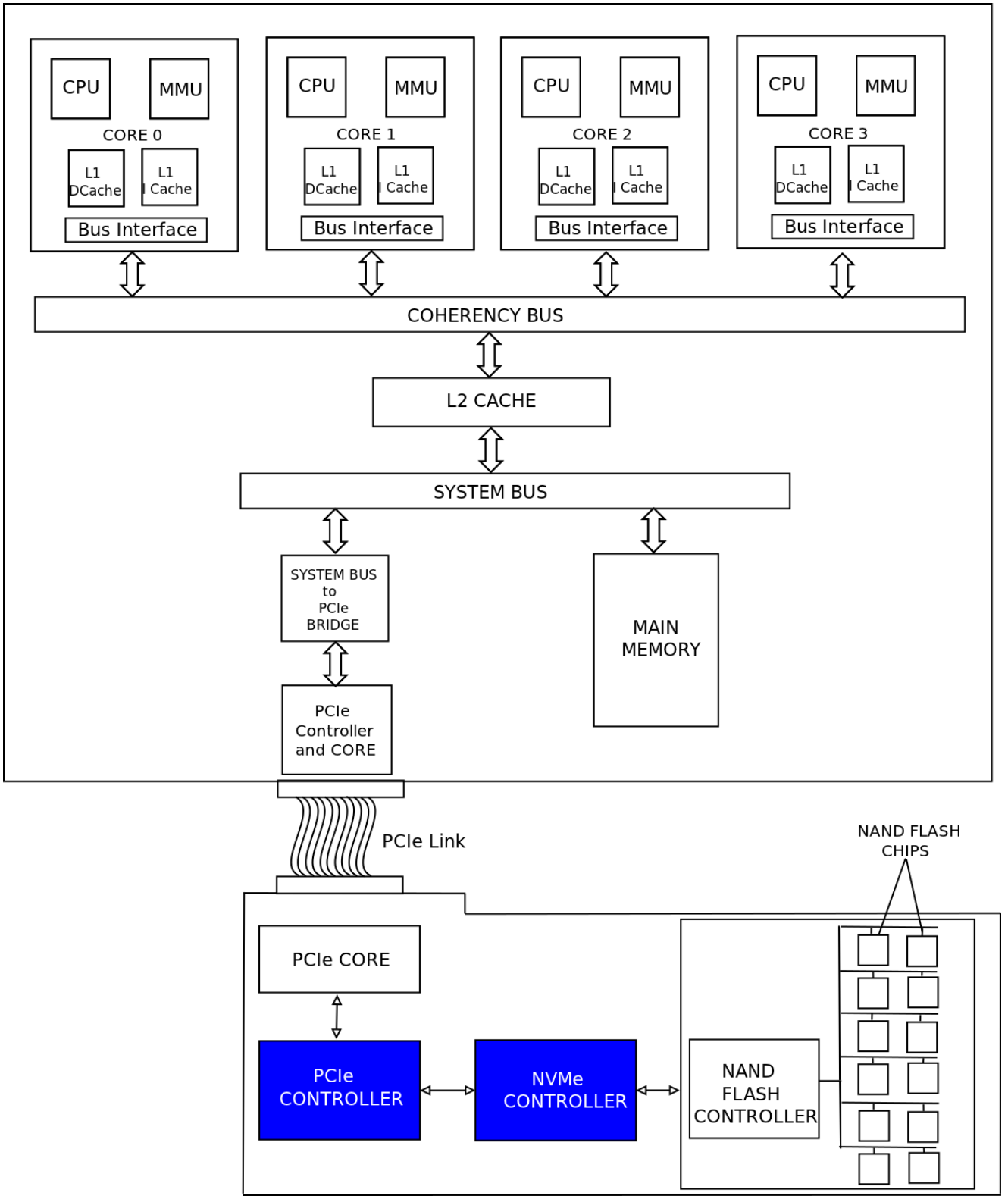
Design and Implementation of NVM Express

M SHANMUKH ABHISHEK
EE11M057
DEPARTMENT OF ELECTRICAL ENGINEERING
IIT MADRAS

CONTENTS

OVERALL ARCHITECTURE AND MY CONTRIBUTION


CPU
MMU
CORE 0
L1 DCache
L1 I Cache
Bus Interface
CPU
MMU
CORE 1
L1 DCache
L1 I Cache
Bus Interface
CPU
MMU
CORE 2
L1 DCache
L1 I Cache
Bus Interface
CPU
MMU
CORE 3
L1 DCache
L1 I Cache
Bus Interface
COHERENCY BUS
L2 CACHE
SYSTEM BUS
SYSTEM BUS to PCIe BRIDGE
MAIN MEMORY
PCIe Controller and CORE
PCIe Link
NAND FLASH CHIPS
PCIe CORE
PCIe CONTROLLER
NVMe CONTROLLER
NAND FLASH CONTROLLER

INTRODUCTION TO BLUESPEC SYSTEM VERILOG (BSV)

BSV Language Background

- Hardware centric language based on industry standard SystemVerilog and Verilog.
- Uses *rules* and *interface methods* for behavioral description.
- Adds a powerful way to express complex concurrency and control :
  - Across multiple shared resources
  - Across module boundaries

How is Parallelization Achieved ?

- Concurrent behavior is expressed implicitly.
- Traditional hardware semantic model of cooperating FSMs.
- Rules express concurrent operations by simply describing the conditions under which the state element(s) are updated.
- Rules are unsequenced atomic transactions.
- Complier introduces the scheduling and the muxing for the shared resources.

Level of Abstraction

- BSV provides a significantly higher level of abstraction than Verilog, SystemVerilog, VHDL and SystemC.
- Behavioral description through :
  - Rules and interface methods
- Structural description :
  - High level abstraction types
  - Powerful static checking
  - Powerful parameterization
  - Powerful static elaboration

Standalone Function Libraries

- BSV has a large set of functional libraries.
- Data containers such as FIFO , registers , BRAMs etc .
- Circuits such as LFSR and completion buffer.
- Interface types , GET, PUT transactors.
- Multiple clockdomain Synchronizers .
- Various bus interfaces such as Common data bus , Z-bus etc.

Simulation and Debugging Flows

- Design is simulated and debugged using source level simulation tool “*Bluesim*”.
- It can also be simulated by using the generated Verilog RTL.
- Simulation tools such as QuestaSim can directly be “*linked*” to Bluesim to debug the designs.
- Full visibility to Bluespec interfaces and state elements.
- Also generates standard VCD files.

NEED FOR NVM Express

Why NVMe ?

- Standardized specification to access SSDs.
- Promising future of SSDs.
- PCIe SSDs with SATA controller has following problems :
  - PCIe to SATA interface conversion overhead.
  - SATA Command (HDD based) to NAND Flash Command (SDD based) conversion overhead.
- PCIe SSDs with NVMe has no overhead.

**SATA – Serial Advanced Technology Attachment*

Why NVMe ?

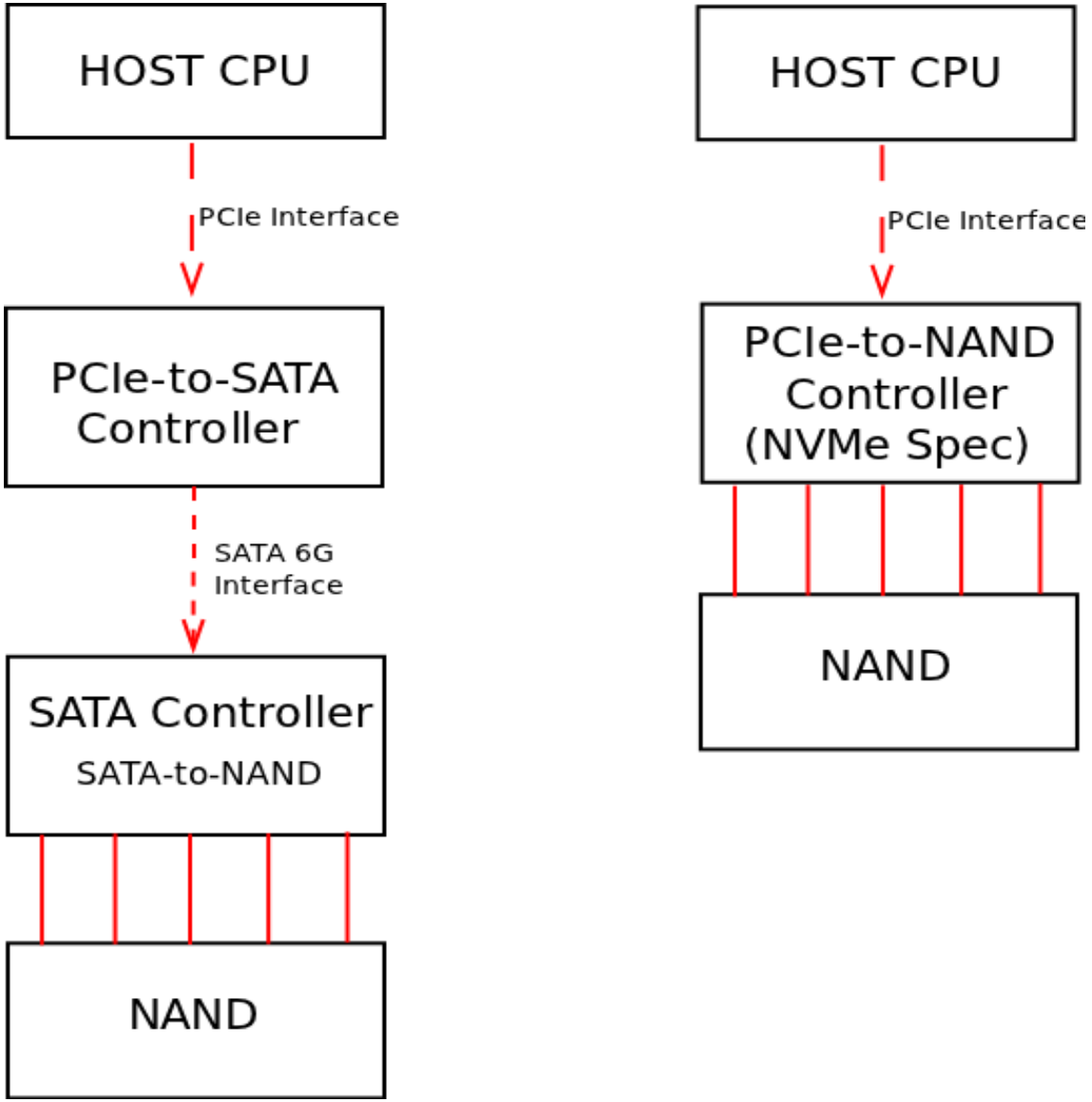


Figure: Illustration of interface overhead invloved in SATA Controller based SSD

NVMe Features in Comparision With AHCI

Feature	AHCI	NVMe
Max Q Depth	1 Command Q 32 Commands per Q	64k Queues 64k Commands per Q
Un-cacheable register access	6 per non-queued cmd 9 per queued comand	2 per command
MSI Inetrrupts	Single Interrupt	2K MSI-X Interrupts or 32 MSI Interupts
Parallelism & Multiple threads	Requires sync lock to issue command	No locking
Efficiency for 4KB Commands	Cmd perameters require 2 host DRAM fetches	Command parameters in one 64B fetch

Table : Comparision of Features of AHCI and NVMe

**AHCI – Advanced Host Controller Interface*

NVMe SUBSYSTEM

NVMe SUB-SYSTEM

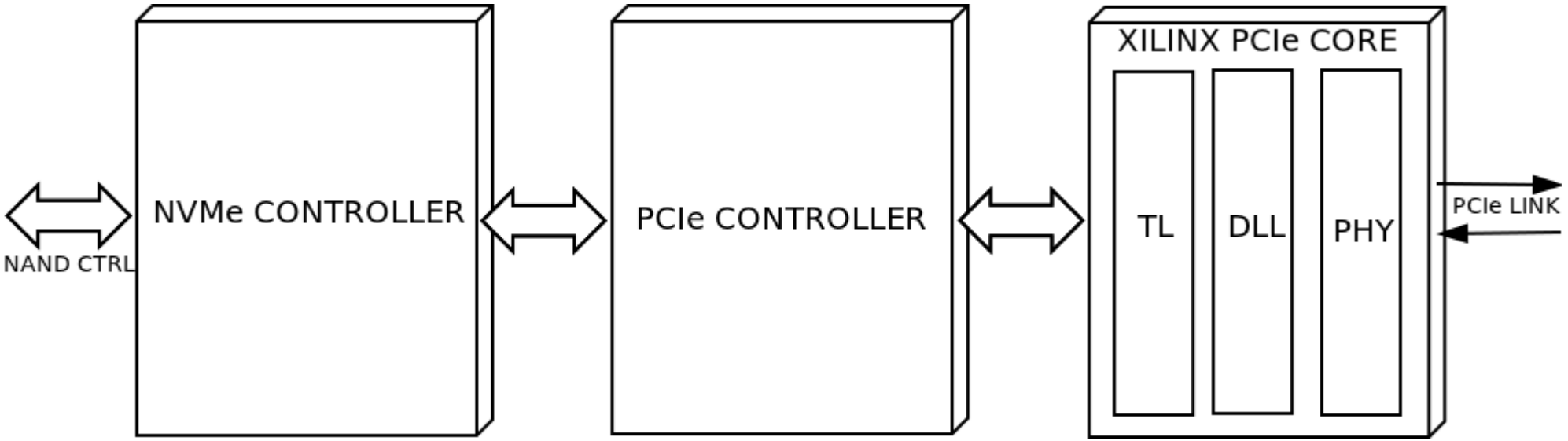


Figure 1 : NVMe Subsytem showing NVMe Controller, PCIe Controller and PCIe Core

PCI Express Controller Implementation

PCIe IMPLEMENTATION

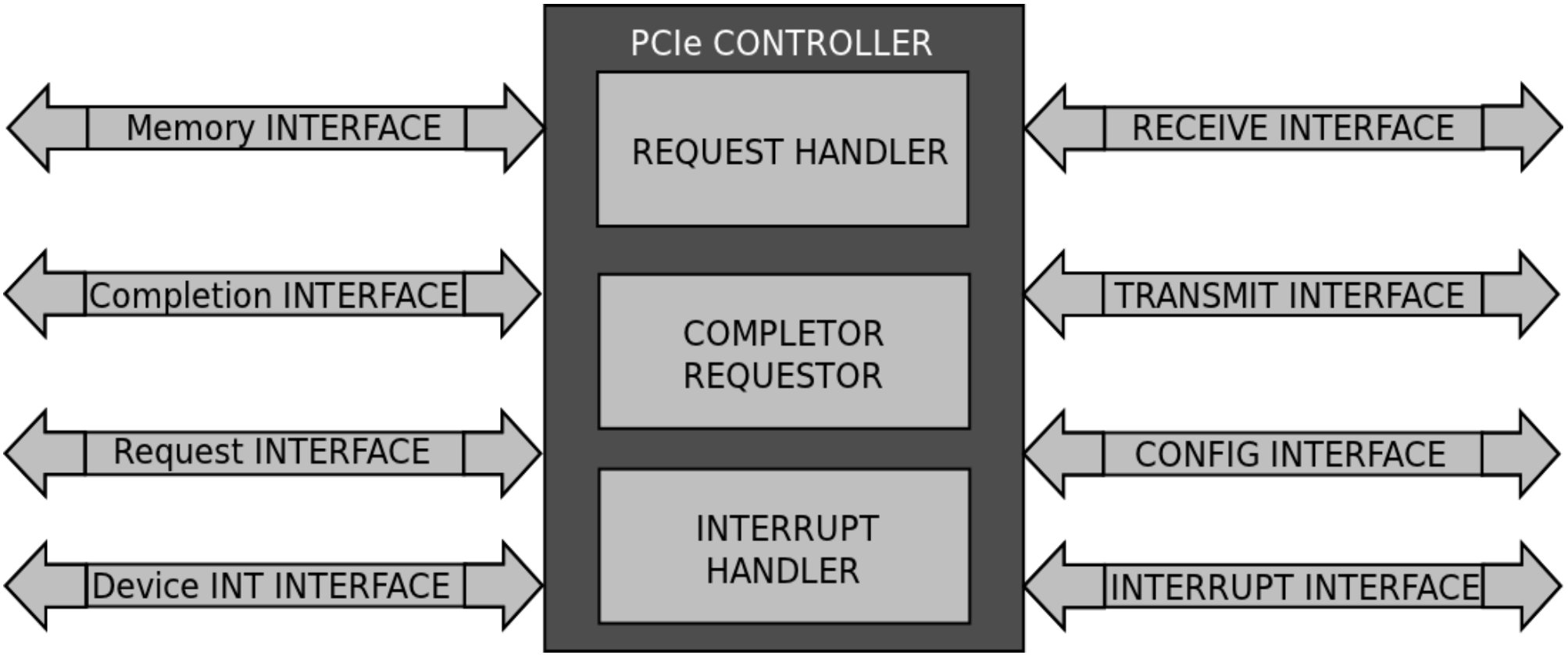


Figure 1 : PCIe Modules and Interfaces

INTERFACE DESCRIPTION

INTERFACE DESCRIPTION

Receive Interface Signals

Name	Direction	Description
wr_rx_tlast	Input	Signals End Of Packet
wr_rx_tdata[31:0]	Input	Packet data being received
wr_rerr_fwd	Input	Marks packet in progress as error poisoned
wr_rx_tvalid	Input	Indicates packet data is valid
rg_rx_tready	Output	Indicates user application is ready to receive data
wr_bar_hit[6:0]	Input	Indicates BAR(s) targetted by current active transaction

Table : A brief description of some of the signals present in the Receive Interface

**The interface definition is present in InterfaceRequestHandler.bsv*

INTERFACE DESCRIPTION

Transmit Interface Signals

Name	Direction	Description
rg_tx_tdata[31:0]	Output	Packet data to be transmitted
rg_src_dsc	Output	Tansmit Source discontinue
wr_buf_av[5:0]	Input	Indicates the Number of Transmit Buffers Available for use.
wr_cfg_req	Input	Tansmit Configuration Request. Indiactes the request generated by core to send config TLPs
rg_cfg_gnt	Output	Tansmit Configuration Grant. Indiactes grant to send config TLPs.

Table : A brief description of some of the signals present in the Transmit Interface

**The interface definition is present in InterfaceCompleterRequester.bsv*

INTERFACE DESCRIPTION

Configuration Interface Signals

Name	Direction	Description
wr_cfg_bus_number[7:0]	Input	Provides assigned bus number for the device. Used in bus number field of outgoing TLPs
wr_cfg_device_number[4:0]	Input	Provides assigned device number for the device. Used in device number field of outgoing TLPs
wr_cfg_function_number[2:0]	Input	Provides assigned function number for the device. Used in function number field of outgoing TLPs
wr_cfg_status[15:0]	Input	Status Register

Table : A brief description of some of the signals present in the Config Interface

**The interface definition is present in InterfaceCompleterRequester.bsv*

INTERFACE DESCRIPTION

Interrupt Interface Signals

Name	Direction	Description
rg_cfg_interrupt	Output	Interrupt request signal to Core
wr_cfg_interrupt_gnt	Input	Interrupt grant signal asserted by Core
rg_cfg_interrupt_di[7:0]	Output	Indicates the MSI vector number
wr_cfg_interrupt_mmenable[2:0]	Input	Multiple Message Enable Signal. This signal is used as the core generated threshold for multiple vector MSI
wr_cfg_interrupt_msienable	Input	Indiactes that the Message Signaling Interrupt (MSI) messaging is enabled

Table : A brief description of some of the signals present in the Interrupt Interface

**The interface definition is present in InterfaceInterruptHandler.bsv*

INTERFACE DESCRIPTION

Memory Interface Signals

Name	Direction	Description
rg_address_rf[31:0]	Output	Address of the register in memory
rg_data_out[63:0]	Output	Data to be written into memory
rg_byte_enable[3:0]	Output	'1111 indicates 64 bit register access '0011 indicates 32 bit register access
rg_read	Output	Memory Read operation
rg_write	Output	Memory Witre operation

Table : A brief description of some of the signals present in the Memory Interface

**The interface definition is present in InterfaceRequestHandler.bsv*

INTERFACE DESCRIPTION

Completions Interface Signals

Name	Direction	Description
rg_data_valid	Output	Indicates Data being written is valid
rg_data_out[31:0]	Output	Completion Data to be written to device
rg_last_DWord	Output	Indicates that this is the “Last Dword “ for this entire transaction . No more data will be received with the same tag.
rg_tag_to_device [6:0]	Output	Tag assigned for the transaction

Table : A brief description of some of the signals present in the Completion Interface

**The interface definition is present in InterfaceRequestHandler.bsv*

INTERFACE DESCRIPTION

Request Interface Signals

Name	Direction	Description
wr_data_in[31:0]	Input	Data to be sent as payload in the Write and Completion TLPs
wr_address_in[63:0]	Input	Address of the Memory location, where the TLPs are to be sent to.
wr_requested_tag[6:0]	Input	Tag for the transaction in read TLPs.
wr_send_completion_tlp	Input	Request for Completion TLP
wr_send_write_tlp	Input	Request for Write TLP
wr_send_read_tlp	Input	Request for Read TLP

Table : A brief description of some of the signals present in the Request Interface

**The interface definition is present in InterfaceCompleterRequester.bsv*

INTERFACE DESCRIPTION

Device Interrupt Interface Signals

Name	Direction	Description
wr_vector_rdy	Input	Intrrupt Vector Ready. Indicates that a new interrupt vector is ready to be taken.
wr_vector_number[4:0]	Input	Interrupt Vector Number.
wr_no_of_MSI_vectors [4:0]	Input	Device side request for the threshold limit of total number of interrupts to be queued before sending Interrupt Message

Table : Description of the signals present in the Device Interrupt Interface

**The interface definition is present in InterfaceInterruptHandler.bsv*

MODULES DESCRIPTION

MODULE DESCRIPTION

Request Handler

- This module receives various kinds of TLPs from the PCIe Core and sends the payload (if any) to the device(Register File or NVMe).
- TLPs include : Memory Write Request TLP, Memory Read Request TLP, Completion TLPs.
- Each TLP can have either 3 Dword Header or a 4 Dword Header depending on weather it is a 32 bit address or a 64 bit address.
- **File Name for code** : "*RequestHandler.bsv*"
- **Amount of code** : 1000 lines.
- **Verification** : The code has been verified for all the different types of TLPs . Some special test cases are highlighted here.

MODULE DESCRIPTION

TLP Decode Operation by the Request Handler

Fmt Field (data[30:29])	Typ Field (data[28:24])	Description	Action taken
2'b00	5'b00000	3 DWord Memory Read	1. Updates the Completer-Requester by supplying the Requester Tag value. 2. Sends read signal and read addrs through memory interface to device.
2'b01	5'b00000	4 Dword Memory Read	Same as 3DW Memory Read
2'b10	5'b00000	3 Dword Memory Write	1. Sends write signal to device memory. 2. Sets the byte enable to '1111 if it receives 2 Dword payload. Sets it to '0011 if it receives 1 DWord payload. 3. Puts the data on data bus.
2'b11	5'b00000	4 DWord Memory Write	Same as 3 Dword Memory write.
2'b00	5'b01010	Completion with No data	Not Applicable
2'b10	5'b01010	Completion with data	Sends the data payload on the Completion data-to-device interface

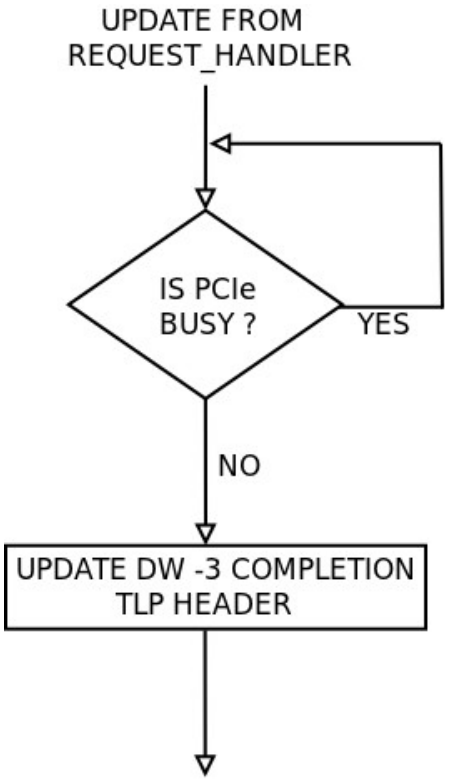
MODULE DESCRIPTION

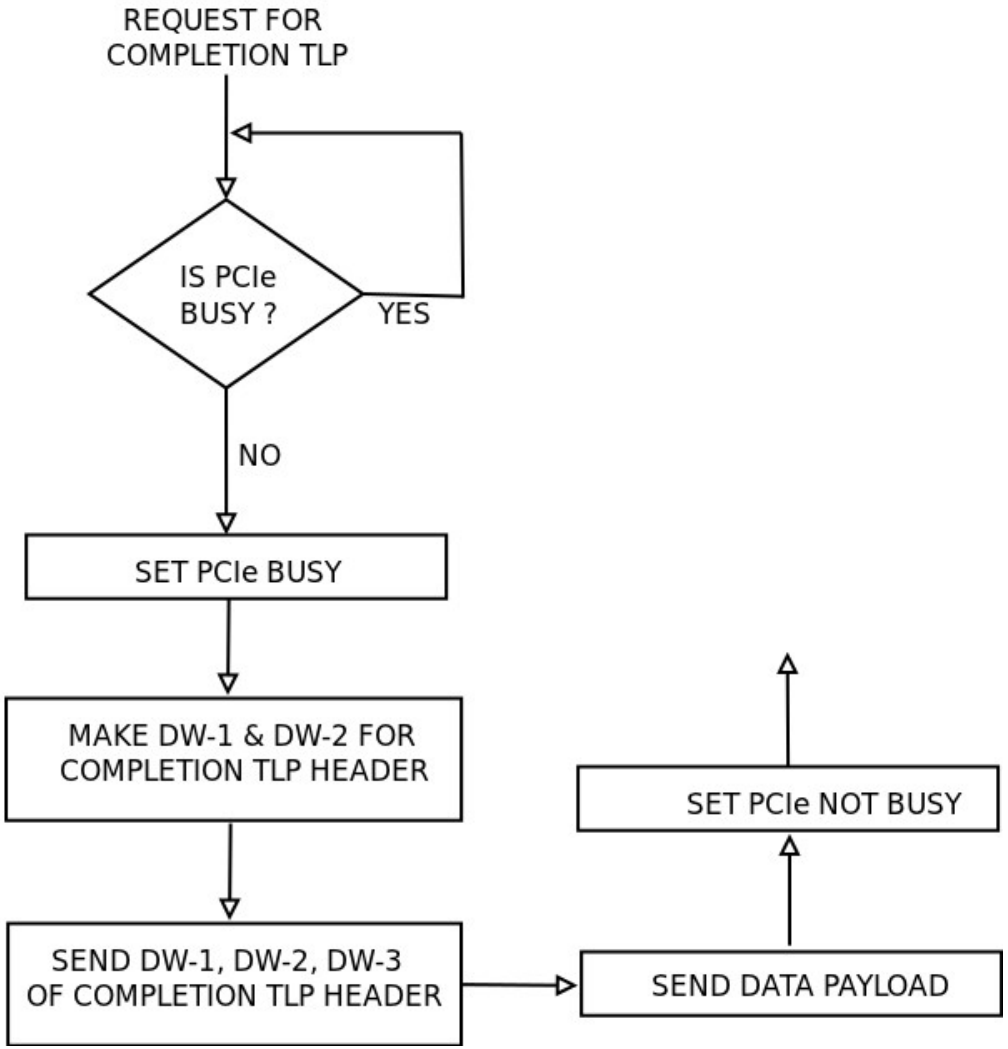
Completer Requester

- This module receives various kinds of requests from the device.
- The device requests are transformed into Read, Write or Completion TLPs.
- The data sent from the device (if any) is sent as data payload to the PCIe Core.
- **File Name for code** : "*CompleterRequester.bsv*"
- **Amount of code** : 1000 lines.
- **Verification** : The code has been verified for all the different types of TLPs . Some special test cases are highlighted here.

MODULE DESCRIPTION

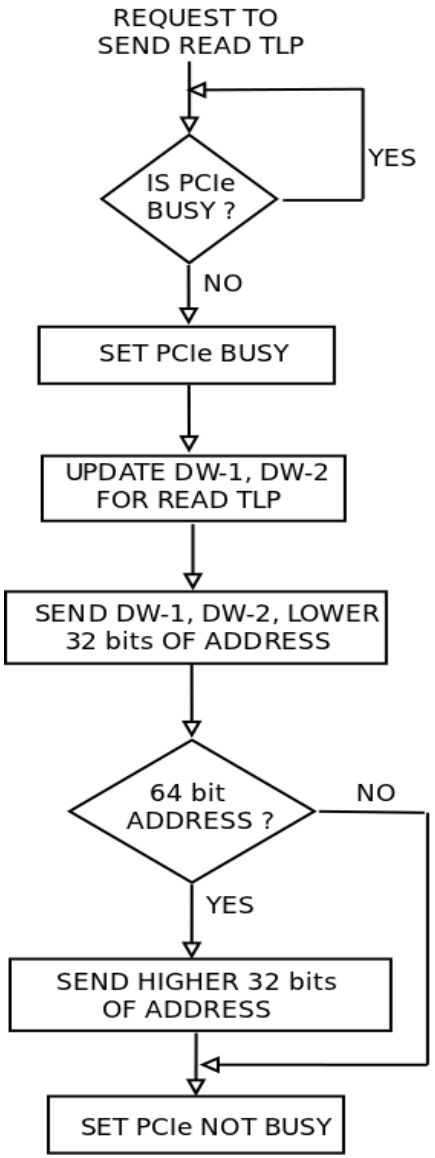
Completion TLP Request From Device

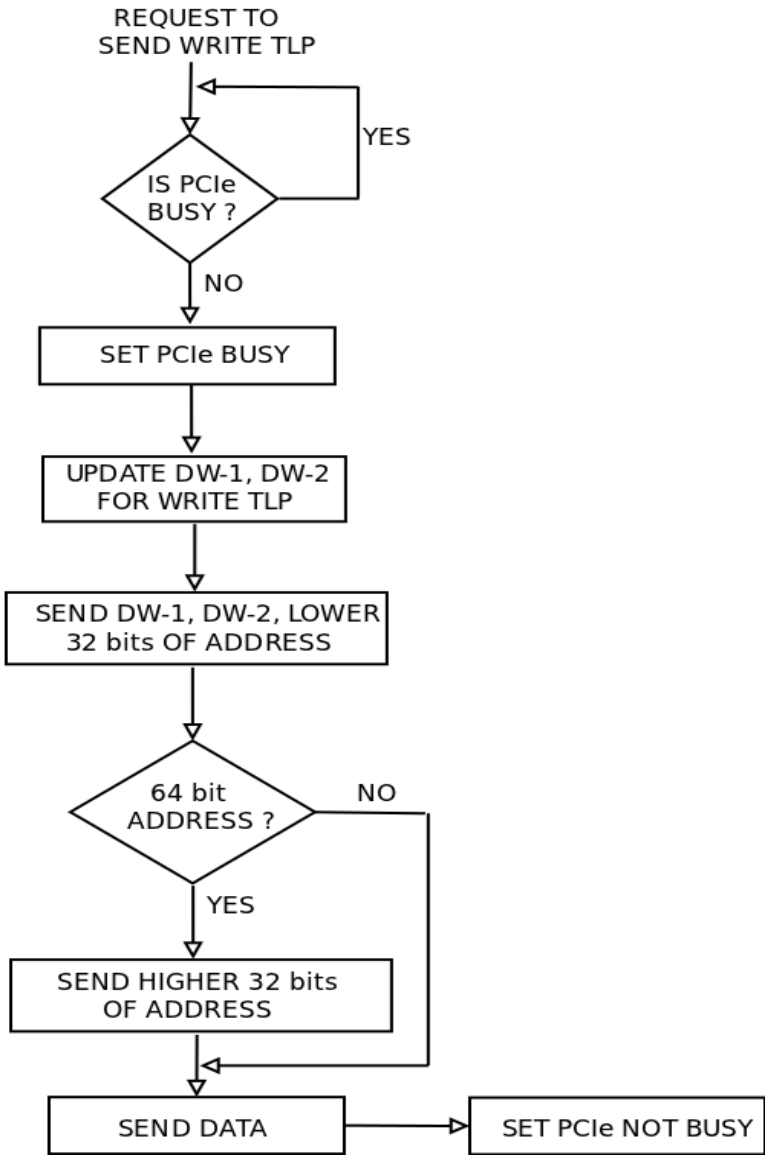




MODULE DESCRIPTION

Read-Write TLP Request From Device





MODULE DESCRIPTION

Interrupt Handler

- This module receives interrupt vector information from the device and is sent to PCIe core.
- It receives the “No. Of Interrupts” from the device and “Vector Threshold from the PCIe”. The minimum of the two is used to queue the interrupts and send to PCIe.
- Interrupt vectors are sent as Message TLPs to the PCIe
- **File Name for code** : “*InterruptHandler.bsv*”
- **Amount of code** : 500 lines.
- **Verification** : The code has been verified for single MSI support and Multiple MSI support.

VERIFICATION

VERIFICATION TEST CASES

Some of the Verification test cases are mentioned below

- Transmit Read, Write, Completion TLPs.
- Receive Read, Write, Completion TLPs.
- Transmit and receive under normal conditons.
- Transmit and Receive with 'destination throttling' , 'source throttling' .
- Transaction 'discontinue' .
- Receive 'Incomplete payload' .
- Transmit Interrupt Vectors as single Message Signalled Interrupt (MSI) and as multiple MSI .

SIMULATION RESULTS

SIMULATION RESULTS

Normal Write TLP Transmit

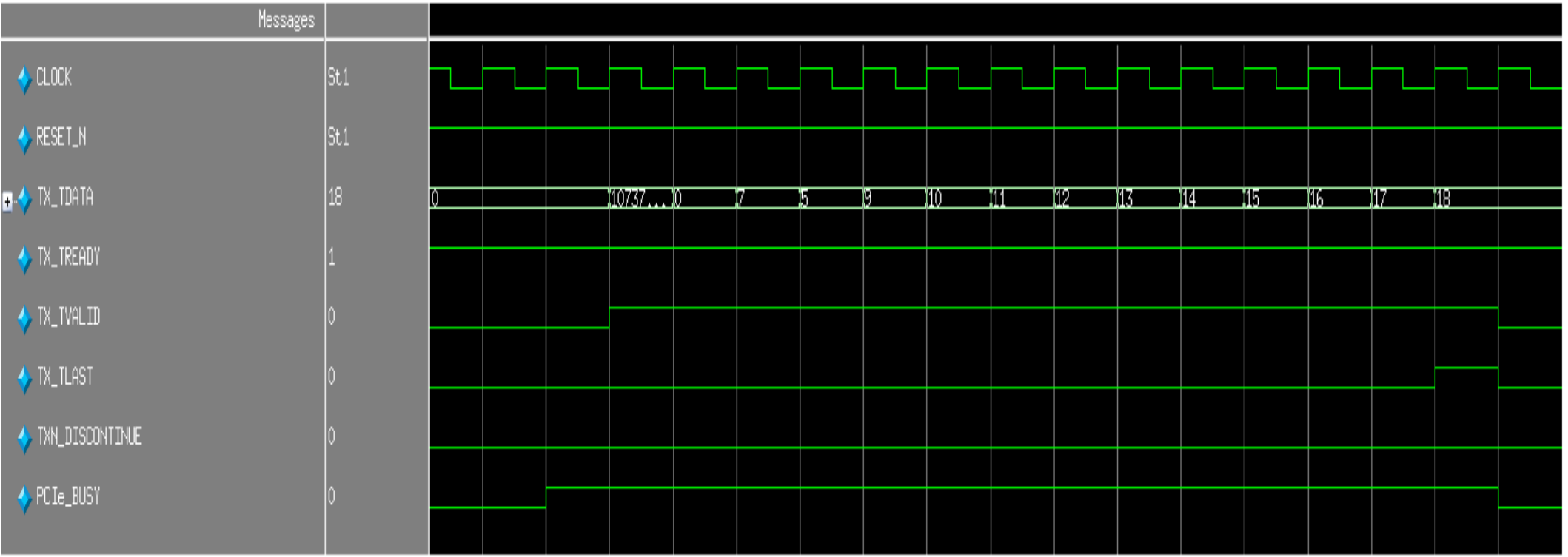


Figure : Simulation Result for a normal Write TLP Transmit Operation

SIMULATION RESULTS

Write TLP Transmit with “Source Throttling”

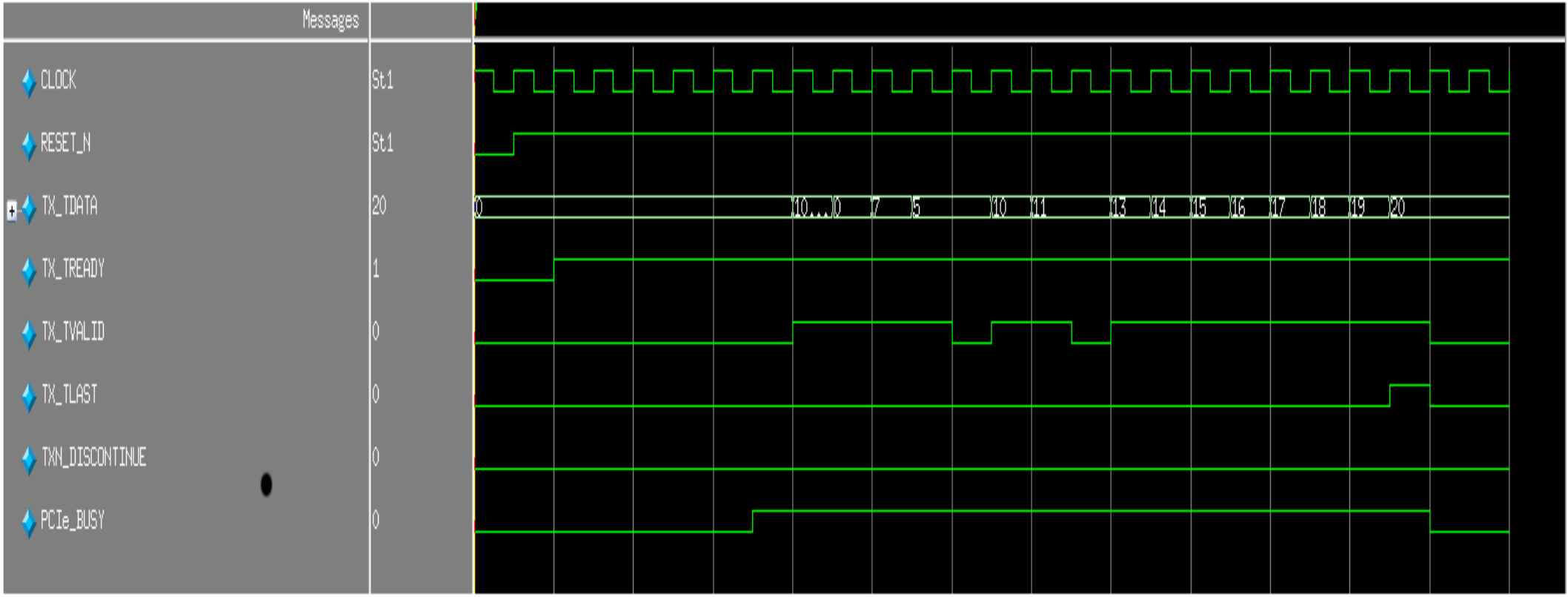


Figure : Simulation Result for a Write TLP Transmit with Source Throttling

SIMULATION RESULTS

Write TLP Transmit with “Destination Throttling”

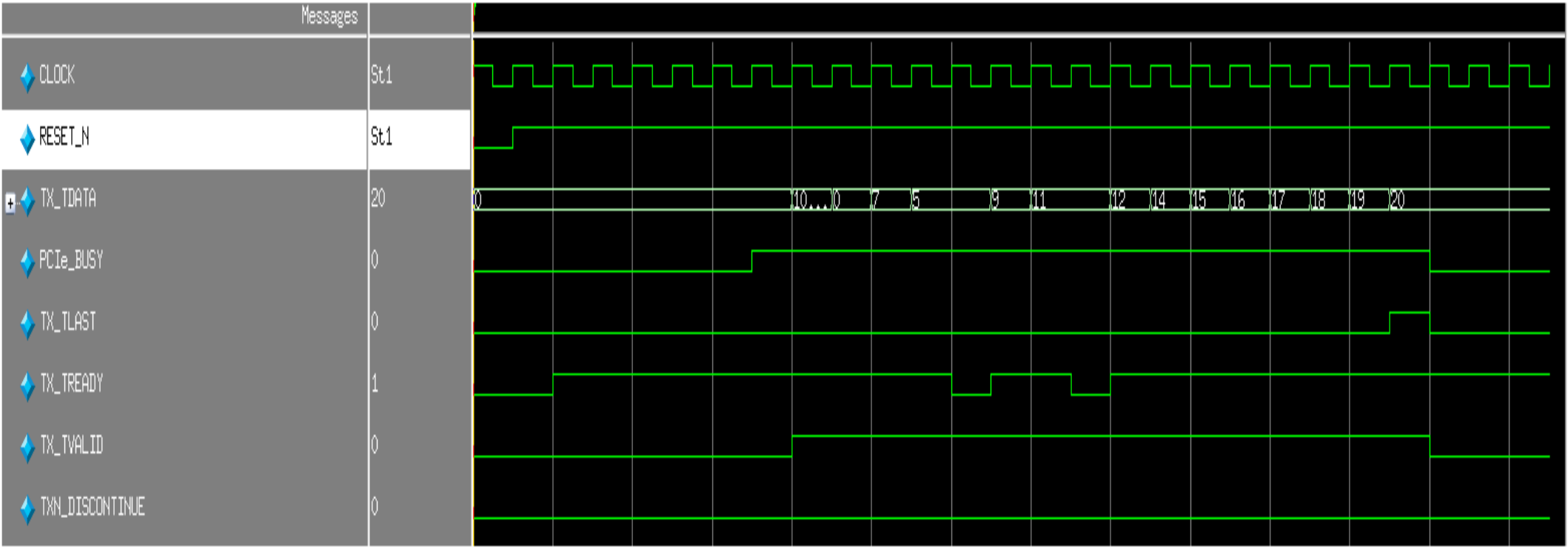


Figure : Simulation Result for a Write TLP Transmit with Destination Throtling

SIMULATION RESULTS

Write TLP Discontinue

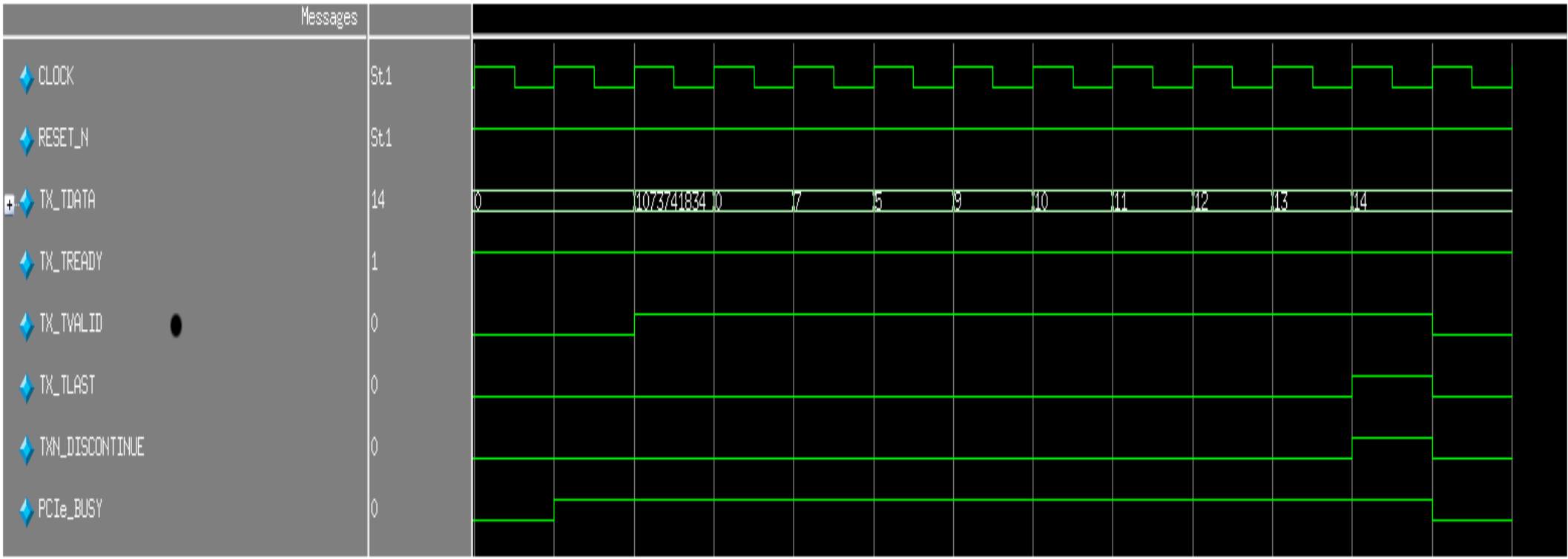


Figure : Simulation Result for Write TLP Transmit getting dicontinued

SIMULATION RESULTS

Completion TLP Receive with Incomplete Payload

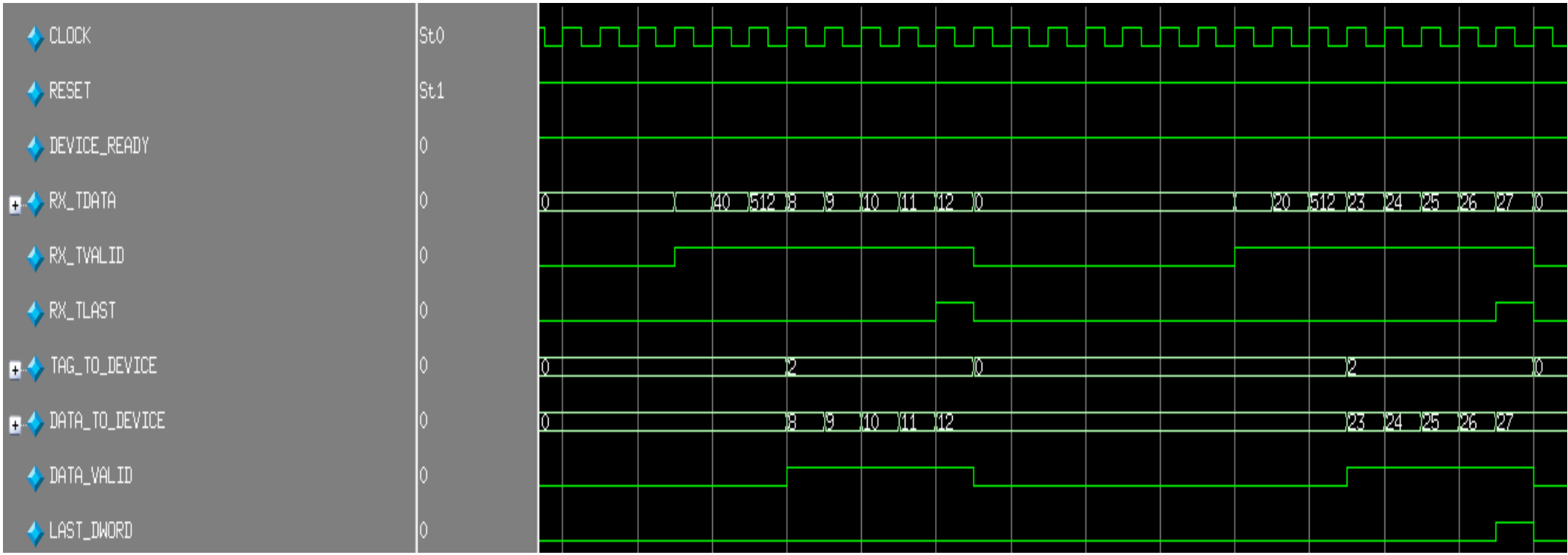


Figure : Simulation Result for Incomplete Receive of Completion TLP

SIMULATION RESULTS

Interrupt Aggregation

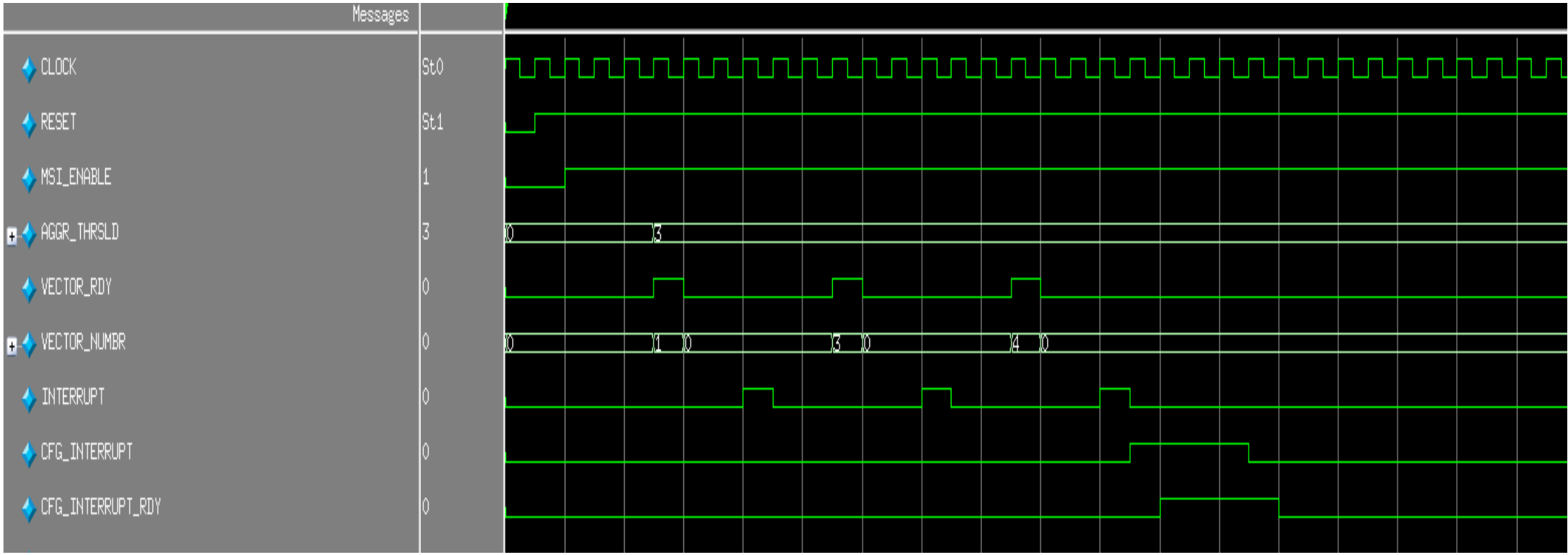


Figure : Simulation Result for three interrupt vectors being aggregated

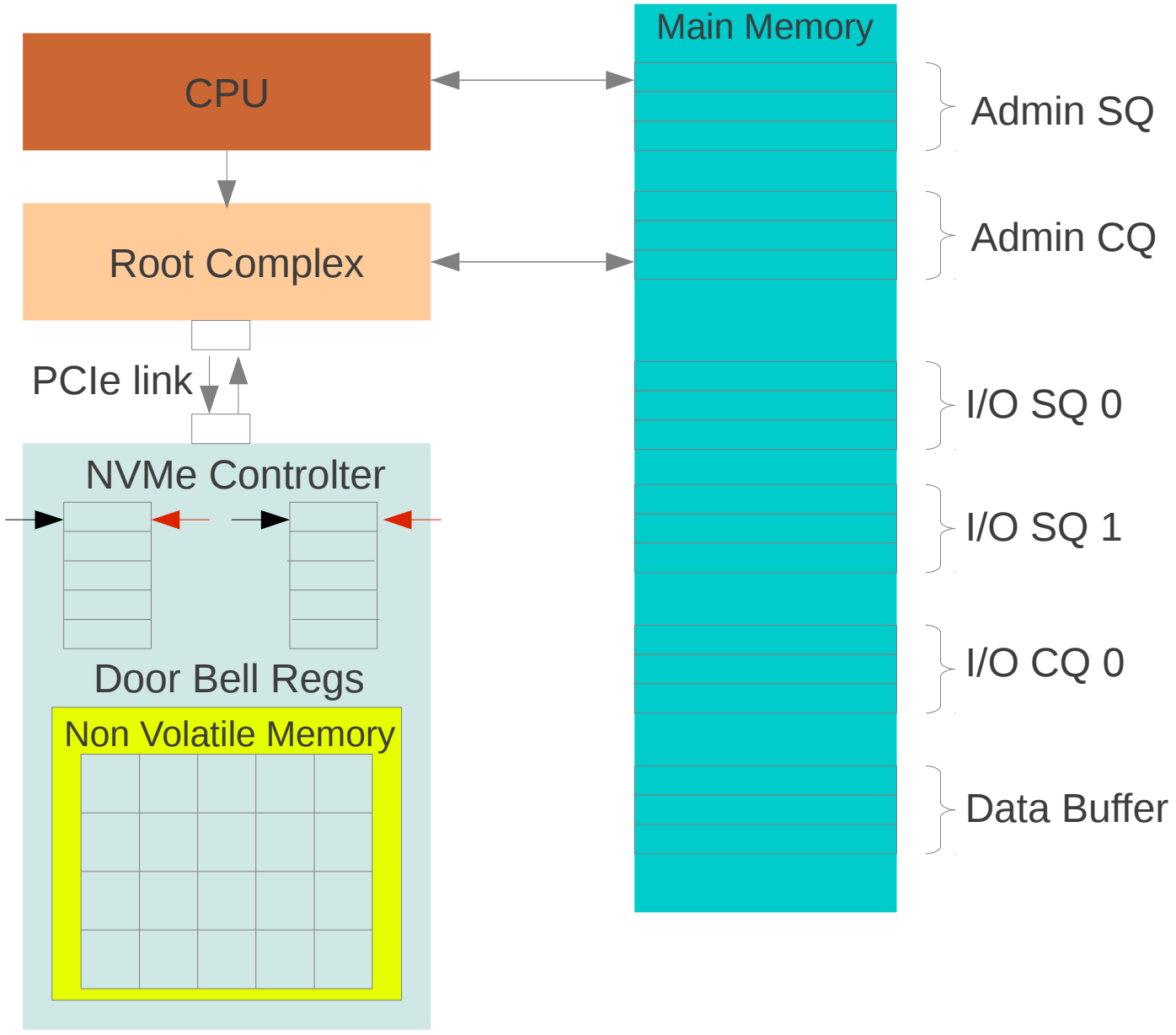
SYNTHESIS REPORT

SUMMARY OF SYNTHESIS REPORT

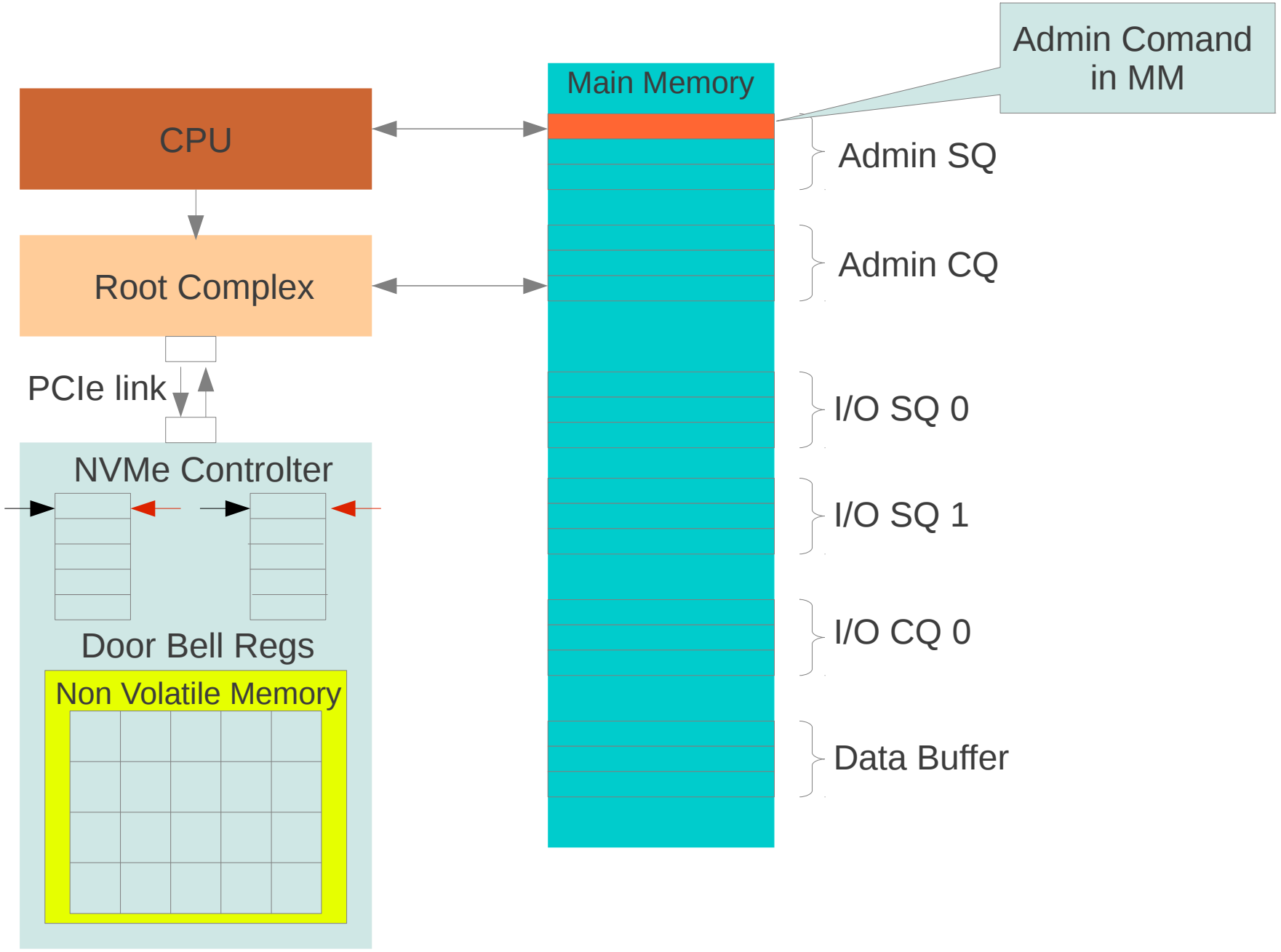
Feature	Summary
No.of Slice Regisers used	720
No.of Slice LUTs used	626
No. of LUT-FF Pairs used	432
Maximum Frequency of operation	295 MHz

NVMe ARCHITECTURE EXPLAINATION

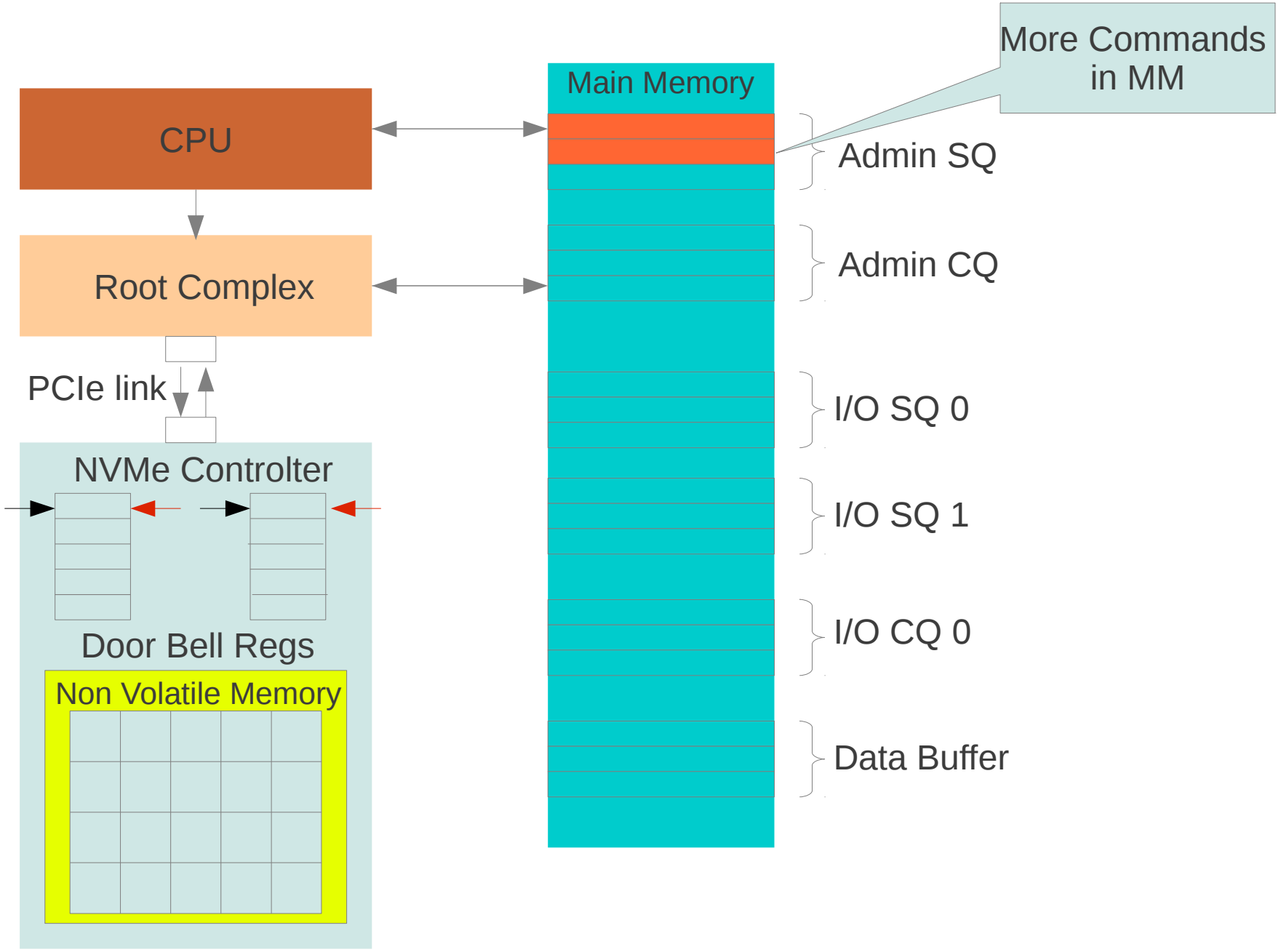
NVMe COMMAND PROCESSING



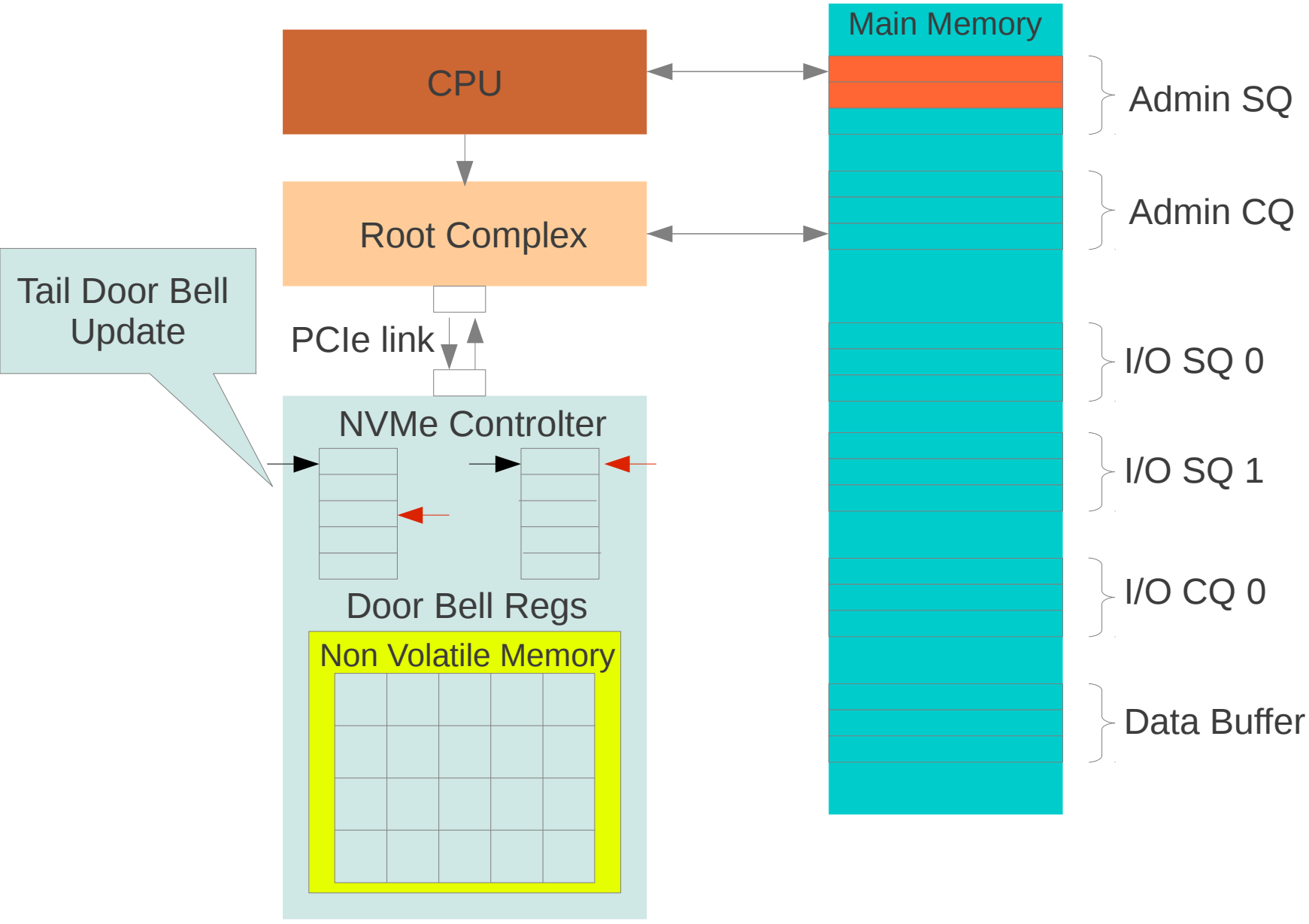
NVMe COMMAND PROCESSING



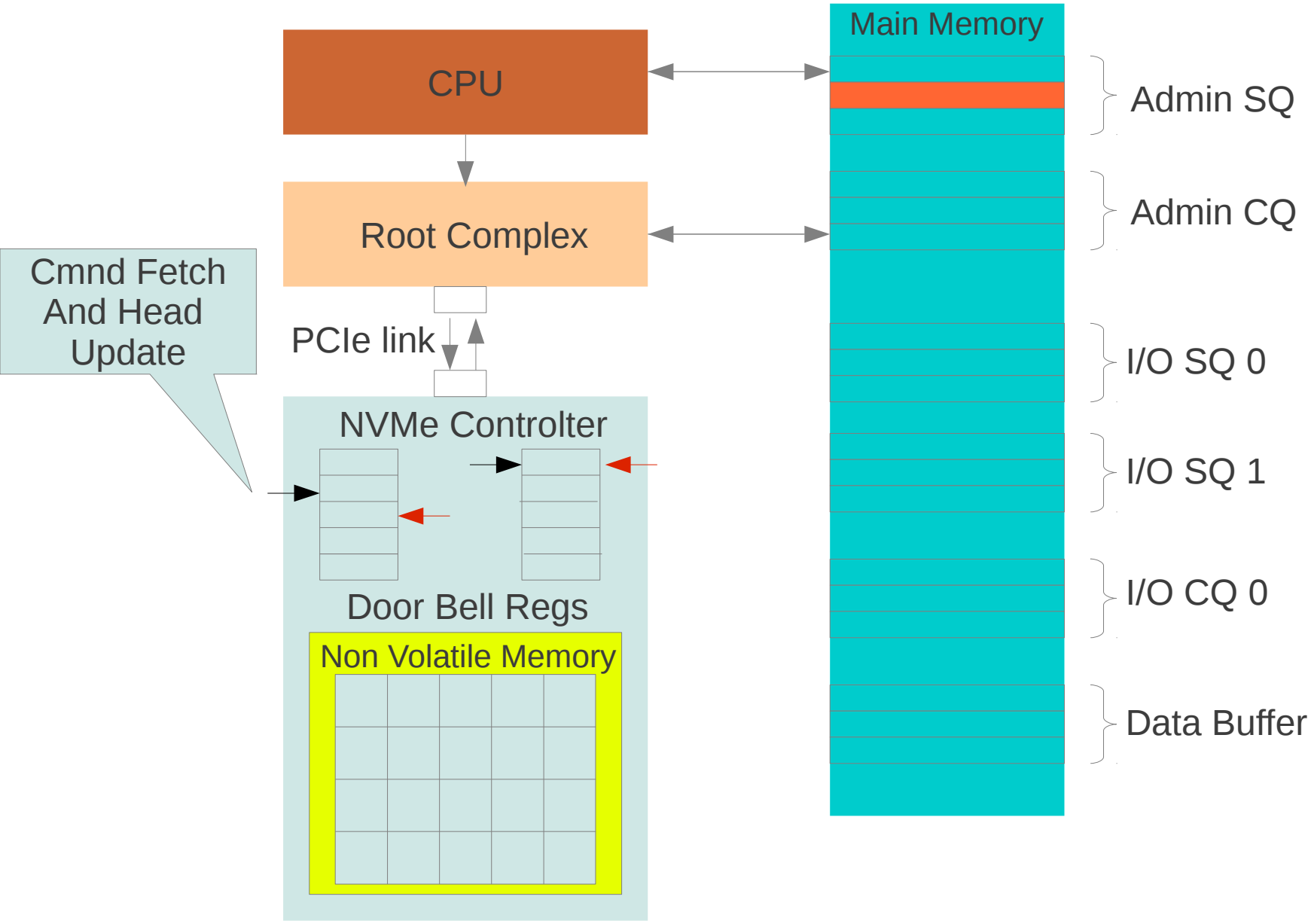
NVMe COMMAND PROCESSING



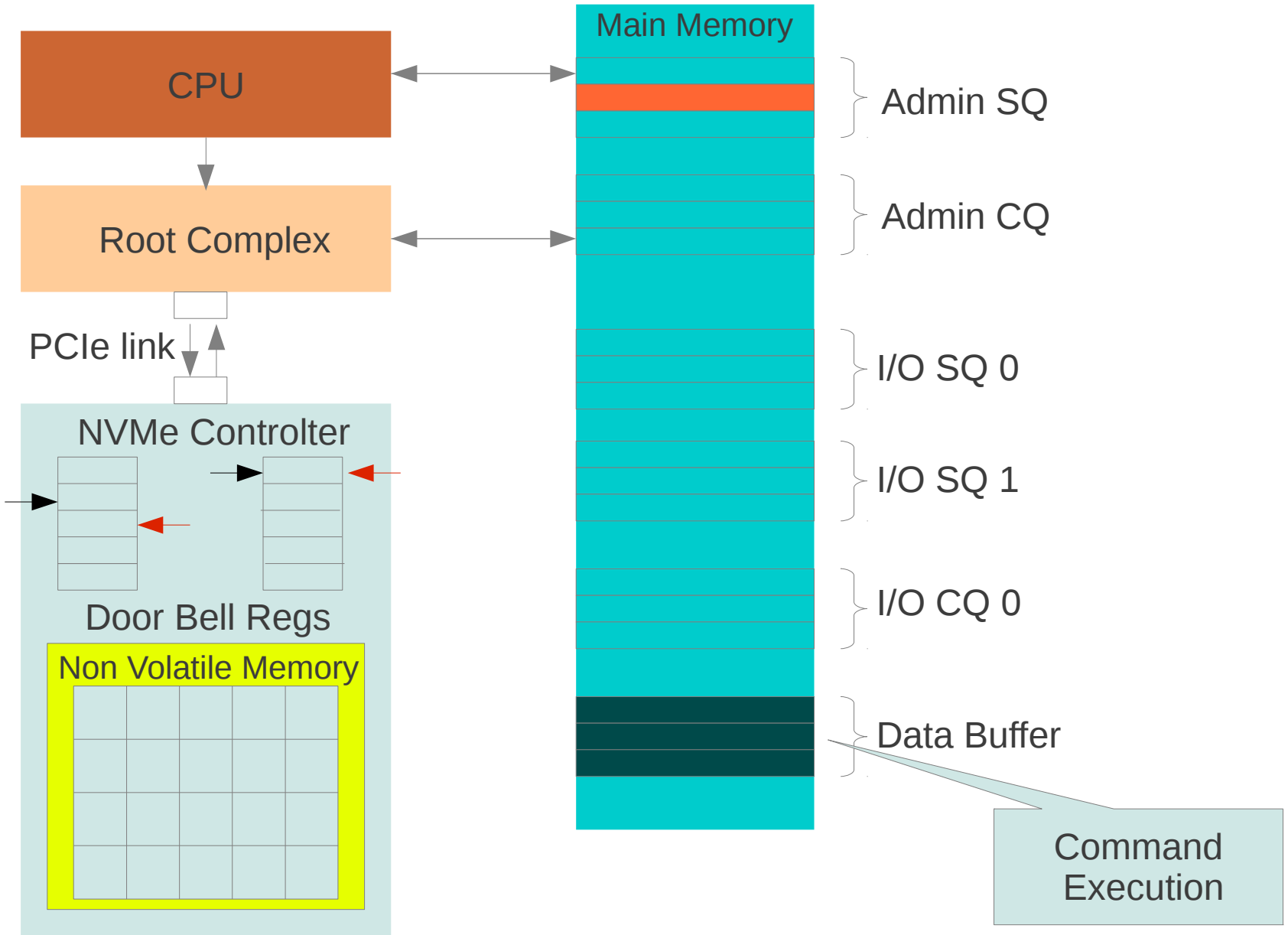
NVMe COMMAND PROCESSING



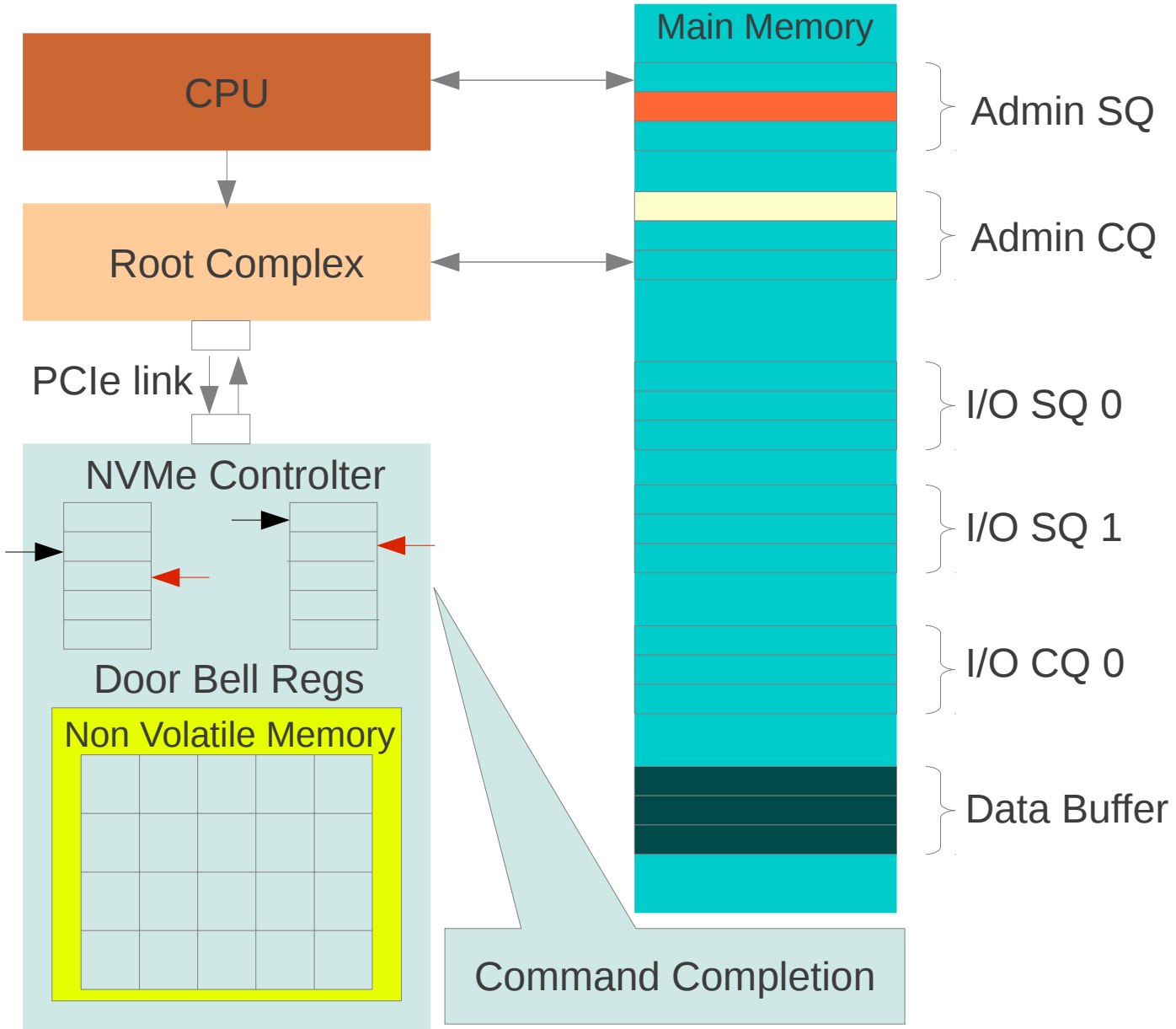
NVMe COMMAND PROCESSING



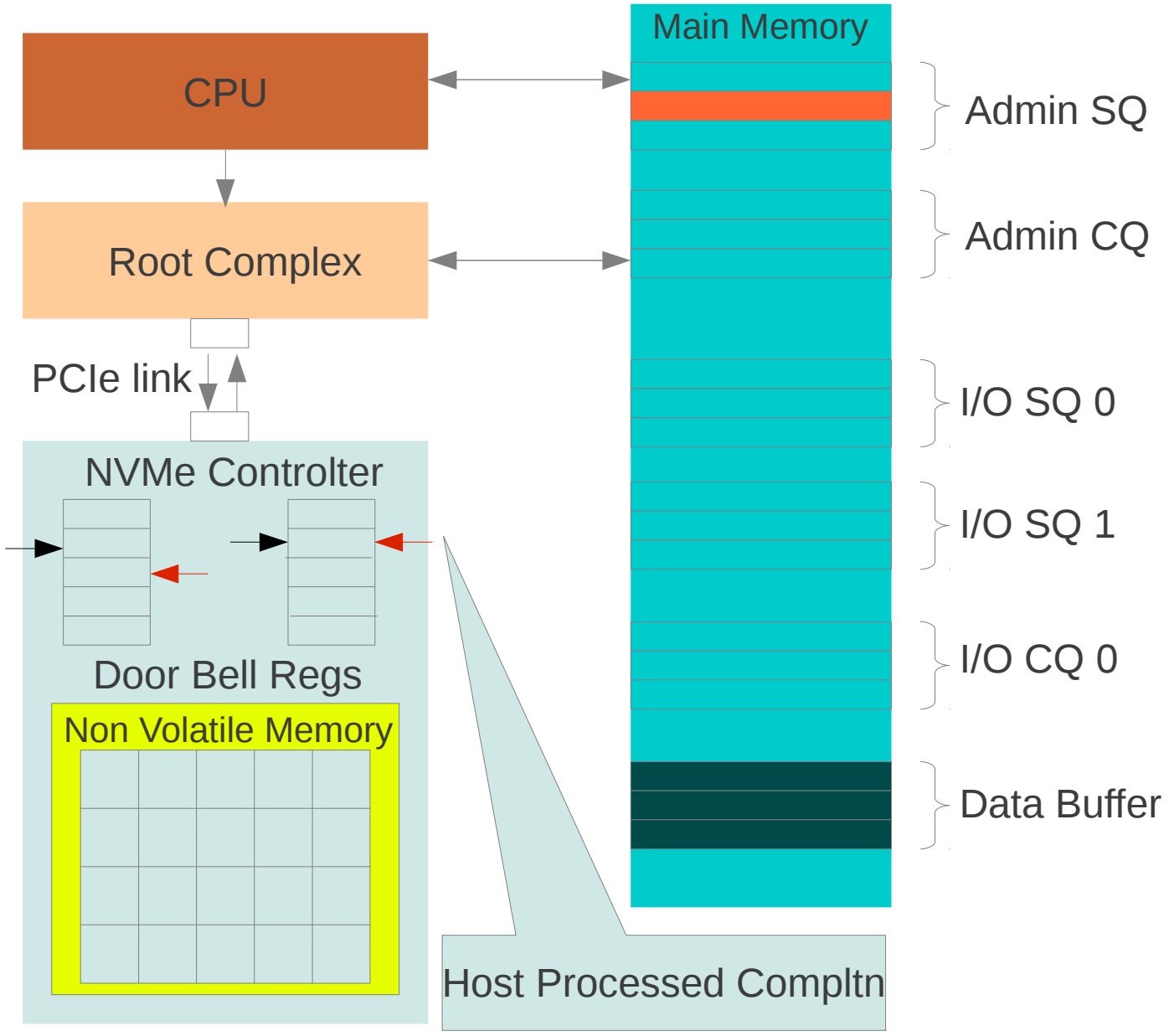
NVMe COMMAND PROCESSING



NVMe COMMAND PROCESSING



NVMe COMMAND PROCESSING



NVMe COMMAND STRUCTURE

Bytes	Description
63:60	**Command Dword 15 (CDW15):** This field is command specific Dword 15.
59:56	**Command Dword 14 (CDW14):** This field is command specific Dword 14.
55:52	**Command Dword 13 (CDW13):** This field is command specific Dword 13.
51:48	**Command Dword 12 (CDW12):** This field is command specific Dword 12.
47:44	**Command Dword 11 (CDW11):** This field is command specific Dword 11.
43:40	**Command Dword 10 (CDW10):** This field is command specific Dword 10.
39:32	**PRP Entry 2 (PRP2):** This field contains the second PRP entry for the command or if the data transfer spans more than two memory pages, then this field is a PRP List pointer.
31:24	**PRP Entry 1 (PRP1):** This field contains the first PRP entry for the command or a PRP List pointer depending on the command.
23:16	**Metadata Pointer (MPTR):** This field contains the address of a contiguous physical buffer of metadata. This field is only used if metadata is not interleaved with the logical block data, as specified in the Format NVM command. This field shall be Dword aligned.
15:08	Reserved
07:04	**Namespace Identifier (NSID):** This field specifies the namespace that this command applies to. If the namespace is not used for the command, then this field shall be cleared to 0h. If a command shall be applied to all namespaces on the device, then this value shall be set to FFFFFFFFh.
03:00	**Command Dword 0 (CDW0):** This field is common to all commands and is defined in Figure 6.

Figure: General NVM Command Structure

NVMe COMMAND COMPLETION

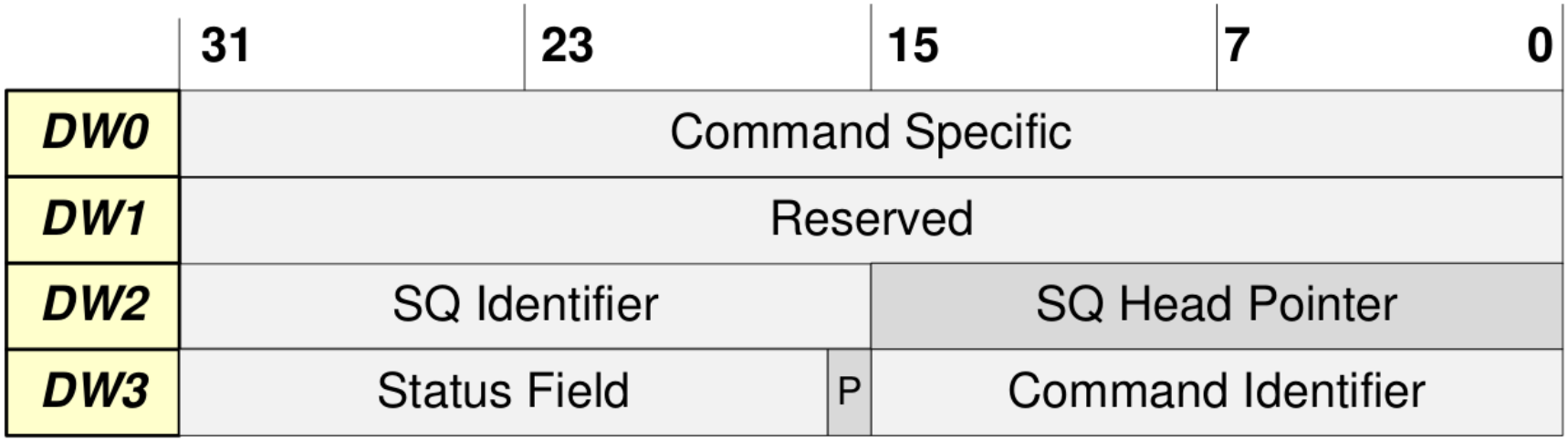


Figure: General Completion Entry Structure

COMMAND SET

ADMIN COMMAND SET

OPCODE	COMMAND
00h	Delete I/O Submission Queue
01h	Create I/O Submission Queue
02h	Get LOG Page
04h	Delete I/O Completion Queue
05h	Create I/O Completion Queue
06h	Identify
08h	Abort
09h	Set Features
0Ah	Get Features
0Ch	Asynchronous Event Request

NVM COMMAND SET

OPCODE	COMMAND
01b	Write
10b	Read

Figure: Showing Admin and NVM Command Sets with their Opcodes

FEATURES INFORMATION

Feature	Implementation
Arbitration Mechanism	Round-Robin with complete burst
LBA information	All LBAs are of the type “File System”
No. of Queues Requested / No. Of Queues Allocated	Configurable Synthesised for 16 SQ and 16 CQ s With 64k depth for contiguous buffers And 512 depth for non contiguous buffers
Interupt Coalesceing Feature	Supported
Interrupt Vector Configuration	Supported on a per vector basis

CONTROLLER INITIALIZATION

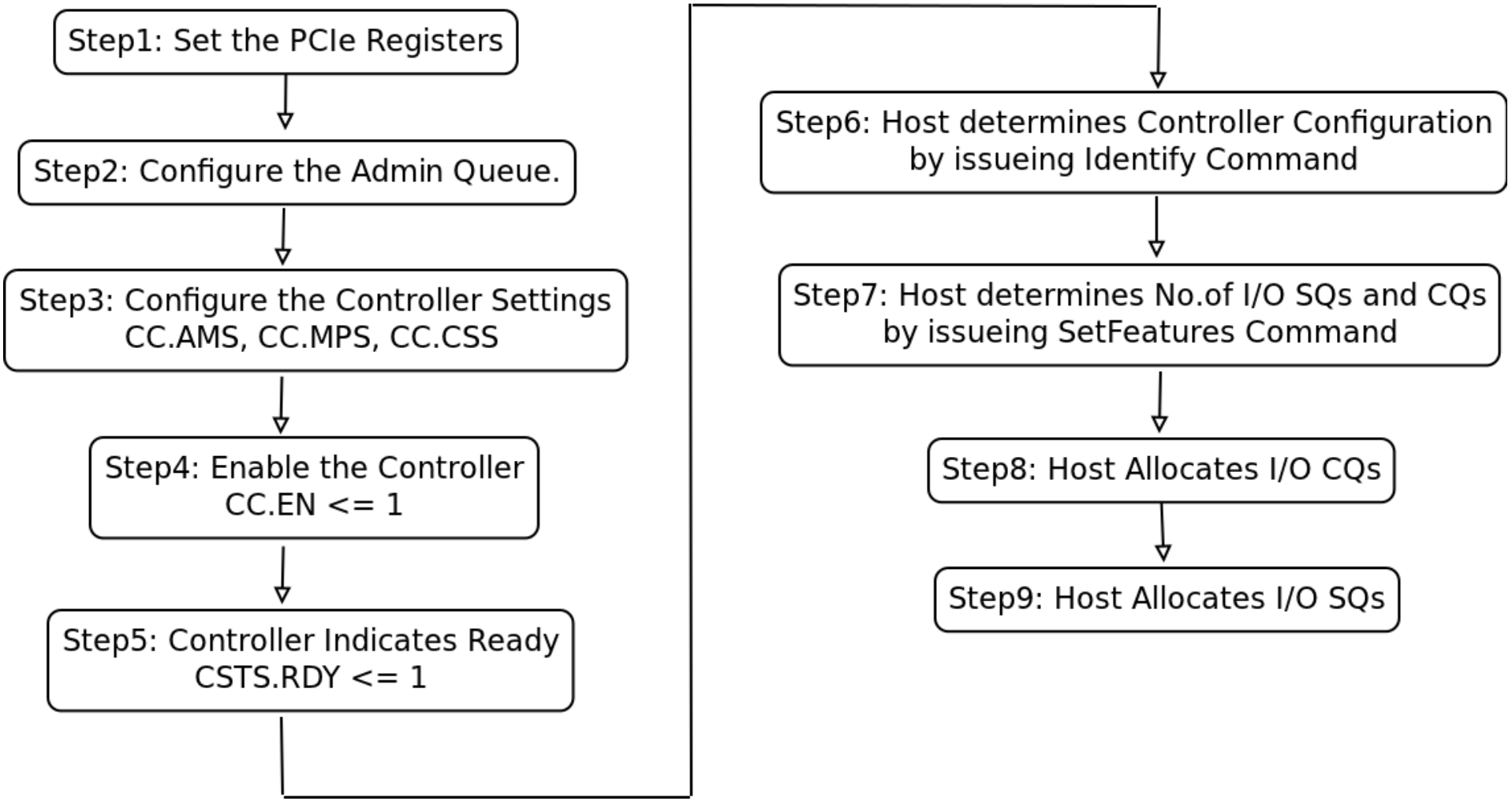


Figure: Flowchart showing steps to initialize the controller

CONTROLLER SHUTDOWN

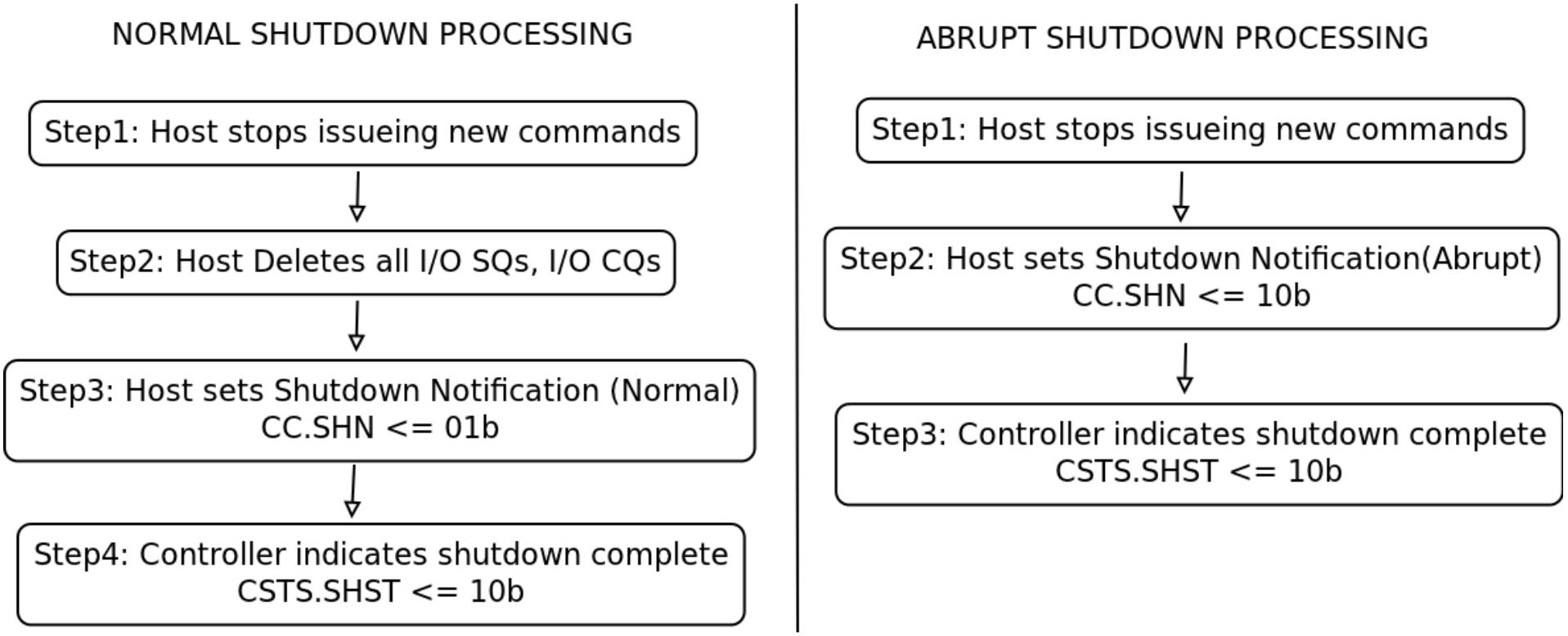
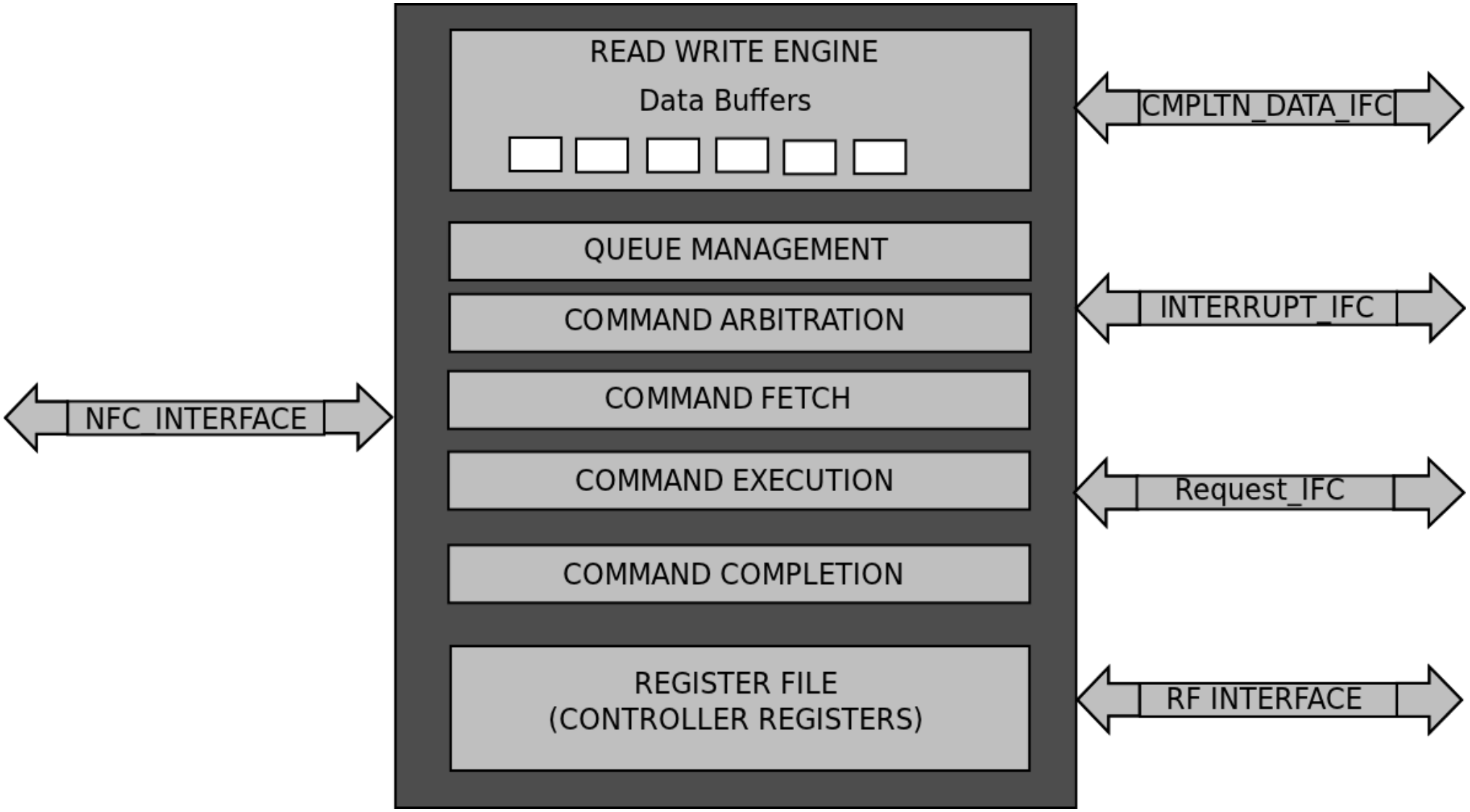


Figure: Flowchart showing Steps for Normal and Abrupt shutdown of controller

NVMe IMPLEMENTATION

NVMe IMPLEMENTAION



INTERFACE DESCRIPTION

INTERFACE DESCRIPTION

Nand Flash Controller Interface Signals

Name	Direction	Description
rg_address_to_nand[11:0]	Output	Address to NAND's memory mapped register file and Buffers
rg_data_to_nand[31:0]	Output	Data to NAND's memory mapped register file and Buffers
wr_data_from_nand[31:0]	Input	Data received from NAND
rg_chip_enable	Output	Chip Enbale signal to NAND
rg_write_enable	Output	Write Enbale signal to NAND
rg_output_enable	Output	Output Enbale signal to NAND
wr_interrupt	Input	Interrupt From NAND indicating Read Data Ready in buffers
wr_read_busy	Input	Indicate Nand is Ready or Busy

Table : A brief description of the signals present in the NFC Interface

**The interface definition is present in InterfaceNvmController.bsv*

NVMe MODULES AND STATE MACHINES

NVMe MODULES AND STATE MACHINES

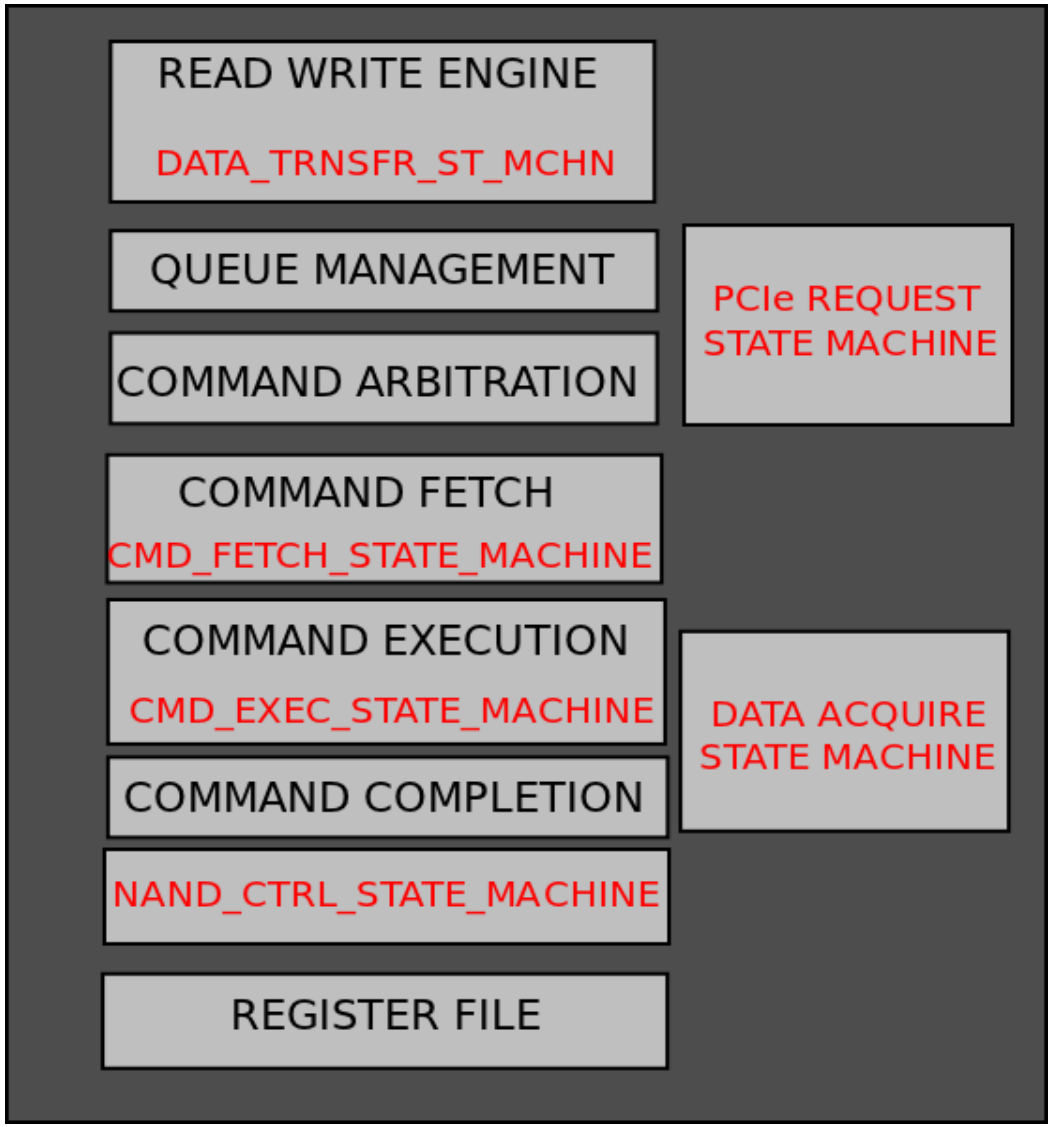


Figure: Block diagram showing modules and the state machines used to implement them

READ WRITE ENGINE

- Used to perform data transfers between PCIe and NVMe Controller.
- Data-Transfer State machine is used to implement the module.
- Data buffers are used for the following purposes:
  - Store a page (4KB) .
  - Store the data structures :
    - Controller data structure (4KB).
    - Namespace data structure (4KB).
  - Controller Features (4KB)
  - Error Logs (5x64 B)

QUEUE MANAGEMENT

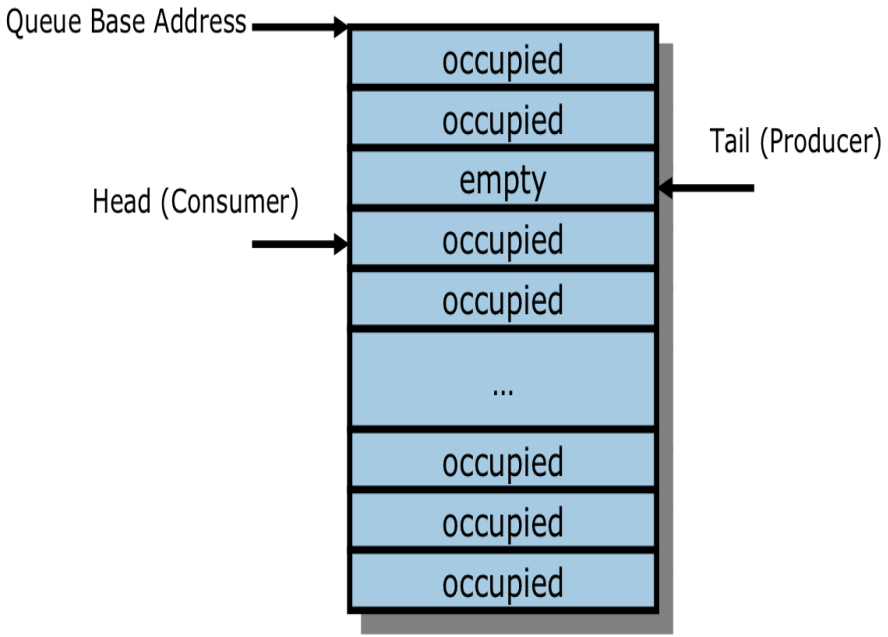


Figure: Q-Full Condition

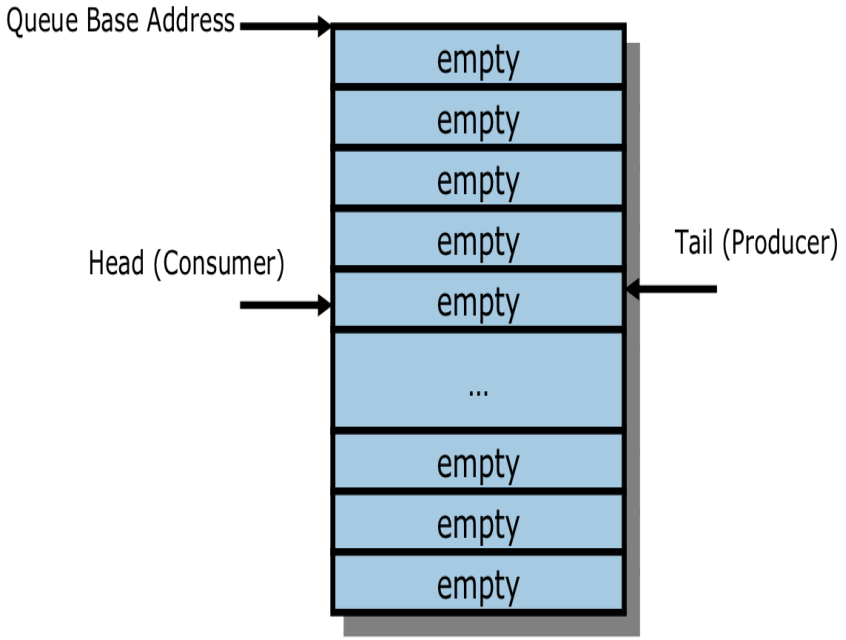


Figure: Q-Empty Condition

Completion Queue:

- Consumer : Host
- Producer : NVMe

Submission Queue:

- Consumer : NVMe
- Producer : PCIe

COMMAND ARBITRATION

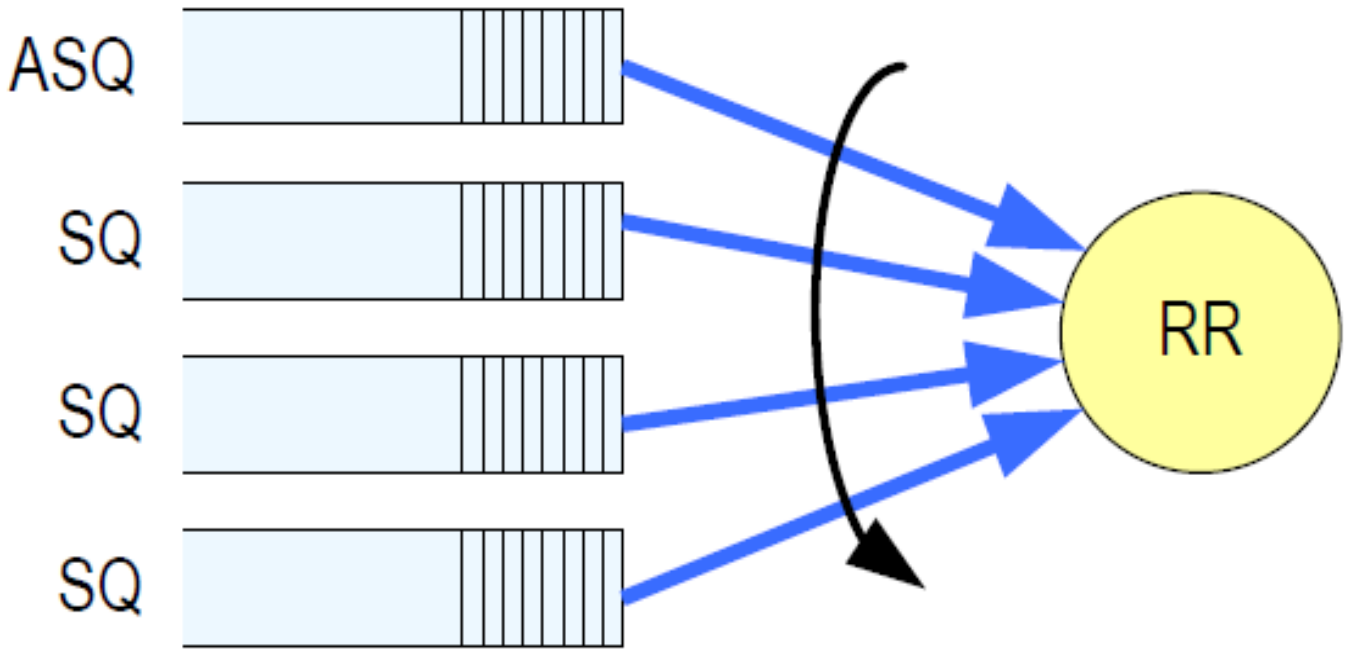


Figure: Round Robin Arbitration Mechanism for Commands

- Round Robin Arbitation used.
- Admin SQ has highest priority.
- I/O SQ priority is according to the SQ Identifier Number.

COMMAND FETCH

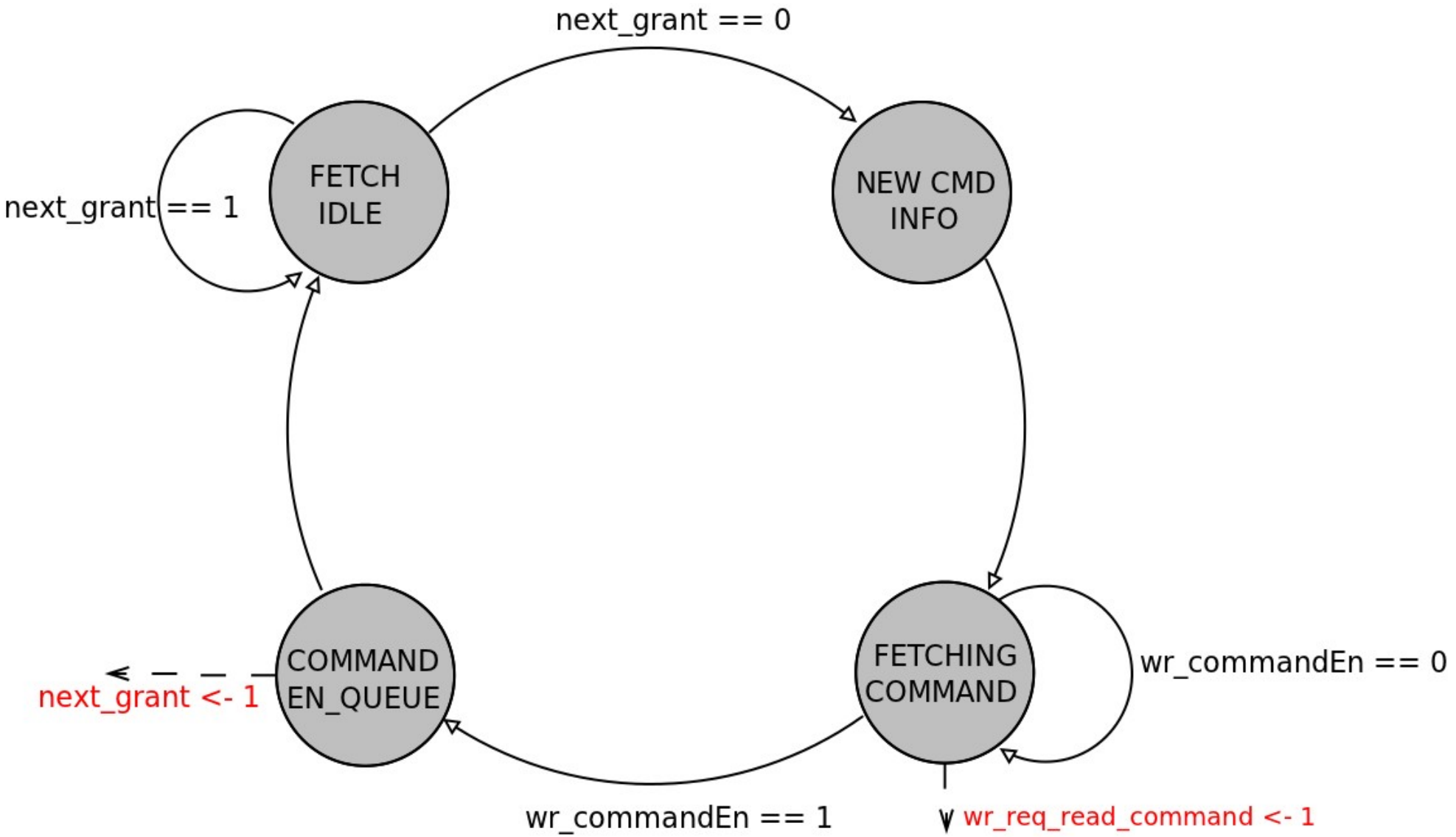
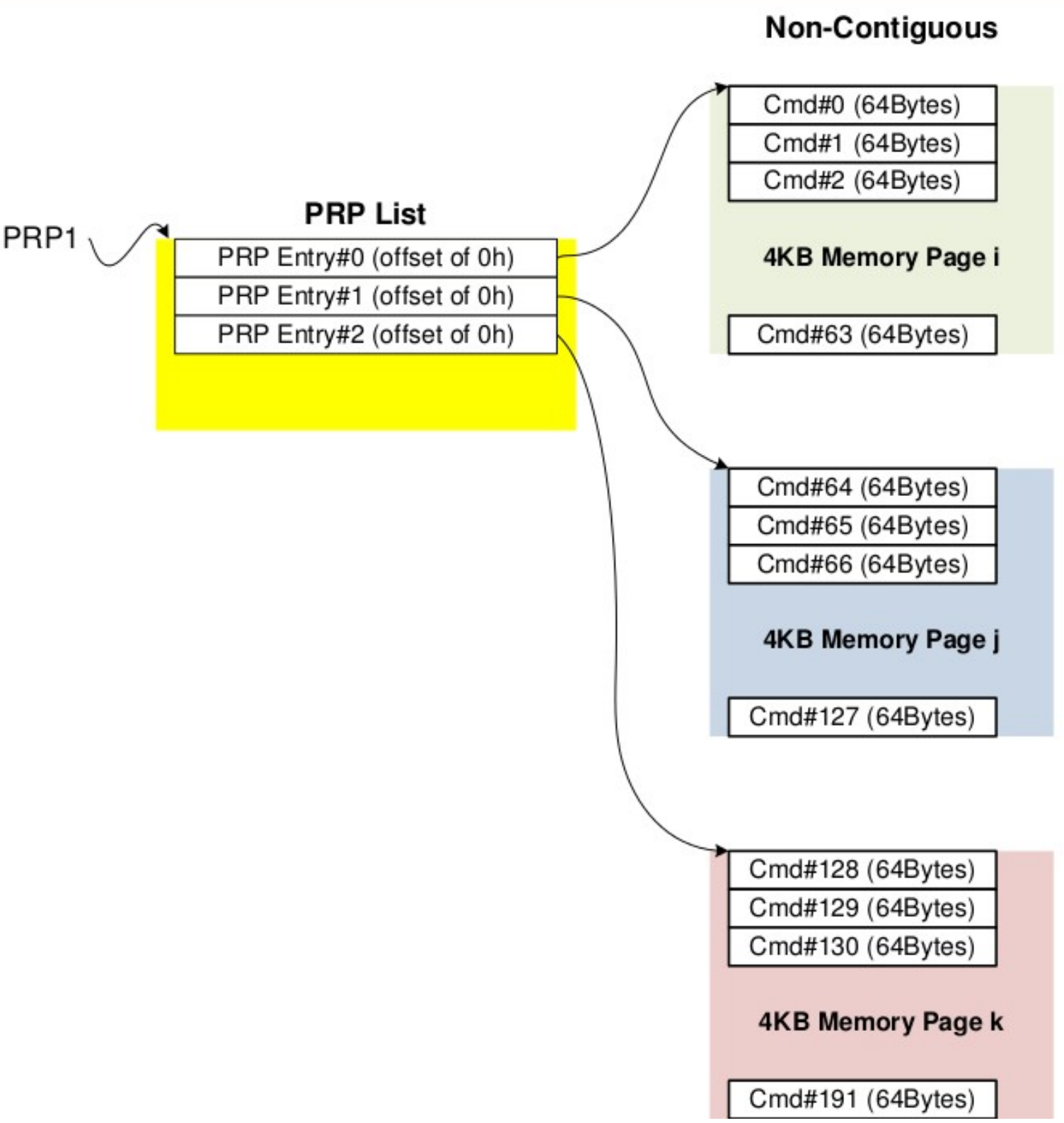


Figure: State Machine for Command Fetch


Non-Contiguous
PRP1
PRP List
PRP Entry#0 (offset of 0h)
PRP Entry#1 (offset of 0h)
PRP Entry#2 (offset of 0h)
Cmd#0 (64Bytes)
Cmd#1 (64Bytes)
Cmd#2 (64Bytes)
4KB Memory Page i
Cmd#63 (64Bytes)
Cmd#64 (64Bytes)
Cmd#65 (64Bytes)
Cmd#66 (64Bytes)
4KB Memory Page j
Cmd#127 (64Bytes)
Cmd#128 (64Bytes)
Cmd#129 (64Bytes)
Cmd#130 (64Bytes)
4KB Memory Page k
Cmd#191 (64Bytes)

COMMAND EXECUTION

ASQ COMMAND EXECUTION

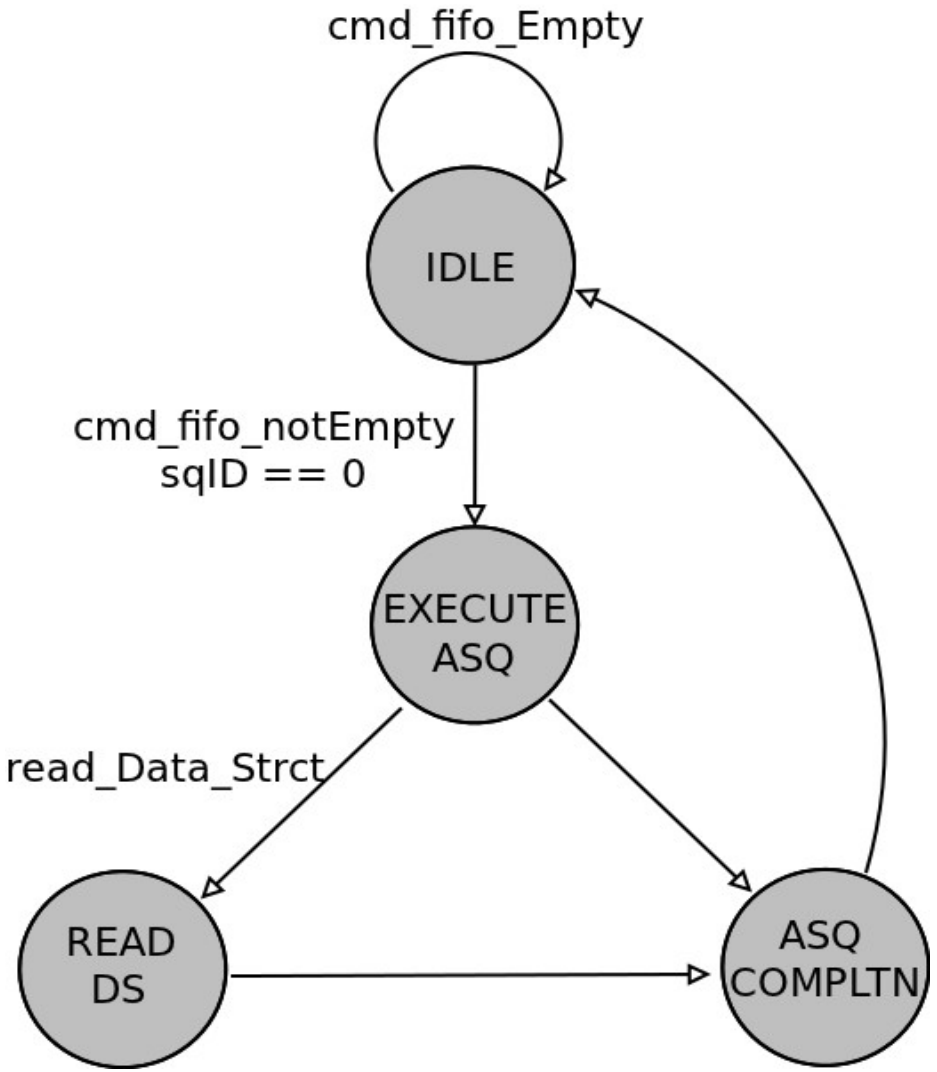


Figure: State Machine for ASQ Command Execution

PRP LIST ACQUIRE

tag [1:0]	tag [2]	tag [6:3]	Description	Txn Initiating Unit	Txn Receiving Unit
00	X	XXXX	Not Used	None	None
01	X	XXXX	Acquire Command	Fetch Unit	Command-Data Acquire State Machine
10	X	XXXX	Acquire Data	Data transfer Unit	Command-Data Acquire State-Machine
11	0	0000 to 1111	Acquire CQ 0 to CQ 15 PRP List	Execution Unit	PRP-List Acquire Rule `rl_prp_list_acquire`
11	1	0000 to 1111	Acquire SQ 0 to SQ 15 PRP List	Execution Unit	PRP-List Acquire Rule `rl_prp_list_acquire`

COMMAND-DATA ACQUIRE STATE MACHINE

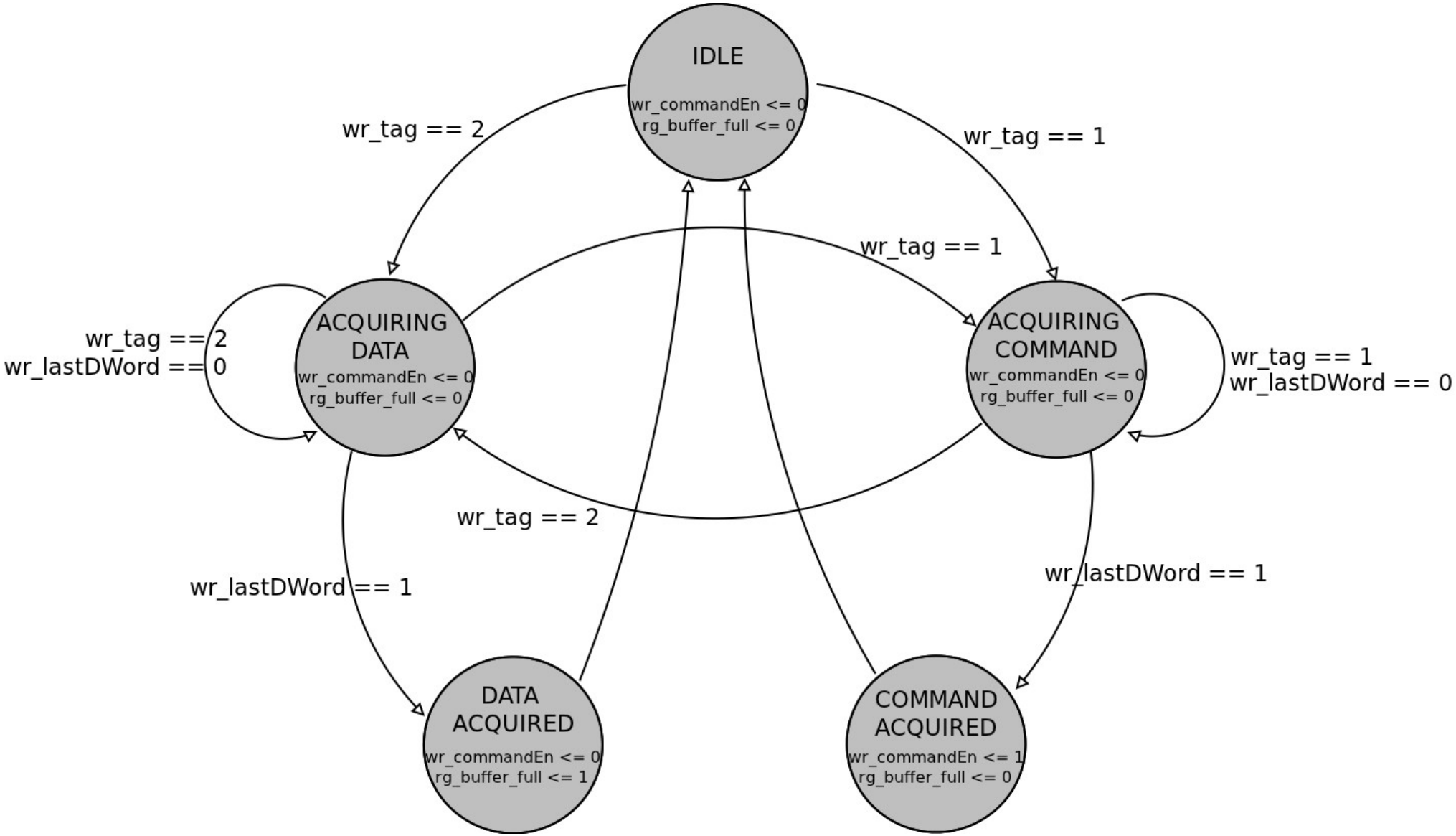


Figure: State Machine for acquiring Command and Data

COMMAND EXECUTION

ISQ COMMAND EXECUTION

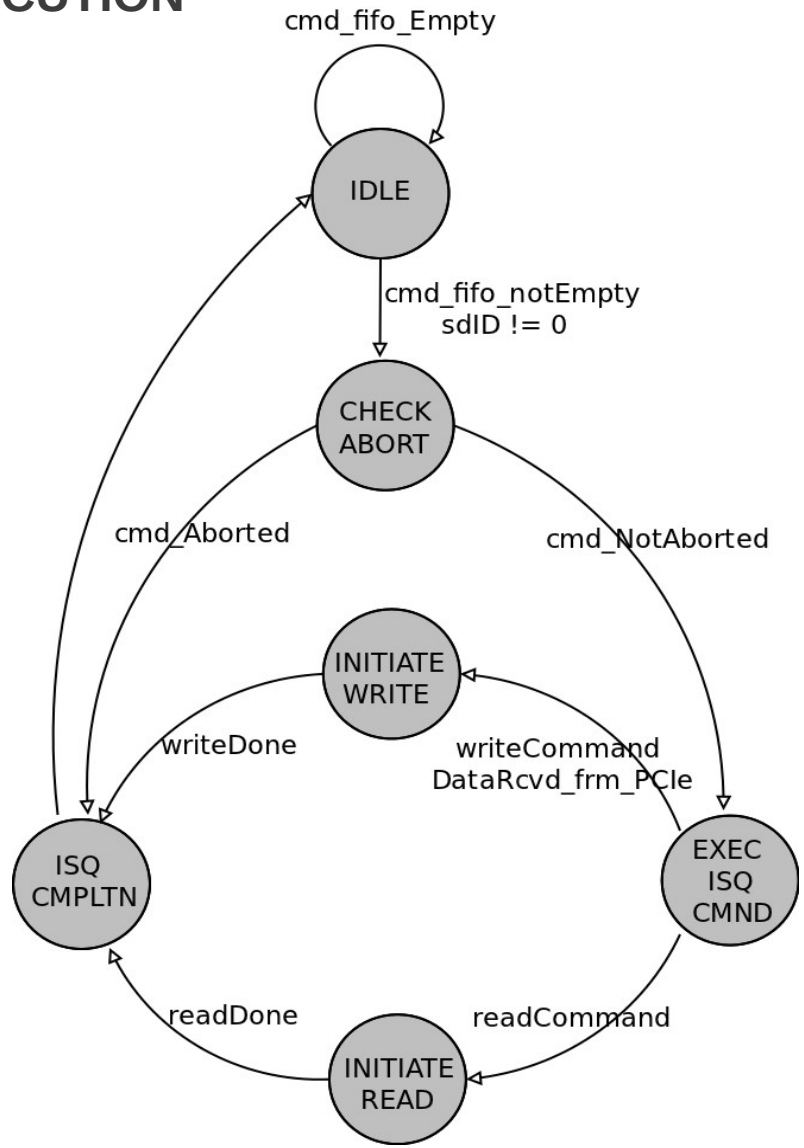


Figure: State Machine for ISQ Command Execution

COMMAND EXECUTION

Sample Abort List

Valid-bit	Command ID	SQ ID
Valid	20	1
In-Valid	18	3
In-Valid	45	1
Valid	57	5
Valid	59	4

DATA TRANSFER STATE MACHINE

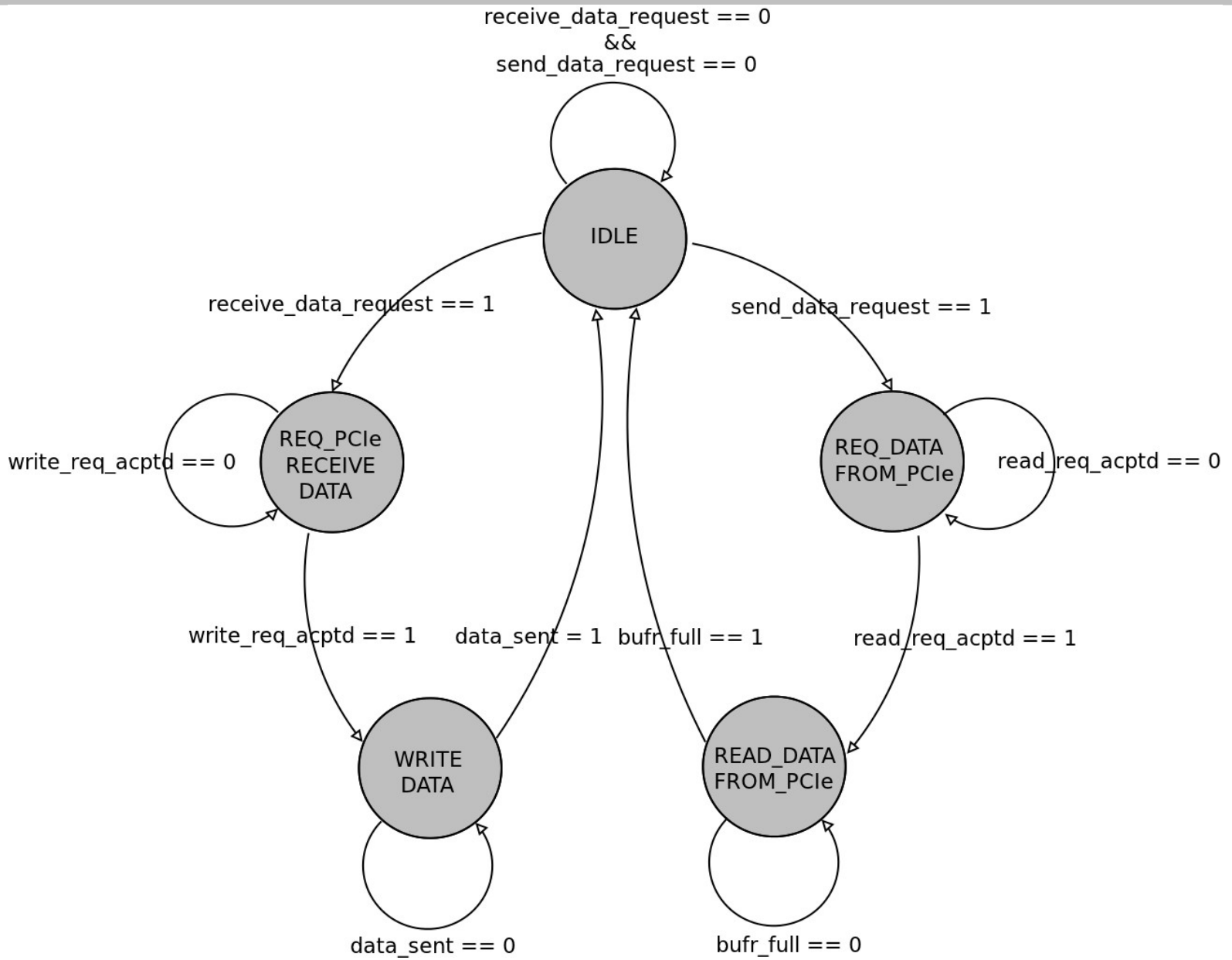


Figure: States for Data-transfer Statemachine

PCIe REQUEST STATE MACHINE

COMMAND COMPLETION

- Controller posts completions to the Completion Queue through the PCIe write TLPs.
- SQ Identifier and Command Identifier fields are used to unquely identify the command to be completed.
- The status of the Command is indicated through the “Status Field” in the CQ Entry.
- The host can re-try with the same command if there is any error.
- CQ Tail is updated by the controller after posting the completion.
- CQ head is updated by the host after processing the completion.

INTERRUPTS

- Controller Interrupts the host after posting the Completions.
- Each completion is associated with an interrupt vector number.
- Controller implements MSI based interrupts.
- Efficient reporting of interrupts with least amount of overhead to host.
- Support for interrupt aggregation or interrupt coalescing.
- Aggregation Threshold in the Interrupt Coalescing feature is specified by the host for minimum interrupt processing overhead.

INTERRUPT ARCHITECTURE

- Interrupt Registers :
  - Interrupt Mask Register (IM) -------- 32 bits
  - Interrupt Status Register (IS) -------- 32 bits
  - Interrupt Mask Clear (INTMC) -------- 32 bits
  - Interrupt Mask Set (INTMS) -------- 32 bits
  - Interrupt Vector Number (IV) -------- 32 bits
  - Interrupt Vector Ready (Irdy) -------- 1 bit
- Interrupt Event Detection
- Interrupt Aggregation

INTERRUPT REGISTERS

Interrupt Mask Register

- Indicates Mask/Unmask condition of each vector.
- IM[5] set high, implies Vector-5 is Masked.
- A pos-edge on INTMC[x] clears IM[x].
- A pos-edge on INTMS[x] sets IM[x].
- Neg-edges have no effect.

INTERRUPT REGISTERS

Interrupt Status Register

- Indicates the Status of each vector.
- A bit is set if the AND of following conditions occur:
  - Un-ack CQ entry utilizes this vector
  - The CQ has interrupts “enabled”
  - Corresponding interrupt vector is “Unmasked”
- A bit is reset after an “Event” is generated for that vector.
- Host has NO access to this register at all.

INTERRUPT REGISTERS

Interrupt Mask Clear(INTMC)

- It is a part of Controller Registers, hence Host can access it.
- To unmask vector-3 , write a value of 08h to INTMC.
- Read operation to this register returns “Mask Register” .

Interrupt Mask Set (INTMS)

- It is a part of Controller Registers, hence Host can access it.
- To mask vector-3 , write a value of 08h to INTMS.
- Read operation to this register returns “Mask Register” .

Interrupt Vector(IV) Number

- Indicates the vector number to PCIe controller.

Interrupt Ready

- Indicates that message number on IV register is valid.

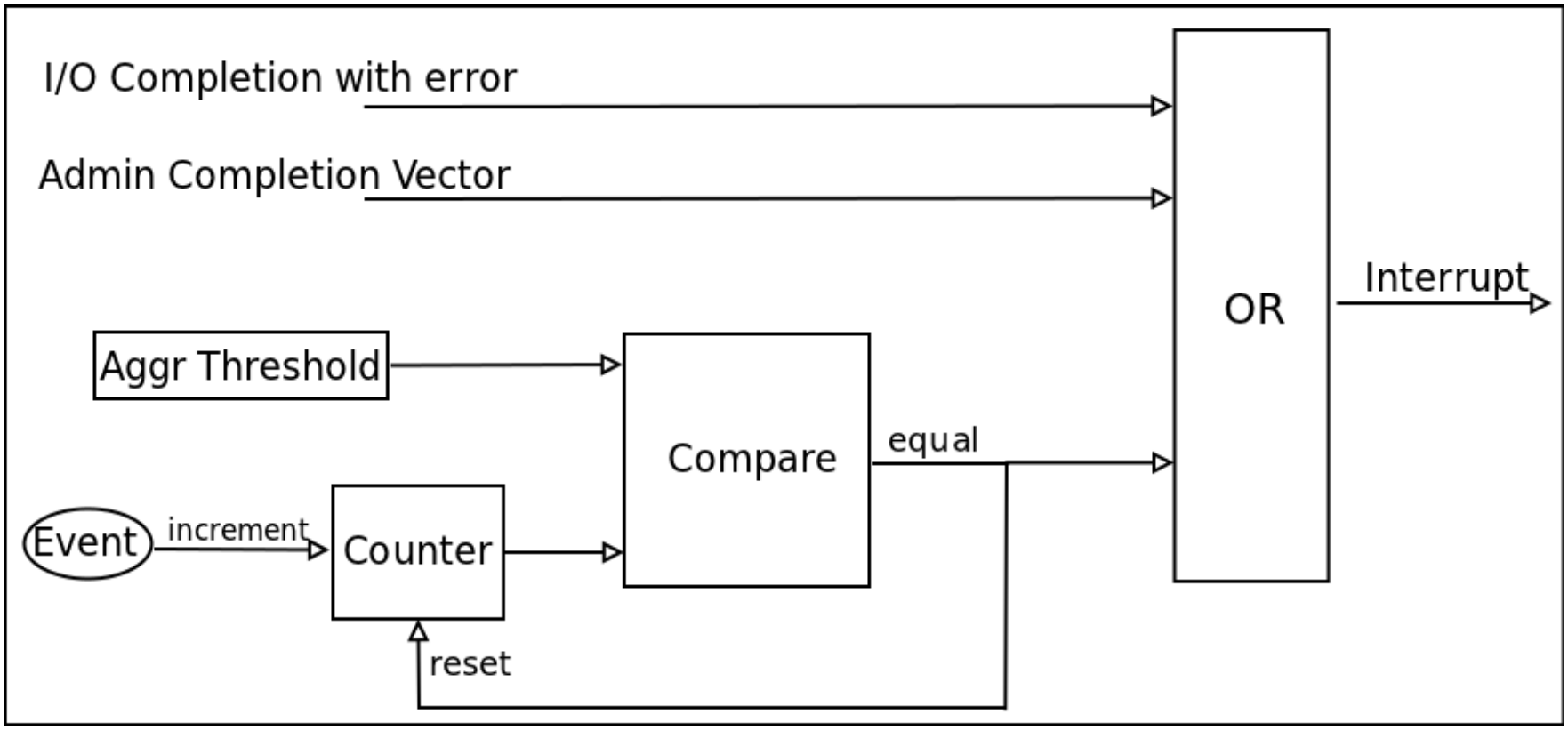
INTERRUPT EVENT

- A positive edge on the Status Register is detected as an event.
- If the vector is masked, then there is no need to interrupt the host.

Maskregisterbit	*StatusRegisterbit*	*Event*	*Description*
Steady 1	Will be steady 0	No Event	Interrupt Masked
Steady 0	Steady 0	No Event	No Interrupt
Transition from 1 to 0	Will be steady 0	No Event	No interrupt
Steady 0	Transition from 0 to 1	Event	Interrupt Sent

INTERRUPT AGGREGATION

- Interrupts are aggregated on a per vector basis.
- Admin CQ interrupts and I/O commands that complete in “Error” are NOT aggregated.



NAND CONTROL STATES

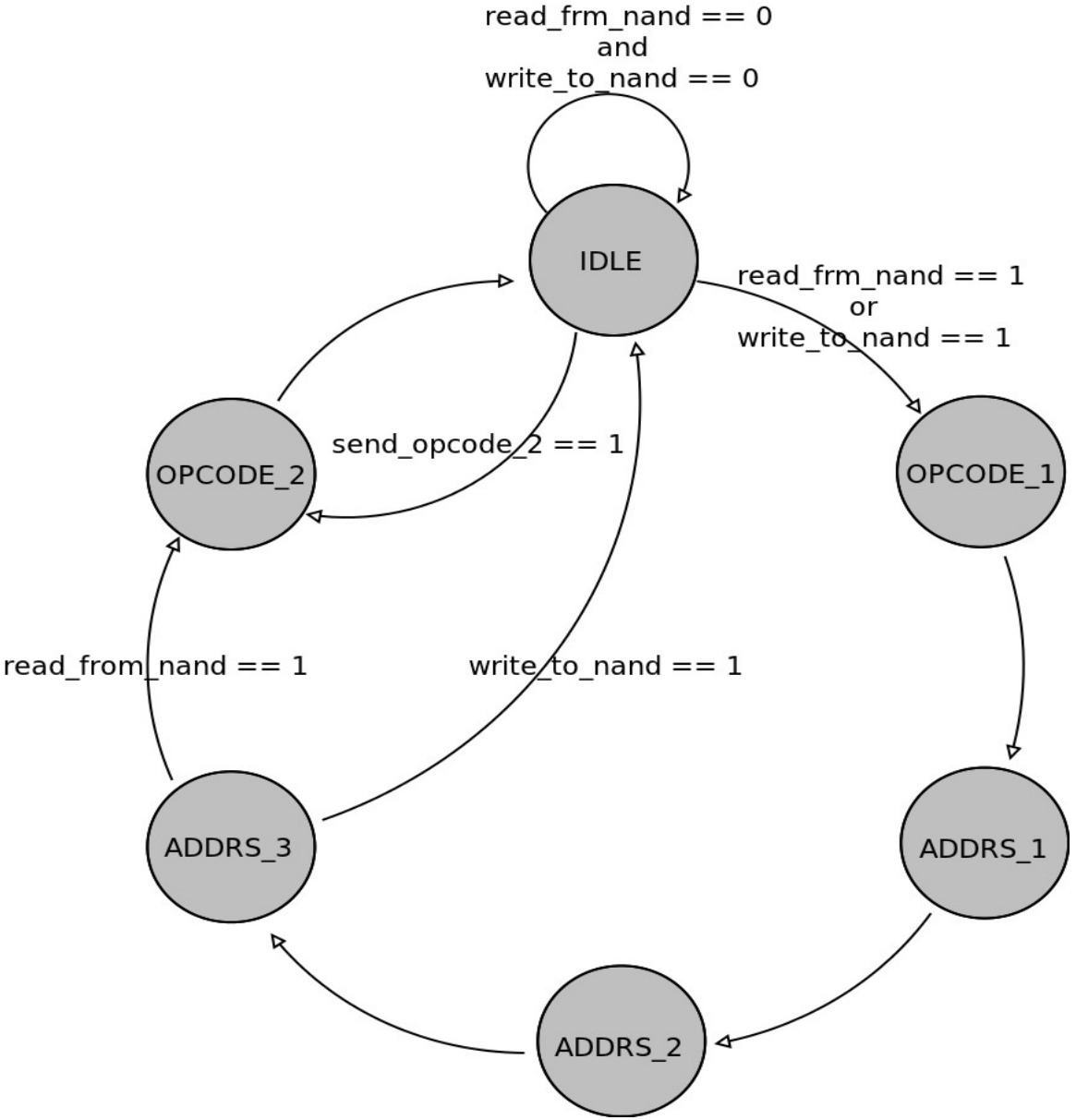


Figure: State Machine for accessing the Nand Flash Controller

REGISTER FILE

Start	End	Symbol	Description
00h	07h	CAP	Controller Capabilities
08h	0Bh	VS	Version
0Ch	0Fh	INTMS	Interrupt Mask Set
10h	13h	INTMC	Interrupt Mask Clear
14h	17h	CC	Controller Configuration
18h	1Bh	Reserved	Reserved
1Ch	1Fh	CSTS	Controller Status
20h	23h	Reserved	Reserved
24h	27h	AQA	Admin Queue Attributes
28h	2Fh	ASQ	Admin Submission Queue Base Address
30h	37h	ACQ	Admin Completion Queue Base Address
38h	EFFh	Reserved	Reserved
F00h	FFFh	Reserved	Command Set Specific
1000h	1003h	SQ0TDBL	Submission Queue 0 Tail Doorbell (Admin)
1000h + (1 * (4 << CAP.DSTRD))	1003h + (1 * (4 << CAP.DSTRD))	CQ0HDBL	Completion Queue 0 Head Doorbell (Admin)
1000h + (2 * (4 << CAP.DSTRD))	1003h + (2 * (4 << CAP.DSTRD))	SQ1TDBL	Submission Queue 1 Tail Doorbell
1000h + (3 * (4 << CAP.DSTRD))	1003h + (3 * (4 << CAP.DSTRD))	CQ1HDBL	Completion Queue 1 Head Doorbell
1000h + (4 * (4 << CAP.DSTRD))	1003h + (4 * (4 << CAP.DSTRD))	SQ2TDBL	Submission Queue 2 Tail Doorbell
1000h + (5 * (4 << CAP.DSTRD))	1003h + (5 * (4 << CAP.DSTRD))	CQ2HDBL	Completion Queue 2 Head Doorbell
…	…	…	…
1000h+ (2y * (4 << CAP.DSTRD))	1003h + (2y * (4 << CAP.DSTRD))	SQyTDBL	Submission Queue y Tail Doorbell
1000h + ((2y + 1) * (4 <<	1003h + ((2y + 1) * (4 <<	CQyHDBL	Completion Queue y Head Doorbell

Figure: Showing the list of Controller Registers (Memory Mapped Register File)

Error Information

Asynchronous Event Request and Get LOG Page
1. Host wrote doorbell of a queue that was not created
2. Doorbell value is out of range
3. Doorbell value is same as previous doorbell value
4. Host adds commands to a Full Submission Queue
5. Host removes from an Empty Completion Queue

Error Information

Command	Possible Error
Create I/O CQ	1. Invalid Q ID (value == 0 or > max value) 2. Max Queue size exceeded 3. Invlaid Interrupt Vector (value == 0 or >32)
Create I/O SQ	1. Invalid CQ ID 2. Max Queue size exceeded 3. Invlaid Q ID(value == 0 or > max value)
Delete I/O CQ	Invalid Q ID (value == 0 or Q ID not created)
Delete I/O SQ	Invalid Q ID (value == 0 or Q ID not created)
Abort Command	Abort command limit exceeded
Asynchronous Event Request	Limit exceeded
Get Features	Invalid Feature ID
Set Features	Invalid Feature ID
Get Log Page	Invalid Log Page ID
Common to all (in Log Page)	1. Invalid Opcode 2. PC set to zero when advertised about physical continuity

DESIGN CHALLENGES

DESIGN CHALLENGES

Design Challenge	Decision Taken / Design implemented
Multiple Requests to PCIe Controller from within the NVMexpress	PCIe Request State Machine was developed. Priorities are assigned in the order : write completion > completion tlp > write data > read data > read prp > read command
Sequence of operations for Command Arbitration and Fetch	Commands can be fetched and then arbitrated or first arbitrated and then fetched. Decision to arbitrate and then fetch was taken. This does not add additional buffers required to store the 64B commands.
Abort Command Limits	Every command has to be checked if it is to aborted or not. This comparision adds a lot of hardware and hence limits the number of outstanding commands to be aborted. A decision to support five outstanding commands is taken.

VERIFICATION

VERIFICATION SETUP

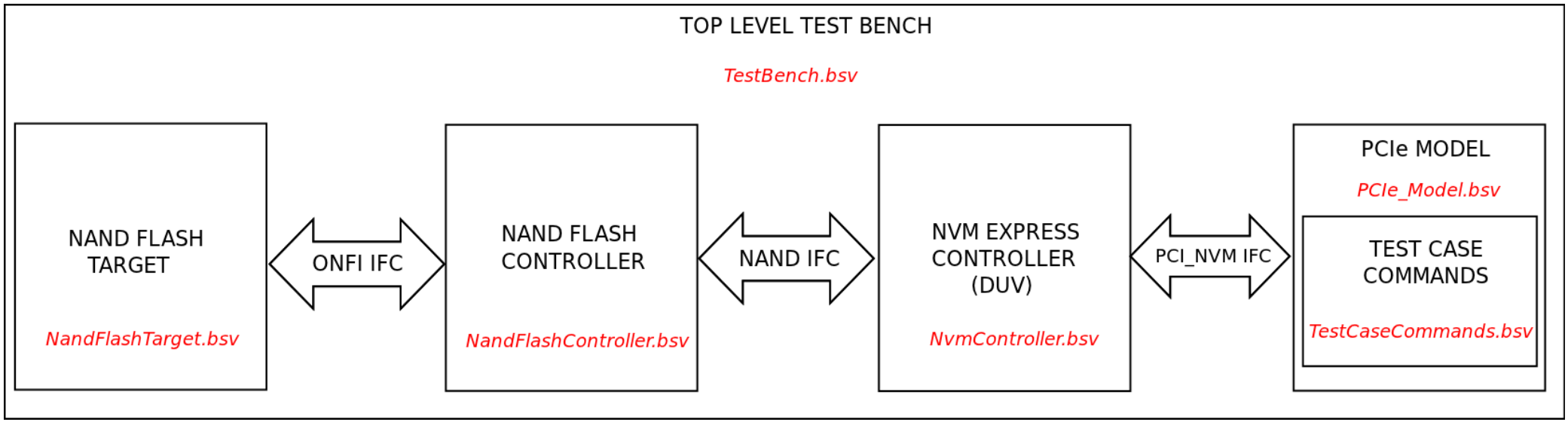


Figure: Verification Setup for Verfying NVMe Controller

VERIFICATION TEST CASES

Controller operation under normal conditions:

- Test cases were written for all commands
  - NVM Command set
  - Admin Command set
- Controller Shutdown
  - Normal shutdown
  - Abrupt shutdown
- Controller Initialization
- Read/Write to Register File.
- Arbitration Mechanism
- Interrupt Mechanism

VERIFICATION TEST CASES

Controller operation under special conditions:

- Abort command with number of outstanding commands greater than Abort Cmd Limit.
- Create I/O CQ with
  - Invalid Q ID
  - Qsize greater than Max Q Size specified
  - Invalid Interrupt Vector
- Create I/O SQ with
  - Invalid Q ID
  - Associated CQ does not exist
  - Max Q Size exceeded
- Delete I/O CQ with
  - Invalid Q ID
- Delete I/O SQ with
  - Invalid Q ID

Files Associated With the Project

- Main NVMe Controller :

  File Name : *NvmController.bsv*

  Amount of code : 3000 lines

- PCIe Model (For Verification) :

  File Name : *PCIe_Model.bsv*

  Amount of code : 1000 lines

- Test Case Commands (For Verification) :

  File Name : *TestCaseCommands.bsv*

  Amount of code : 500 lines

SYNTHESIS REPORT

SUMMARY OF SYNTHESIS REPORT

Feature	Summary
No.of Slice Regisers used	4773
No.of Slice LUTs used	6803
No. of LUT-FF Pairs used	2851
No.of Block RAMs used	13
Maximum Frequency of operation	204 MHz

CONCLUSION

- NVMe controller has been implemented with all the mandatory functionalities as per the NVMe 1.0c Specification.
- A generic PCIe controller is also developed inorder to interface any PCIe device with Xilinx PCIe IP core.
- With the implementation of the Flash Translation Layer, the Non Volatile Memory System will be complete.

REFERENCES

[1] J. S. Adam Wilen and R. Thornburg, Introduction to PCI Express: A Hardware and Software Developer’s Guide. Intel Press, 2003.

[2] Xilinx, Spartan-6 FPGA Integrated Endpoint Block for PCI Express, ug672 ed., January 18, 2012.

[3] Bluespec, Inc, Bluespec System Verilog Reference Guide, revision: 17 ed., 2012.

[4] R. S. Nikhil and K. Czeck, BSV by Example. Bluespec, Inc, 2010.

[5] “Pci express base specification,” April 29, 2002.

[6] A. Huffman, “Nvm express,” February 16, 2012.

[7] L. C. Rino Micheloni and A. Marelli, Inside NAND Flash Memories. Springer, 2010.