6. i3c_controller_private_i2c_write()

This API is used to write in the I2C private transfer mode. It follows the defined user packet frame which writes to Tx FIFO through the LMMI interface.

```
unsigned int i3c_controller_private_i2c_write(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_static_addr	• Member of the i3c_controller_handle_t structure. • Static address of the I3C target.	
In	wr_buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned.	
In	wr_len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data.	

2.7. i3c_controller_private_i2c_read()

This API is used to read in the I2C private transfer mode. The read data response from the Target device is stored in Rx FIFO and read through the LMMI interface.

```
unsigned int i3c_controller_private_i2c_read(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_static_addr	• Member of the i3c_controller_handle_t structure. • Static address of the I3C target.	
Out	rd_buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is collected.	
In	rd_len	• Member of the i3c_controller_handle_t structure. • Buffer length for read data.	

2.8. i3c_controller_broadcast_ccc()

This API is used to check whether the active I3C controller sends the Broadcast CCC commands which address all Target registers.

```
unsigned int i3c_controller_broadcast_ccc(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned.	
In	len	• Member of the i3c_controller_handle_t structure. • Buffer length for write CCC command.	

2.9. i3c_controller_direct_ccc()

This API is used to check whether the active I3C controller sends the Direct CCC commands which address individual Target registers.

```
unsigned int i3c_controller_direct_ccc(i3c_controller_handle_t *handle,bool read1_write0)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C target.	
In	buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer assigned to the CCC Command.	
In	len	• Member of the i3c_controller_handle_t structure. • Buffer length for write CCC command.	
In	direct_ccc_optnal_bytes	• Member of the i3c_controller_handle_t structure. • Assigns the length of optional bytes to write or read in the register.	
In/Out	direct_ccc_optnal_data_array	• Member of the i3c_controller_handle_t structure. • Assigns optional bytes data to write or read.	

In/Out	Parameter	Description	Returns
In	Read1_write0	Assign *1* to read from *Rx FIFO*, and *0* to write to *Tx FIFO*.	

2.10. i3c_controller_hdr_ddr_write()

This API is used to write in the HDR mode. It converts from SDR to HDR mode. The function gives HDR command followed by HDR data words, and HDR CRC word. It has start and stop conditions for the HDR mode.

```
unsigned int i3c_controller_hdr_ddr_write(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C controller.	
In	wr_buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned in the HDR mode.	
In	wr_len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data in the HDR mode.	
In	user_7bit_code	• Member of the i3c_controller_handle_t structure. • Assigns write or read status.	

2.11. i3c_controller_hdr_ddr_read()

This API is used to read in the HDR mode. It converts from SDR to HDR mode. The function gives HDR command followed by HDR data words and HDR CRC word. It has start and stop conditions for the HDR mode.

```
unsigned int i3c_controller_hdr_ddr_read(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C controller.	
Out	rd_buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is collected in the HDR mode.	
In	rd_len	• Member of the i3c_controller_handle_t structure. • Buffer length for read data in the HDR mode.	
In	user_7bit_code	• Member of the i3c_controller_handle_t structure. • Assigns write or read status.	

2.12. i3c_controller_ibi_handling()

This API is used to enable the in-band interrupt request from Target. The function checks the status of Tx FIFO, IBI read count, and IBI response registers. It provides ACK or NACK to the controller. The function reads the mandatory and optional bytes and stores them in Rx FIFO.

```
unsigned int i3c_controller_ibi_handling(i3c_controller_handle_t *handle,bool ibi_config)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure

In/Out	Parameter	Description	Returns
In	ibi_read_count	• Member of the i3c_controller_handle_t structure. • Assigns the length of the IBI payload.	
Out	ibi_payload_data	• Member of the i3c_controller_handle_t structure. • Assigns the IBI payload data.	
In	ibi_config	Set ibi_config to *1* to assign IBI auto response low.	

2.13. i3c_controller_hj_handling()

This API is used to enable the Hot-Join request from Target. The function checks the status of Tx FIFO , IBI read count and IBI response registers. It provides ACK or NACK to the controller. The function reads the mandatory and optional bytes and stores them in Rx FIFO.

```
unsigned int i3c_controller_hj_handling(i3c_controller_handle_t *handle,bool ibi_config)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	ibi_config	Set ibi_config to *1* to assign IBI auto response low.	

2.14. i3c_controller_Contrlr_handoff()

This API is used to enable the Active controller role that can be transferred to another controller .This is applicable only when the Secondary Controller Capability is enabled.

```
unsigned int i3c_controller_Contrlr_handoff(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	direct_ccc_optnal_bytes	• Member of the i3c_controller_handle_t structure. • Assigns the length of optional bytes to write or read in the register.	
In	direct_ccc_optnal_data_array	• Member of the i3c_controller_handle_t structure. • Assigns optional bytes data to write or read.	

2.15. i3c_controller_Consecutive_private_i3c_write()

This API is used to write to 2 targets consecutively in the I3C private transfer mode. It follows the defined user packet frame which writes to Tx FIFO through the LMMI interface.

```
unsigned int i3c_controller_Consecutive_private_i3c_write(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C target.	
In	buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned.	
In	len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data.	

In/Out	Parameter	Description	Returns
In	target2_dyn_addr	• Member of the i3c_controller_handle_t structure. • I3C target dynamic address for target 2.	
In	len2	• Member of the i3c_controller_handle_t structure. • Buffer length of write data for target 2.	
In	buf2	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned for target 2.	

2.16. i3c_controller_Consecutive_private_i3c_write_read()

This API is used to write and read to 2 targets consecutively in the I3C private transfer mode. It follows the defined user packet frame which writes to Tx FIFO through the LMMI interface.

```
unsigned int i3c_controller_Consecutive_private_i3c_write_read(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C target.	
In	buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned.	
In	len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data.	
In	target2_dyn_addr	• Member of the i3c_controller_handle_t structure. • I3C target dynamic address for target 2.	
In	len2	• Member of the i3c_controller_handle_t structure. • Buffer length of read data for target 2.	
Out	buf2	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is stored for target 2.	

2.17. i3c_controller_Consecutive_private_i2c_write_read()

This API is used to write and read to 2 targets consecutively in the I2C private transfer mode. It follows the defined user packet frame which writes to Tx FIFO through the LMMI interface.

```
unsigned int i3c_controller_Consecutive_private_i2c_write_read(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C target.	
In	buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned in the I2C mode.	
In	len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data in the I2C mode.	
In	target2_dyn_addr	• Member of the i3c_controller_handle_t structure. • I3C target dynamic address for target 2.	
In	len2	• Member of the i3c_controller_handle_t structure. • Buffer length of read data for target 2.	

In/Out	Parameter	Description	Returns
Out	buf2	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is stored for target 2.	

2.18. i3c_controller_Consecutive_hdr_ddr_write_read()

This API is used to write and read to 2 targets consecutively in HDR-DDR transfer mode. It follows the defined user packet frame which writes to Tx FIFO through the LMMI interface.

```
unsigned int i3c_controller_Consecutive_hdr_ddr_write_read(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C target.	
In	buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned in the HDR mode.	
In	len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data in the HDR mode.	
In	target2_dyn_addr	• Member of the i3c_controller_handle_t structure. • I3C target dynamic address for target 2.	
In	len2	• Member of the i3c_controller_handle_t structure. • Buffer length of read data for target 2.	
Out	buf2	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is stored for target 2.	
In	user_7bit_code	• Member of the i3c_controller_handle_t structure. • Assigns write or read status.	

2.19. i3c_controller_hdr_broadcast_ccc()

This API is used to check whether the active I3C controller sends the Broadcast CCC commands which address all Target registers in the HDR mode.

```
unsigned int i3c_controller_hdr_broadcast_ccc(i3c_controller_handle_t *handle)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer where data is assigned in the HDR mode.	
In	len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data in the HDR mode.	
In	user_7bit_code	• Member of the i3c_controller_handle_t structure. • Assigns write or read status.	

2.20. i3c_controller_hdr_direct_ccc()

This API is used to check whether the active I3C controller sends the Direct CCC commands which address individual Target registers in the HDR mode.

```
unsigned int i3c_controller_hdr_direct_ccc(i3c_controller_handle_t *handle,bool read1_write0)
```

In/Out	Parameter	Description	Returns
In	base_addr	• Member of the i3c_controller_handle_t structure. • Base address of the I3C controller.	1: success 0: failure
In	target_dyn_addr	• Member of the i3c_controller_handle_t structure. • Dynamic address of the I3C controller.	
In	buf	• Member of the i3c_controller_handle_t structure. • Buffer pointer assigned to the CCC Command.	
In	len	• Member of the i3c_controller_handle_t structure. • Buffer length for write data.	
In	user_7bit_code	• Member of the i3c_controller_handle_t structure. • Assigns write or read status.	
In	direct_ccc_optnal_bytes	• Member of the i3c_controller_handle_t structure. • Assigns the length of optional bytes to write or read in the register.	
In/Out	direct_ccc_optnal_data_array	• Member of the i3c_controller_handle_t structure. • Assigns optional bytes data to write or read.	
In	Read1_write0	Assign *1* to read from *Rx FIFO*, and *0* to write to *Tx FIFO*.	

3. Function Call Flow Diagrams

3.1. i3c_controller_init flow

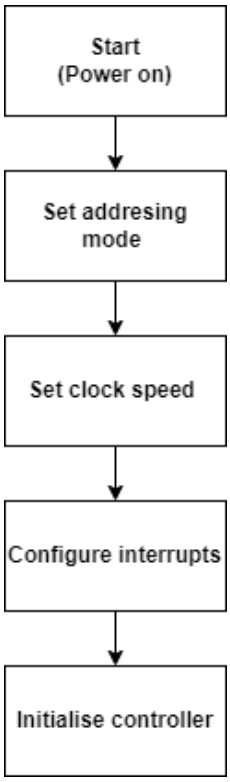

Figure 3.1. Initialization

3.2. i3c_controller_daa_1tgt flow

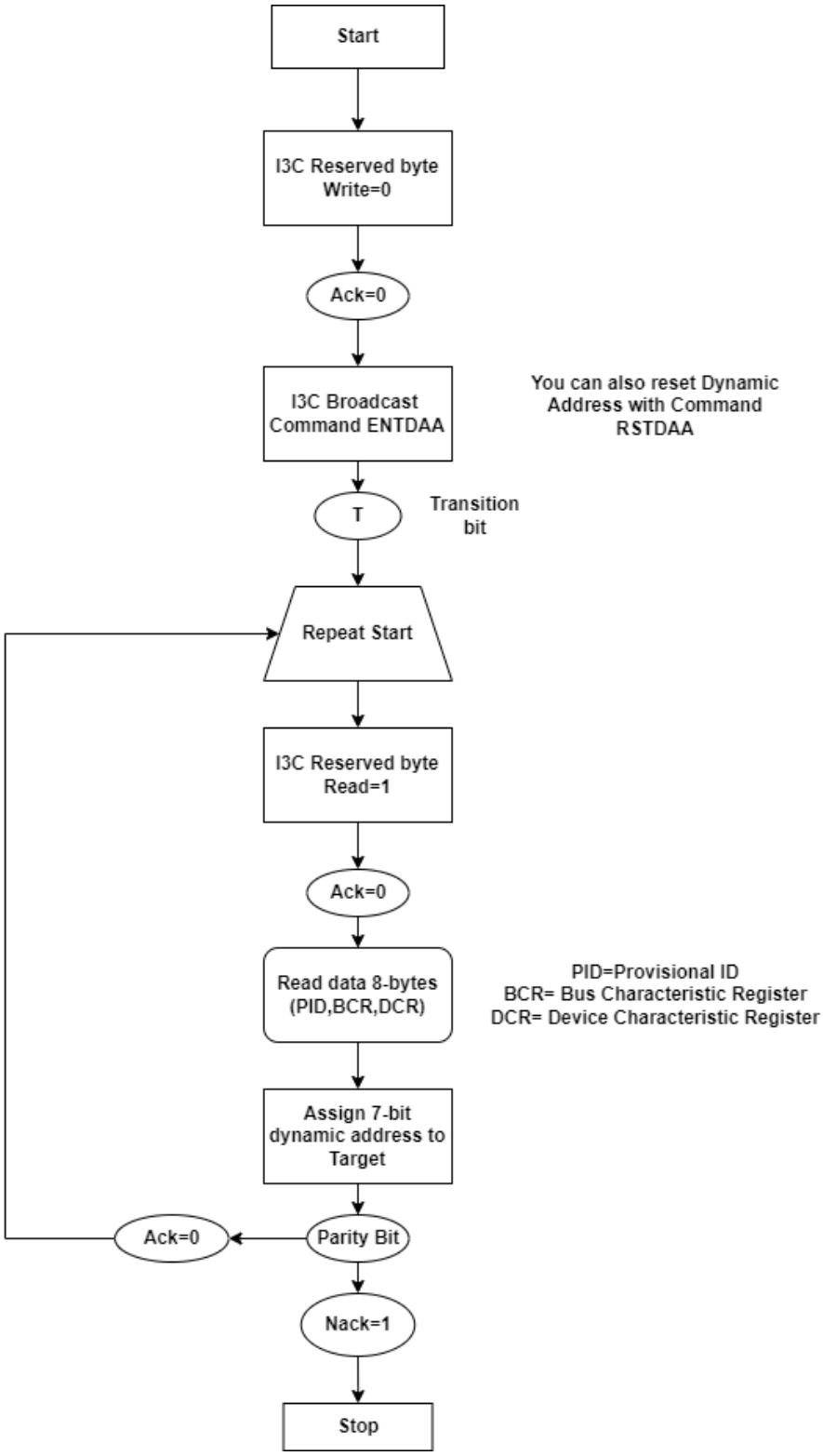

Figure 3.2. i3c_controller_daa_1tgt

3.3. i3c_controller_entdaa_multitargets flow

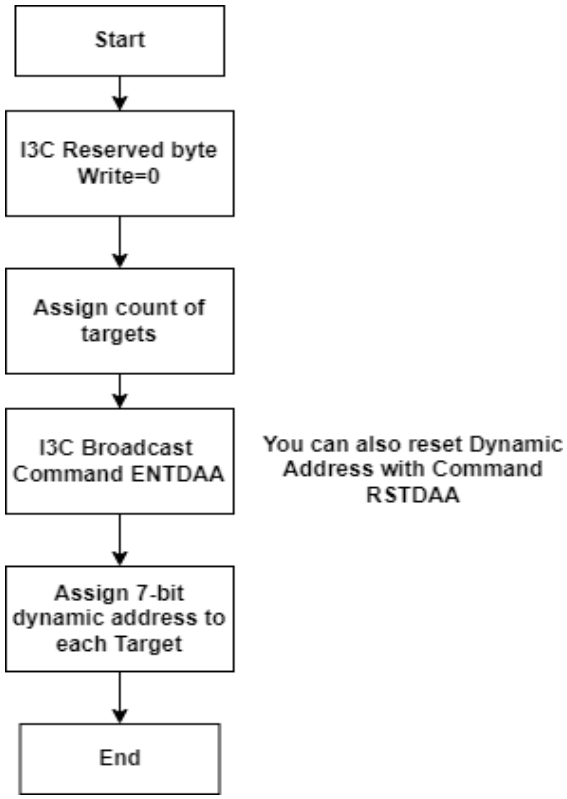

Figure 3.3. i3c_controller_entdaa_multitargets

3.4. i3c_controller_private_i3c_write flow

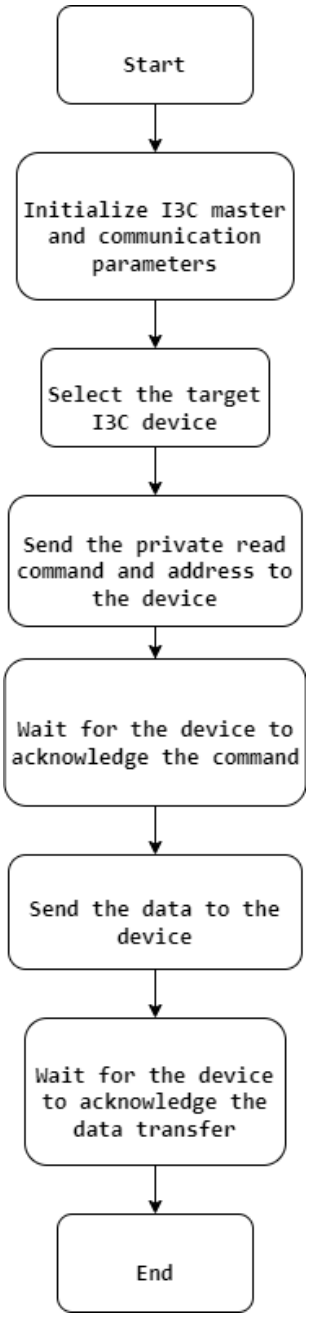

Figure 3.4. i3c_controller_private_i3c_write

3.5. i3c_controller_private_i3c_read flow

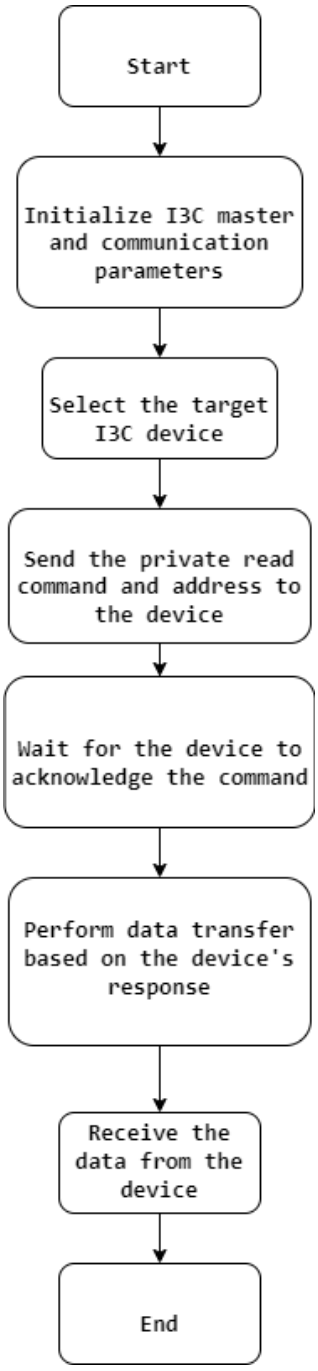

Figure 3.5. i3c_controller_private_i3c_read

3.6. i3c_controller_private_i2c_write flow

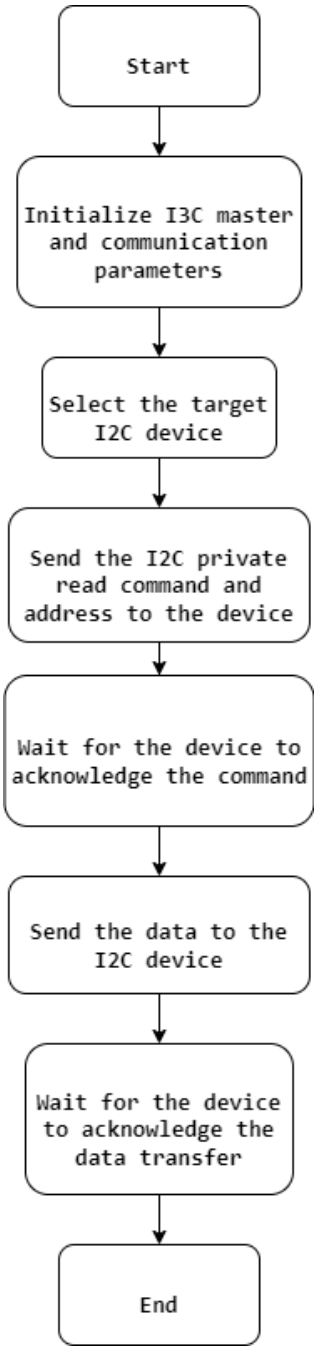

Figure 3.6. i3c_controller_private_i2c_write

3.7. i3c_controller_private_i2c_read flow

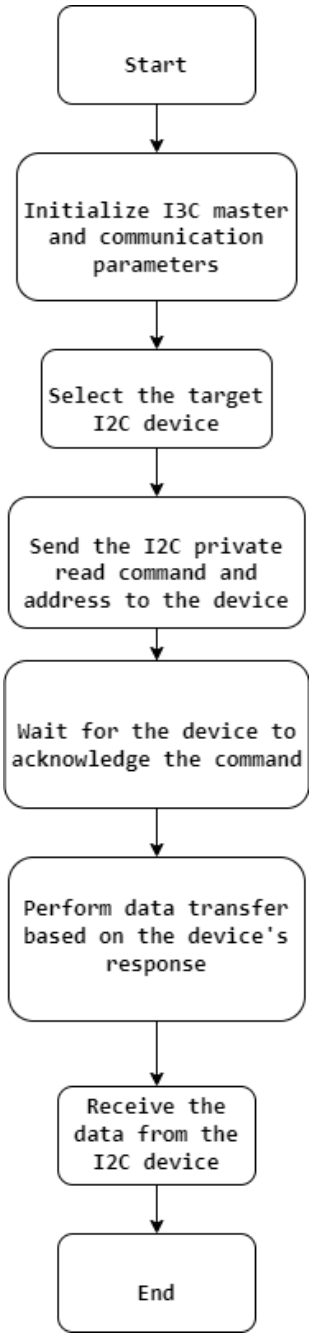

Figure 3.7. i3c_controller_private_i2c_read

3.8. i3c_controller_broadcast_ccc flow

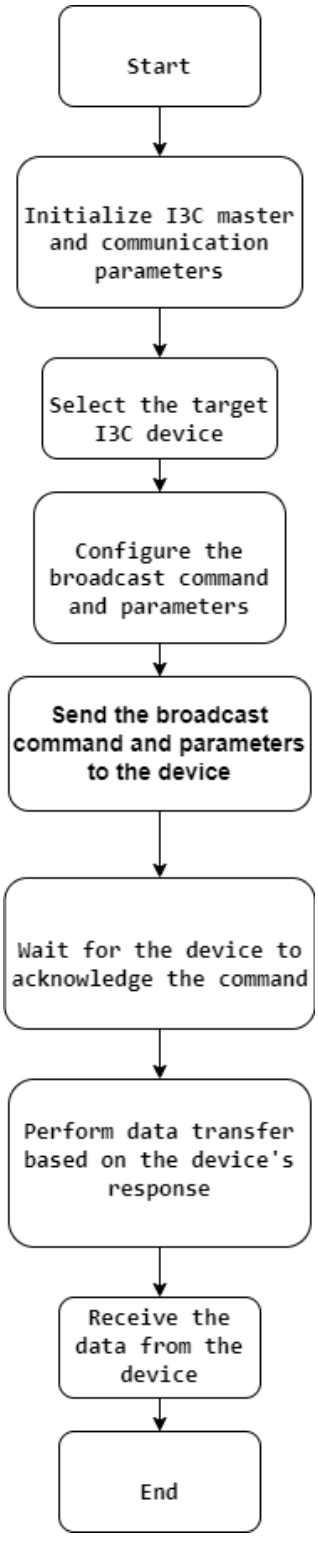

Figure 3.8. i3c_controller_broadcast_ccc

3.9. i3c_controller_direct_ccc flow

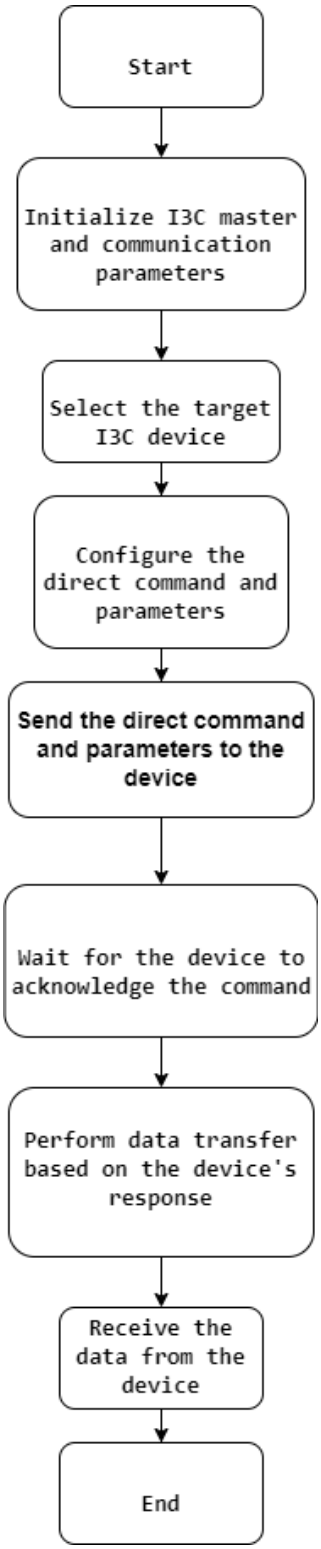

Figure 3.9. i3c_controller_direct_ccc

3.10. i3c_controller_hdr_ddr_write flow

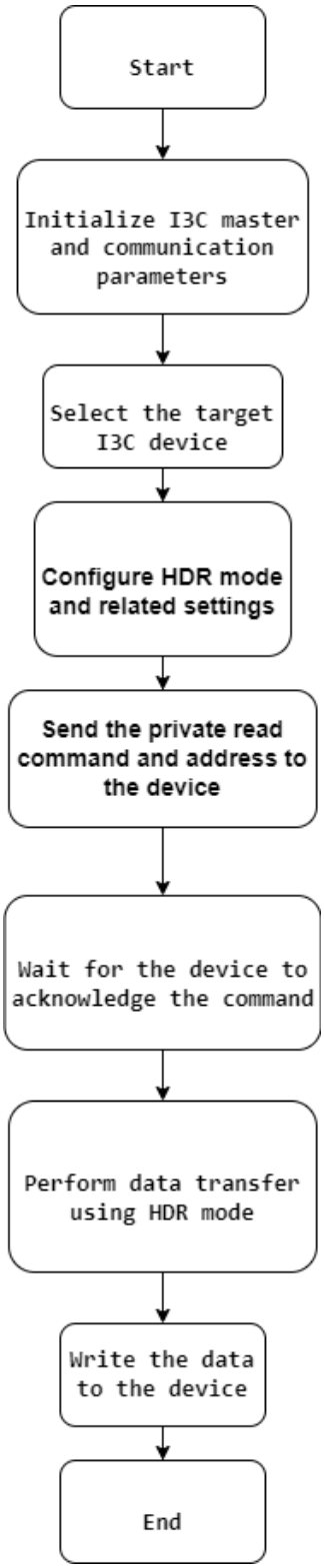

Figure 3.10. i3c_controller_hdr_ddr_write

3.11. i3c_controller_hdr_ddr_read flow

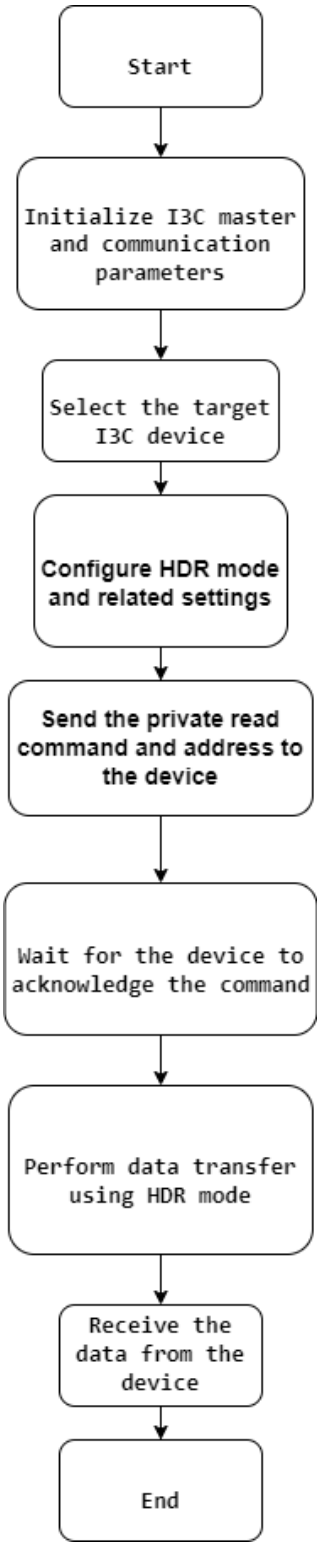

Figure 3.11. i3c_controller_hdr_ddr_read

3.12. i3c_controller_ibi_handling flow

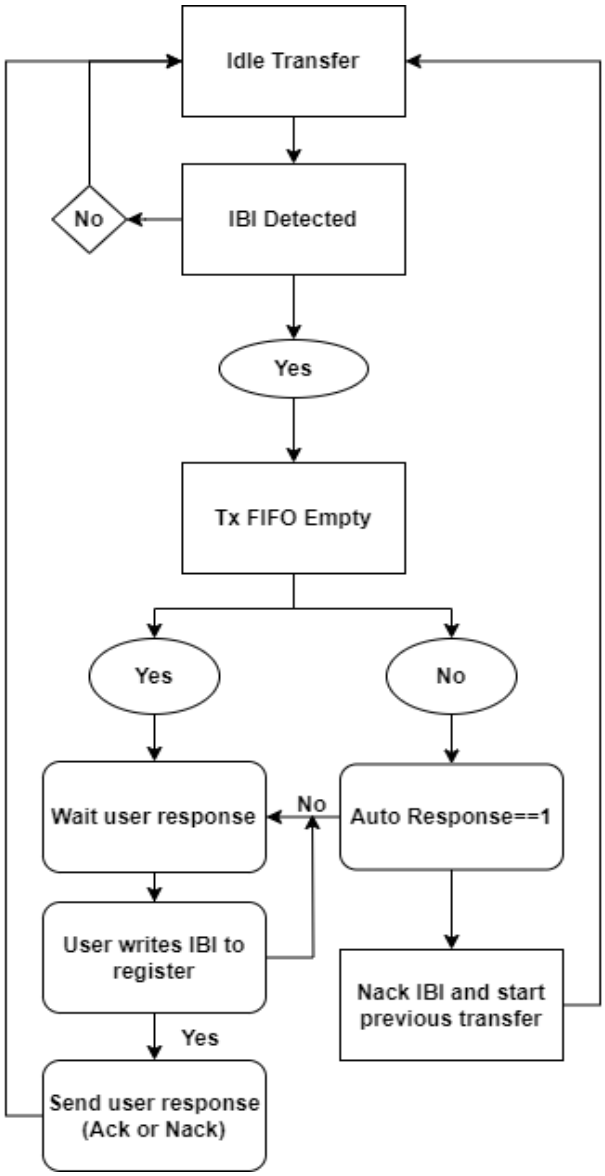

Figure 3.12. i3c_controller_ibi_handling

3.13. i3c_controller_hj_handling flow

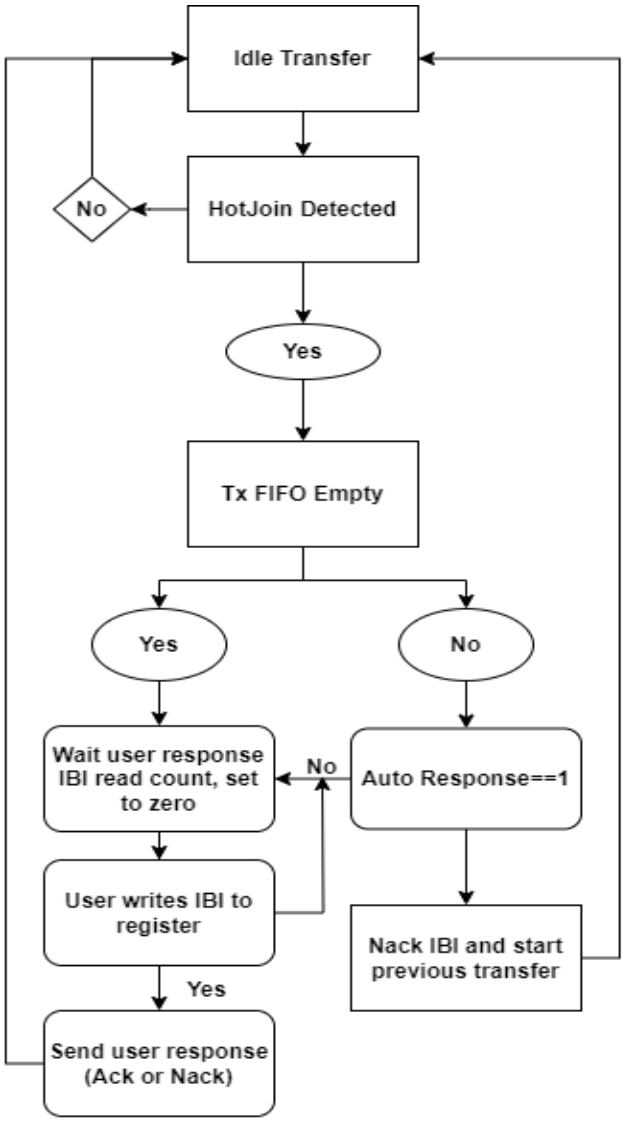

Figure 3.13. i3c_controller_hj_handling

3.14. i3c_controller_Contrlr_handoff flow

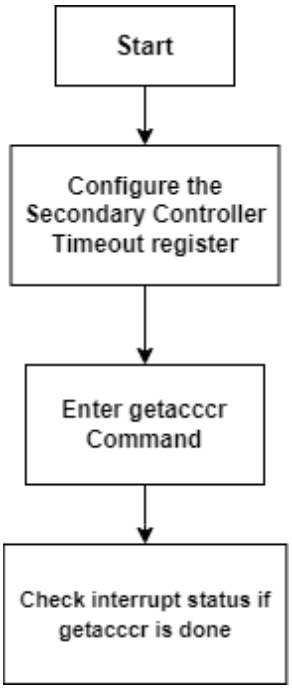

Figure 3.14. i3c_controller_Contrlr_handoff

3.15. i3c_controller_Consecutive_private_i3c_write flow

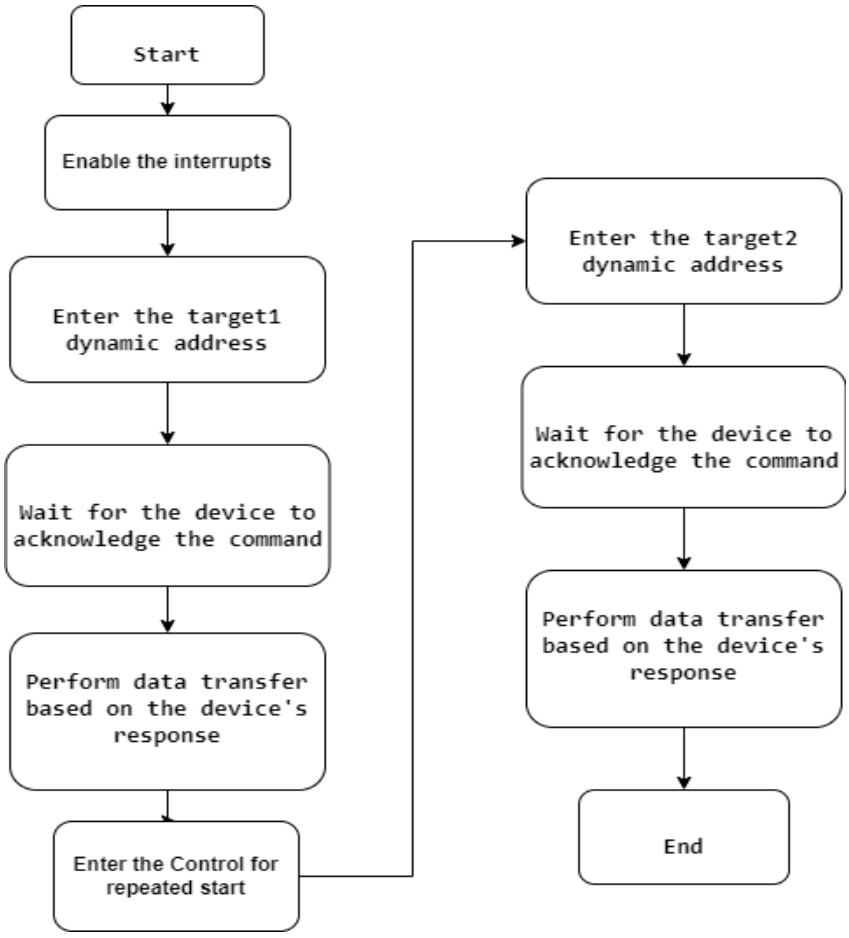

Figure 3.15. i3c_controller_Consecutive_private_i3c_write

3.16. i3c_controller_Consecutive_private_i3c_write_read flow

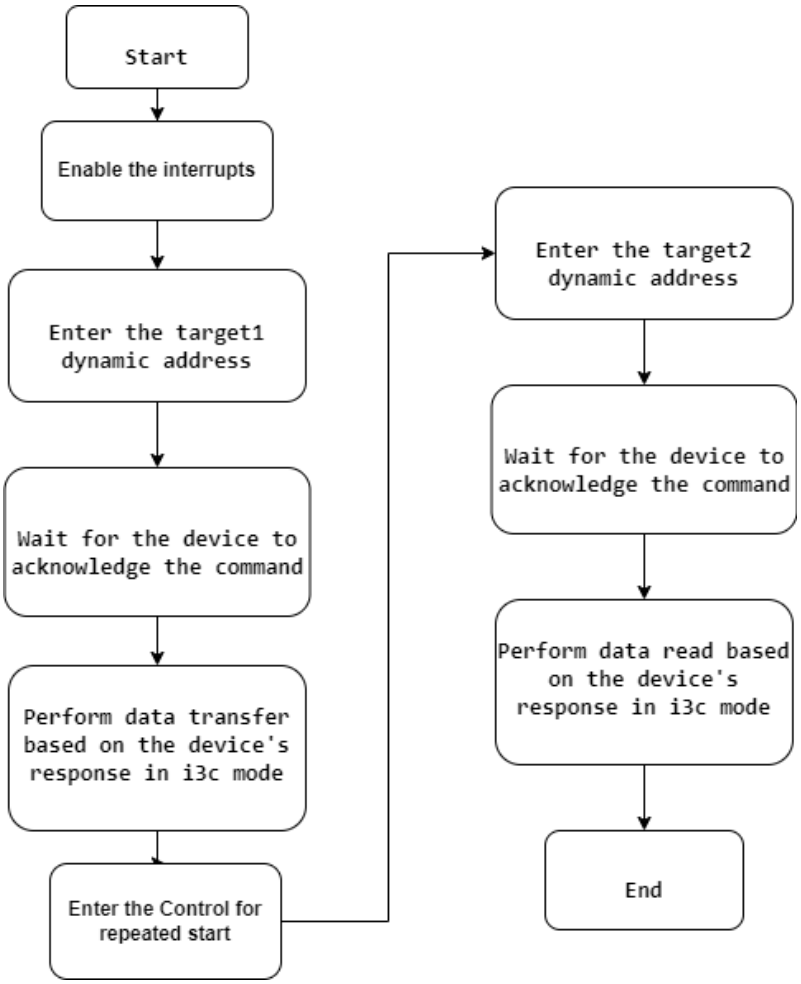

Figure 3.16. i3c_controller_Consecutive_private_i3c_write_read

3.17. i3c_controller_Consecutive_private_i2c_write_read flow

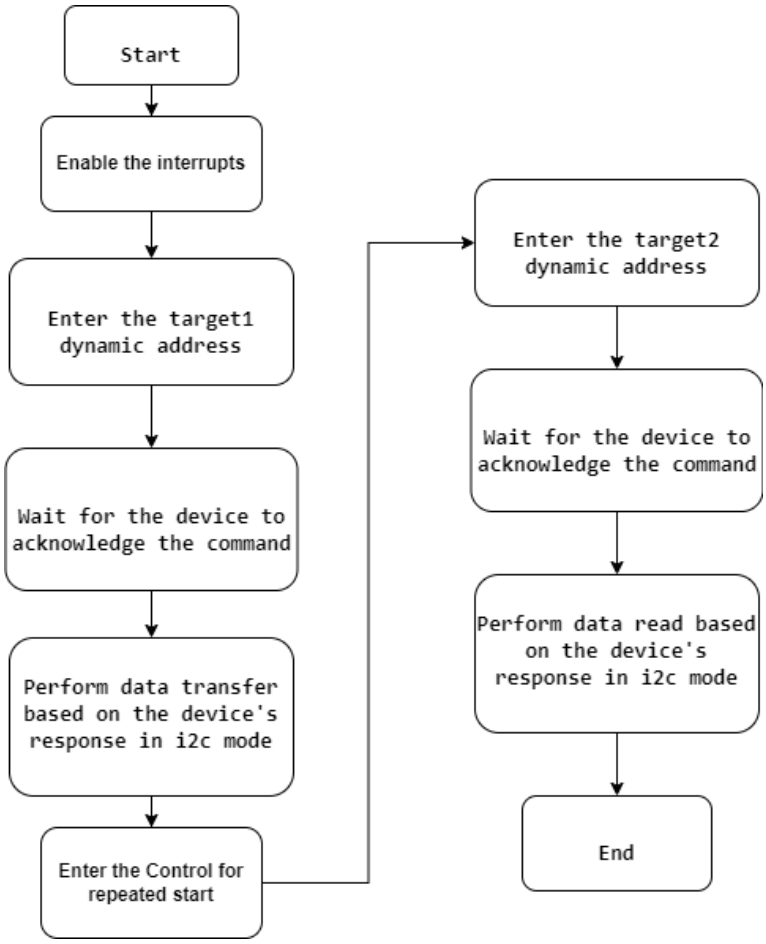

Figure 3.17. i3c_controller_Consecutive_private_i2c_write_read

3.18. i3c_controller_Consecutive_hdr_ddr_write_read flow

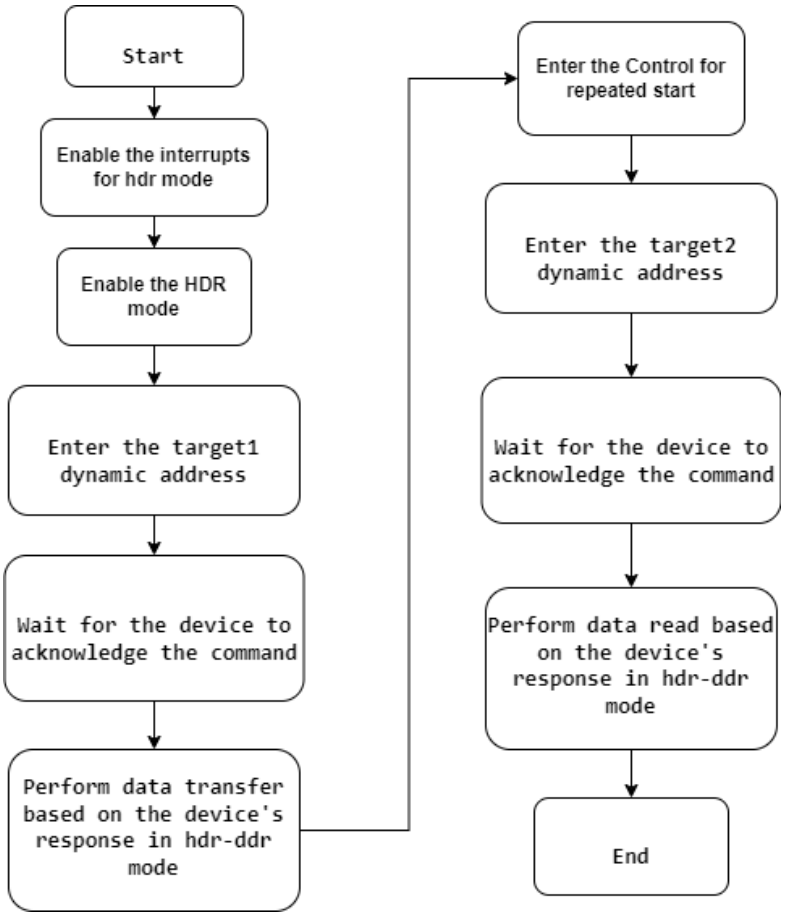

Figure 3.18. i3c_controller_Consecutive_private_hdr_ddr_write_read

3.19. i3c_controller_hdr_broadcast_ccc flow

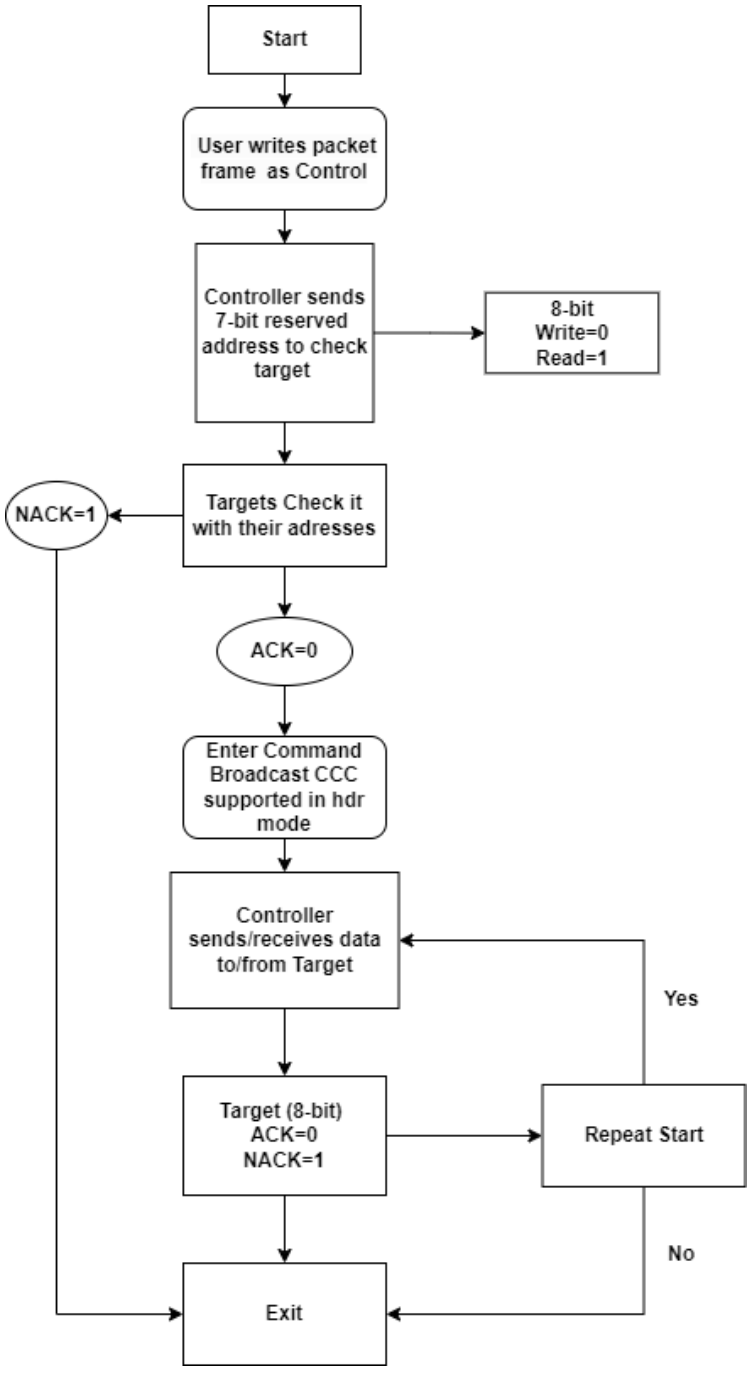

Figure 3.19. i3c_controller_hdr_broadcast_ccc

3.20. i3c_controller_hdr_direct_ccc flow

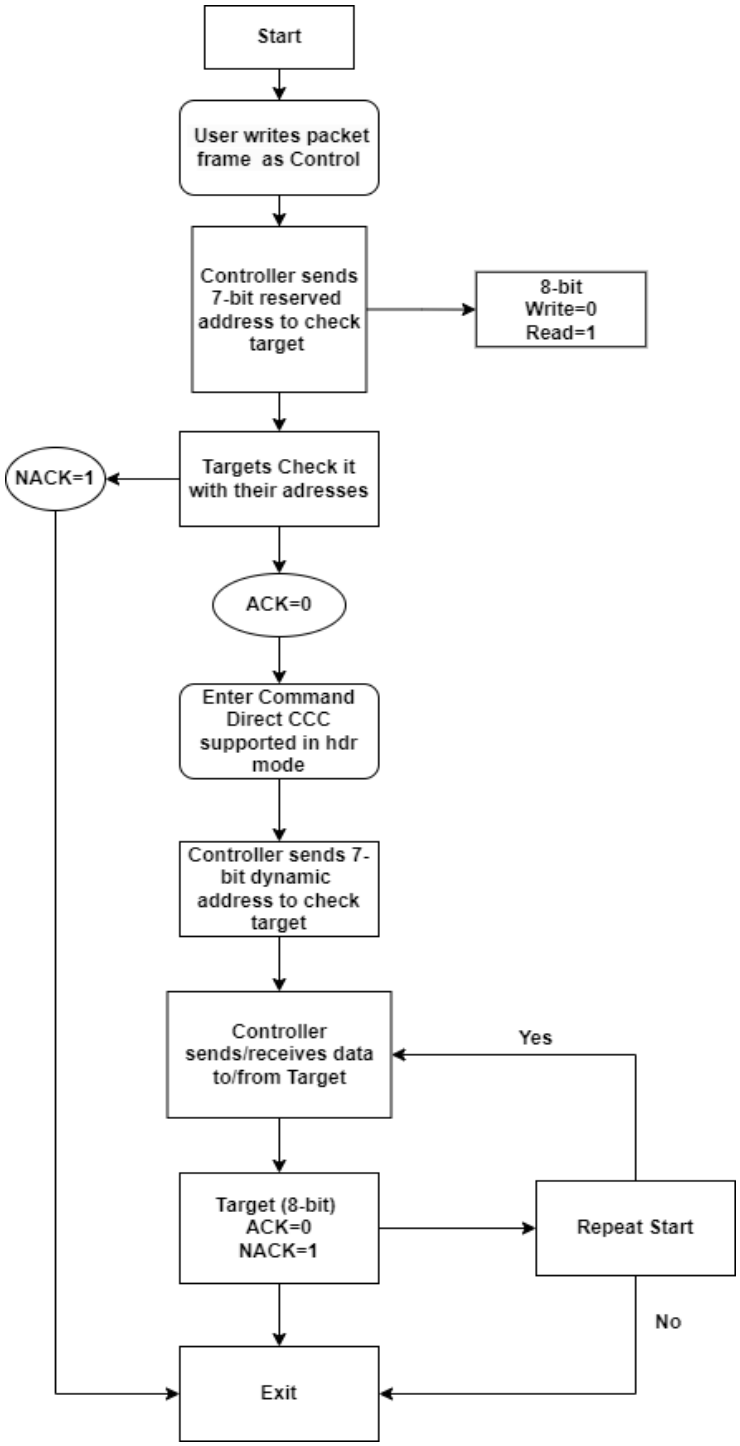

Figure 3.20. i3c_controller_hdr_direct_ccc

4. API Data Structures

This section describes the structures used in the API.

4.1. i3c_cntler_reg_t (32-bit)

This structure is used to assign offset for the I3C controller register. The following register offset is used in this structure. It is in 32-bit used in the DWORD interface.

```
volatile unsigned int i3c_controller_clock_divider;  //0x01
volatile unsigned int i3c_controller_config0;
volatile unsigned int i3c_controller_open_drain;
volatile unsigned int i3c_controller_i2c_clock_divider;
volatile unsigned int pull_up_resistor_disable;     //0x05
volatile unsigned int hdr_ddr_config0;
volatile unsigned int soft_reset_register;
volatile unsigned int reserved1[9];                 //reserved 0x08 to 0x10
volatile unsigned int i3c_controller_start_reg;
volatile unsigned int reserved2[3];                 //reserved 0x12 to 0x14
volatile unsigned int secondary_contr_status;
volatile unsigned int set_device_role;
volatile unsigned int secondary_contr_timeout_17;   //0x17
volatile unsigned int secondary_contr_timeout_18;   //0x18
volatile unsigned int secondary_contr_timeout_19;   //0x19
volatile unsigned int reserved3[2];                 //reserved 0x1a to 0x1b
volatile unsigned int i3c_controller_da_ack_reg;
volatile unsigned int ibi_read_count_register;
volatile unsigned int ibi_response_register;
volatile unsigned int ibi_address_register;         //0x1f
volatile unsigned int i3c_controller_intrpt_status0;
volatile unsigned int i3c_controller_intrpt_set0;
volatile unsigned int i3c_controller_intrpt_enable0;
volatile unsigned int reserved4;                     //reserved 0x23
volatile unsigned int i3c_controller_intrpt_status1;
volatile unsigned int i3c_controller_intrpt_set1;
volatile unsigned int i3c_controller_intrpt_enable1;
volatile unsigned int reserved5;                     //reserved 0x27
volatile unsigned int bus_cond_debug;
volatile unsigned int last_nack_addr;
volatile unsigned int last_ack_addr;
volatile unsigned int reserved6;                     //reserved 0x2b
volatile unsigned int i3c_controller_intrpt_status2;
volatile unsigned int i3c_controller_intrpt_set2;
volatile unsigned int i3c_controller_intrpt_enable2;
volatile unsigned int reserved7;                     //reserved 0x2f
volatile unsigned int i3c_controller_tx_fifo;        // 0x30
volatile unsigned int reserved[15];                  //reserved 0x31 to 0x3f
volatile unsigned int i3c_controller_rx_fifo;
```

4.2. i3c_cntler_reg_t (8-bit)

This structure is used to assign offset for the I3C controller register. The following register offset is used in this structure. It is in 8-bit used in the APB interface.

```
volatile unsigned char reserved0;                       //0x00
volatile unsigned char i3c_controller_clock_divider;  //0x01
volatile unsigned char i3c_controller_config0;
volatile unsigned char i3c_controller_open_drain;
volatile unsigned char i3c_controller_i2c_clock_divider;
volatile unsigned char pull_up_resistor_disable;     //0x05
volatile unsigned char hdr_ddr_config0;
volatile unsigned char soft_reset_register;
volatile unsigned char reserved1[9];                  //reserved 0x08 to 0x10
volatile unsigned char i3c_controller_start_reg;
volatile unsigned char reserved2[3];                  //reserved 0x12 to 0x14
volatile unsigned char secondary_contr_status;
volatile unsigned char set_device_role;
volatile unsigned char secondary_contr_timeout_17;   //0x17
volatile unsigned char secondary_contr_timeout_18;   //0x18
volatile unsigned char secondary_contr_timeout_19;   //0x19
volatile unsigned char reserved3[2];                  //reserved 0x1a to 0x1b
volatile unsigned char i3c_controller_da_ack_reg;
volatile unsigned char ibi_read_count_register;
volatile unsigned char ibi_response_register;
volatile unsigned char ibi_address_register;         //0x1f
volatile unsigned char i3c_controller_intrpt_status0;
volatile unsigned char i3c_controller_intrpt_set0;
volatile unsigned char i3c_controller_intrpt_enable0;
volatile unsigned char reserved4;                       //reserved 0x23
volatile unsigned char i3c_controller_intrpt_status1;
volatile unsigned char i3c_controller_intrpt_set1;
volatile unsigned char i3c_controller_intrpt_enable1;
volatile unsigned char reserved5;                       //reserved 0x27
volatile unsigned char bus_cond_debug;
volatile unsigned char last_nack_addr;
volatile unsigned char last_ack_addr;
volatile unsigned char reserved6;                       //reserved 0x2b
volatile unsigned char i3c_controller_intrpt_status2;
volatile unsigned char i3c_controller_intrpt_set2;
volatile unsigned char i3c_controller_intrpt_enable2;
volatile unsigned char reserved7;                       //reserved 0x2f
volatile unsigned char i3c_controller_tx_fifo;        // 0x30
volatile unsigned char reserved[15];                  //reserved 0x31 to 0x3f
volatile unsigned char i3c_controller_rx_fifo;
```

4.3. struct mm2s_desc_tx_t

This structure is used to declare different types of parameters which are used for the I3C controller API. Table 4.1 shows the available parameters and description.

```
bool mode;
unsigned int len;
unsigned int *buf;
unsigned int scl_clk_divider;
unsigned int base_addr;
unsigned int target_dyn_addr;
unsigned int target_static_addr;
```

```
unsigned int target_nums;
unsigned int direct_ccc_optnal_bytes;
unsigned int *direct_ccc_optnal_data_array;
unsigned int target2_dyn_addr;
unsigned int len2;
unsigned int *buf2;
unsigned int ibi_read_count;
unsigned int *ibi_payload_data;
unsigned int user_7bit_code;
unsigned int target_dyn_addr_multi[];
```

Table 4.1. struct mm2s_desc_tx_t Parameters

Parameter	Description
mode	To assign I3C or I2C mode.
len	To assign the length to write or read for each API.
buf	To write or read data for each API.
scl_clk_divider	To check the frequency assigned for both modes.
base_addr	Base address of the target.
target_dyn_addr	To assign dynamic addresses to the targets.
target_static_addr	To assign static addresses to the targets in I2C mode.
target_nums	Number of targets.
direct_ccc_optnal_bytes	To assign the length of optional bytes to write or read in the register.
direct_ccc_optnal_data_array	To assign optional bytes data to write or read.
target2_dyn_add	To assign dynamic address to target 2.
len2	To assign the length to write or read for target 2.
buf2	To write or read data for target 2.
ibi_read_count	To assign the length of IBI payload.
ibi_payload_data	To assign IBI payload data.
user_7bit_code	To assign write or read status.
target_dyn_addr_multi	To assign dynamic addresses to multitargets.

4.4. i3c_dev_info_t

This structure is used to store the PID, DCR, and BCR of a target which is assigned during the dynamic address assignment. Table 4.2 shows the available parameters and description.

```
unsigned int init_dyn_addr;
unsigned int bcr;
unsigned int dcr;
unsigned int pid;
```

Table 4.2. i3c_dev_info_t Parameters

Parameter	Description
init_dyn_addr	To store the dynamic address for each target
pid	To store the provisional id for each target.
dcr	To store the device-characteristics register for each target.
bcr	To store the bus-characteristics register for each target.

5. API Variables

Table 5.1 shows the variables and their description.

Table 5.1. Variable Description

Variable	Description
handle	A structure variable that consists of base address, buffer, and length.
control	Gives packet frame values to APIs.
status	Gives return types for all APIs.
index	The increment values in for loop.

6. API Macros

6.1. Reg 32-bit and 8-bit (Set by the user)

```
#define REG_32_BIT                  1
#define REG_8_BIT                   0
```

6.2. Set and Clear

```
#define SET                         0x01
#define CLEAR                       0x00
#define ALL_BITS_EN                 0xff
```

6.3. Success and Failure

```
#define FAILURE                      0
#define SUCCESS                     1
```

6.4. Write and Read

```
#define WRITE                       0
#define READ                        1
```

6.5. User Packet Frame

```
#define CCC_CMD1_WR_RD0             0x01
#define REPEATED_START_IN_FRAME     0x02
#define STOP_IN_FRAME               0X04
#define CCC_COMMAND_WITH_START      0X08
#define I2C_FRAME                   0x10
#define WAIT_NEXT_PACKET            0X20
#define HDR_MODE                    0X80
```

6.6. CCC Commands

6.6.1. Broadcast CCC

```
#define ENEC_B                      0x00        //Broadcast R
#define DISEC_B                     0x01        //Broadcast R
#define ENTAS0_B                    0x02        //Broadcast C
#define ENTAS1_B                    0x03        //Broadcast O
#define ENTAS2_B                    0x04        //Broadcast O
#define ENTAS3_B                    0x05        //Broadcast O
#define RSTDAA                      0x06        //Broadcast R
#define ENTDAA                      0x07        //Broadcast R
#define DEFTGTS                     0x08        //Broadcast C
#define SETMWL_B                    0x09        //Broadcast R
#define SETMRL_B                    0x0A        //Broadcast R
#define ENDXFER_B                   0x12        //Broadcast C
#define ENTHDR0                     0x20        //Broadcast C
#define SETXTIME                    0x28        //Broadcast C
#define SETAASA                     0x29        //Broadcast O
```

```
#define RSTACT                          0x2A        //Broadcast R
```

6.6.2. Direct CCC

```
#define ENEC_D                          0x80        //Direct R
#define DISEC_D                         0x81        //Direct R
#define ENTAS0_D                        0x82        //Direct C
#define ENTAS1_D                        0x83        //Direct O
#define ENTAS2_D                        0x84        //Direct O
#define ENTAS3_D                        0x85        //Direct O
#define SETDASA                         0x87        //Direct O
#define SETNEWDA                        0x88        //Direct C
#define SETMWL_D                        0x89        //Direct R
#define SETMRL_D                        0x8A        //Direct R
#define GETMWL                          0x8B        //Direct R
#define GETMRL                          0x8C        //Direct R
#define GETPID                          0x8D        //Direct C
#define GETBCR                          0x8E        //Direct C
#define GETDCR                          0x8F        //Direct C
#define GETSTATUS                       0x90        //Direct R
#define GETACCCR                        0x91        //Direct C
#define ENDXFER_D                       0x92        //Direct C
```

6.7. Hot Join and IBI

```
#define IBI_AUTO_RESP                   0x20
#define HOT_JOIN_INTRPT_RCVD            0X08
#define IBI_AUTO_RESP_LOW               0xdf
#define IBI_INTRPT_RCVD                 0X10
#define WAITING_IBI_RESP                0x40
#define IBI_RD_DONE_EN                  0X04
```

6.8. Secondary Controller Handoff

```
#define CRH_TIMEOUT_VALUE               0x009C4
#define CRH_TIMEOUT_VALUE_REG17         0x000ff
#define CRH_TIMEOUT_VALUE_REG18         0x0ff00
#define CRH_TIMEOUT_VALUE_REG19         0xf0000
#define GET_ACCR_DONE                   0x08
#define GET_ACCR_RESULT                 0x04
#define SEC_CTRRL_INTRPT_RCVD           0X20
```

6.9. Interrupts

```
#define EN_DAA_UID_IN_RXFIFO            0x02
#define IGNORE_RCVD_NAK                 0x04
#define IGNORE_CMD_DONE                 0x10
#define RCVD_SLAVE_NACK                 0x80
#define RCVD_CMND_DONE                  0x40
#define TX_FIFO_FULL                    0X04
#define RX_FIFO_n_EMPTY                 0x02
#define RD_CMD_DONE                     0x01
#define DDR_IGNORE_CMD_DONE             0x08
```

```
#define NO_CRC_AFTER_TERM          0X04
#define EN_WR_EARLY_TERM           0x02
#define EN_WRCMD_ACKNAK_CAP        0X01
#define I2C_MODE_ALLOWED           0x28
#define I3C_MODE_ALLOWED           0x20
#define HDR_RD_CMD_DONE            0x01
#define HDR_SLAVE_NACK             0x80
#define HDR_CMND_DONE              0x40
```

6.10. Shift Values

```
#define READ_LEFT_SHIFT            0x01
#define READ_RIGHT_SHIFT           0X80
#define PULL_UP_RES_DISABLE        0X09
#define EIGHT_LEFT_SHIFT           8
#define FIVE_SHIFT                 5
```

6.11. Values

```
#define RESERVED_ADDRESS           0x7e
#define I3C_MODE                   1
#define MAX_ADRR_SIZE              127
#define MAX_FIFO_SIZE              512
```

References

- I3C Controller IP Core User Guide (FPGA-IPUG-02228)
- MachXO2 web page
- MachXO3D web page
- MachXO3 web page
- CrossLink-NX web page
- Certus-NX web page
- CertusPro-NX web page
- Mach-NX web page
- MachXO5-NX web page
- Avant-E web page
- Avant-G web page
- Avant-X web page
- Lattice Radiant Software 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, December 2023

Section	Change Summary
All	Initial release.

www.latticesemi.com