



FPGA-BASED PESTI MUX WITH INTEGRATED SNOOPER

Author(s):

Cristopher Mc Guinness Rodríguez,	Hewlett Packard Enterprise Company
Charles Hartman,	Hewlett Packard Enterprise Company
Edgar Campos Duarte,	Hewlett Packard Enterprise Company
Rachel Brownell,	Hewlett Packard Enterprise Company

Executive Summary

The Open Compute Project (OCP) aims to ensure interoperability among components from different vendors by defining standardized specifications that establish consistent design rules. One such specification, M-PESTI, provides a framework for efficient 1-pin communication. While the current M-PESTI specification offers foundational guidance and includes examples of Fan Out techniques, this white paper builds upon those methods. It proposes a new FPGA-based Fan Out controller to further optimize design approaches in this area.

By embedding the Multiplexer's (MUX) logic directly into the FPGA, which already hosts the Target/Snooper, external ICs and their associated control signals are removed. This simplifies the design, minimizes pin usage, and reduces board size. Furthermore, the FPGA-based MUX enables robust Break Event detection by monitoring connected lines, a feature not achievable with external ICs. The paper also defines mechanisms for sharing Break Event information with the Initiator, ensuring smooth and efficient communication.

Due to the internal logic of FPGAs, its pins are inherently associated with I/O buffers. This creates a limitation, as it forces the MUX to know the direction of the information beforehand so that it can arbitrate correctly. To mitigate this issue, a training mechanism is introduced to dynamically share virtual wire count and direction, enabling seamless communication between the Initiator and Targets. The paper also proposes enhancements to the M-PESTI specification, including new timers and a new payload format, to improve interoperability and efficiency.

This solution streamlines design, reduces costs, and improves system performance, aligning with OCP's goals of efficiency and scalability. By leveraging a protocol-aware Snooper and the proposed training mechanism, designers can achieve optimized communication architectures that resolve current limitations and enable innovation in scalable systems.

Table of Contents

1. Introduction	4
2. Compliance with Open Compute Project Tenets	5
2.1. Openness	5
2.2. Efficiency	5
2.3. Impact	5
2.4. Scale	5
3. An FPGA-Based MUX with an integrated Snooper	6
3.1. Why create an FPGA-Based MUX with an integrated Snooper?	6
3.2. How is the FPGA-Based MUX different from the options in the specification?	7
4. Specification changes needed and constraints.	10
4.1. Required Changes.	10
4.2. Optional changes.	12
5. FPGA-based MUX with Snooper solution.	13
5.1. Snooper Does Not Occupy a Position in the MUX.	14
5.2. At the start:	15
5.3. Break Release Detection and information sharing mechanism.	15
5.4. FPGA-based MUX functionality requirements.	16
5.5. Training Mechanism:	17
6. Conclusion	21
7. Glossary	22
8. References	22
9. License - Creative Commons	22
10. About Open Compute Foundation	23

1. Introduction

Modern computing systems are increasingly reliant on efficient communication between key hardware components to ensure optimal performance and functionality. To improve communication between hardware components, the M-PESTI protocol was introduced by the Open Compute Project (OCP), offering a streamlined solution for communication between Initiator and Target devices. The protocol simplifies the communication process by enabling interaction with each Target device through a single pin, significantly reducing the number of pins required for data exchange.

Additionally, the M-PESTI specification covers Fan-Out Techniques which define controller modules that allow the Initiator to communicate with multiple Targets using only one pin. These Fan-Out Methods leverage integrated circuits, such as FET multiplexers (FET MUX) or switches, to manage communication pathways, and rely on programmable logic devices, such as Field Programmable Gate Arrays (FPGAs), to control these integrated circuits. Regardless, the M-PESTI specification also leaves room for a third alternative, an approach that integrates both the MUX and Snooper functionalities directly within an FPGA, which we explore in this paper.

This white paper examines the M-PESTI protocol and its implementation. Building on this foundation, we propose an integrated solution, an FPGA-based MUX with a built-in Snooper, offering a streamlined and effective method for communication between the Initiator and the Targets. We aim to unlock the full potential of M-PESTI and contribute to the advancement of modern computing system design.

2. Compliance with Open Compute Project Tenets

This white paper complies with the OCP tenets by implementing modifications to the design that expand its functionality and provide third parties with more flexibility. These changes will have a positive impact on the industry by creating more opportunities for increased efficiency while aiming to use natural resources responsibly and considering the different ways the design may be implemented by third parties.

2.1. Openness

The OCP M-PESTI specification provides foundational examples for fan-out, which we build upon by embedding the MUX within an FPGA. This solution enhances functionality while addressing bus direction and protocol awareness, enabling flexible and efficient communication. By sharing this implementation, we empower third parties to adapt and customize designs, fostering openness and collaboration.

2.2. Efficiency

Expanding on the M-PESTI specification, we integrate the MUX functionality directly into the FPGA, eliminating the need for external components. This approach improves efficiency by reducing PCB usage, lowering material costs, and simplifying design. By leveraging programmable devices, the solution optimizes resource utilization and enhances cost-effectiveness.

2.3. Impact

Integrating the Fan-Out MUX into an FPGA makes M-PESTI more accessible, supporting more devices with minimal hardware changes. Existing systems can adapt to new specifications by updating FPGA code, driving impactful advancements through flexible and efficient designs.

2.4. Scale

The current M-PESTI specification supports switching between 8 lines, but future designs may require greater flexibility. By incorporating the MUX into an FPGA, the ability to scale and handle a larger number of lines is enabled, expanding on what the specification provides.

3. An FPGA-Based MUX with an integrated Snooper

The integration of MUX and Snooper functionalities within a single FPGA represents a significant advancement in the implementation of the M-PESTI protocol.

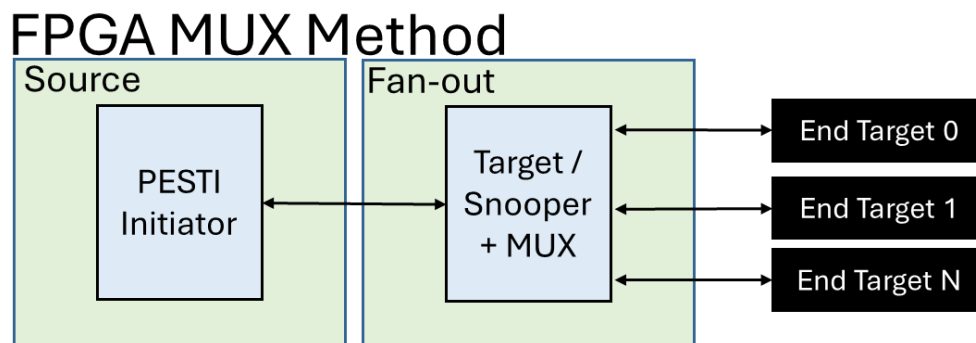


Figure 1. M-PESTI Block Diagram with an FPGA-based MUX with an integrated Snooper.

3.1. Why create an FPGA-Based MUX with an integrated Snooper?

There are compelling reasons that demonstrate its significant benefits and relevance.

3.1.1. Optimizing Pin Usage and Reducing IO Costs.

One of the primary advantages lies in minimizing the number of pins required for communication. In modern FPGA designs, input/output (IO) pins are a costly resource, and reducing their usage is critical for both cost-efficiency and board design optimization. The M-PESTI protocol is inherently designed to enable communication with multiple devices while minimizing pin requirements. By integrating the MUX functionality directly into the FPGA, the number of pins required can be reduced from $2N+1$ to $N+1$ (using the Switch Fan Out Method example as a reference). This improvement eliminates the need for external control signals for FET MUX or Switch components, as the logic is inherently built into the FPGA. Such a reduction in pin count aligns more closely with the protocol's goal of efficient communication using a single wire to connect the Initiator and Target, while also avoiding the need for larger or more pin-dense (and consequently more expensive) FPGA components.

3.1.2. Reducing Dependence on Additional Hardware.

Another key advantage is the reduction in board complexity and cost associated with external integrated circuits (ICs). The inclusion of standalone MUX or Switch ICs increases both the physical footprint of the board and the overall cost due to the price of the components

themselves. By embedding the MUX logic directly into the FPGA these external ICs can be eliminated entirely. This integration reduces the overall board size and simplifies the design. While there is a potential trade-off if the added logic consumes significant FPGA resources, necessitating a larger or higher-spec device, this scenario may not occur in most implementations. Assuming resource consumption is manageable, the benefits of reducing hardware costs and board complexity far outweigh this potential downside.

3.2. How is the FPGA-Based MUX different from the options in the specification?

Several key factors differentiate an FPGA-based MUX from the Fan-Out Controllers shown in the specification available in the specification that rely on FET multiplexers or Switches. These differences arise primarily from the distinct behaviors of the hardware components and how they handle data flow in the context of the M-PESTI protocol.

3.2.1. Determine Data Direction.

One of the core challenges of implementing an FPGA-based MUX lies in handling the bidirectional nature of the M-PESTI protocol, which operates over a single wire. As outlined shown in the following diagram, FET multiplexers act as low-impedance pathways that allow data to pass freely in either direction. This simplicity means that FET MUX implementations do not need to explicitly manage the direction of data flow, as wires are simply interconnected, and communication occurs seamlessly.

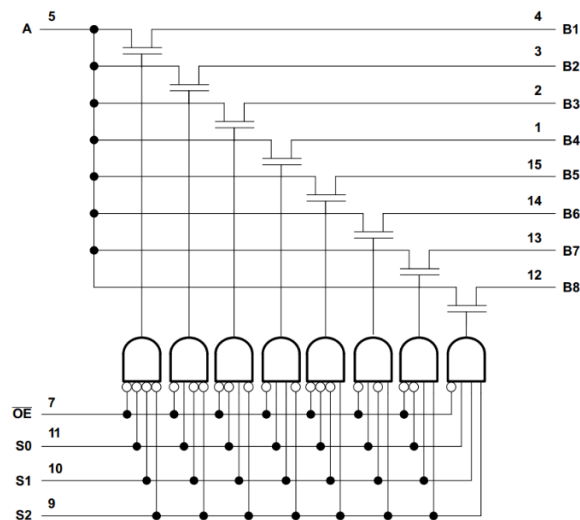


Figure 2. Functional Diagram of the FET MUX SN74CBT3251 [1].

Unlike FET multiplexers, which inherently support bidirectional communication, FPGA-based designs require explicit management of data direction due to their Input/Output Buffers, as shown in the following diagram, require they require the data direction to be explicitly set.

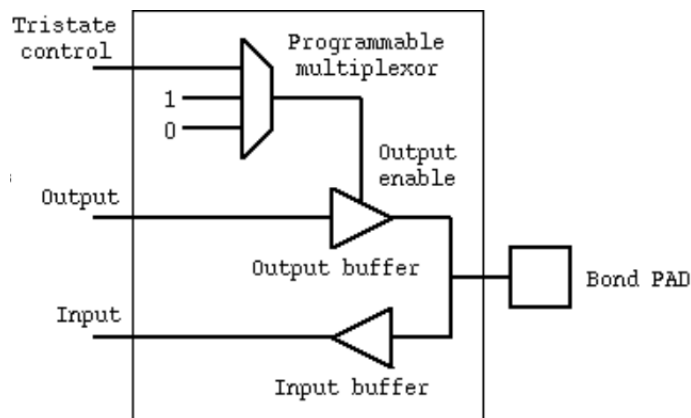


Figure 3. FPGA Input and Output Buffers [2].

Consequently, the module within the FPGA must have the necessary information to determine whether data is flowing from the Initiator to the Target or vice versa. This introduces a critical design challenge that must be resolved to ensure proper functionality. To better understand this issue, it is helpful to examine all the communication scenarios.

Managing Data Direction in the Four Scenarios Covered in the Specification:

The M-PESTI protocol defines four primary operational scenarios: Discovery Phase, Active Phase, MUX Switch Control, and Broadcast. Each of these scenarios has unique requirements for managing data direction:

1. **Discovery Phase:** In this phase, the direction of data flow is predetermined, simplifying the communication process. Since only 1 byte is sent from the Initiator to the Target and the payload size could be obtained by decoding one of the bytes of the payload.
2. **MUX Switch Control:** Since the number of bytes in each direction is fixed, the direction can be inferred.
3. **Broadcast:** Data flows from the Initiator to all connected Targets, making directionality easy to resolve.
4. **Active Phase:** This phase introduces the most complexity. During Active Phase communication, the number of bytes exchanged between the Initiator and each Target is

not fixed and may only be determined after a payload has been transmitted to or received from a Target. The FPGA-based mux with a Snooper integrated must dynamically resolve the number of bytes that need to travel in each direction upon detecting a command with a code of 0x1. Specifically, the module must know the number of Virtual Wires Out to properly arbitrate the communication and ensure accurate data transfer.

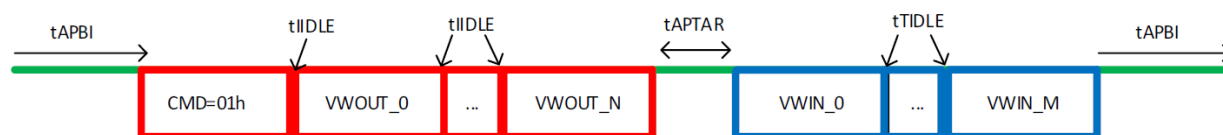


Figure 4. Multi-byte Virtual Wire Exchange [3].

3.2.2. Break Detection.

One aspect of the current specification that remains open-ended is the mechanism for Break Event detection in the context of the examples of the Fan-Out Methods in the specification. A Break Event signals that a Target is ready to receive a command, and detecting these events reliably is critical for seamless communication between the Initiator and connected Targets. While implementations where both the Initiator and round-robin servicing logic reside within the same FPGA make Break Event detection straightforward, this process becomes significantly more challenging when implementing fan-out module.

The complexity arises from the fact that a Break Event can occur at any moment, since the Initiator can only be connected to one Target at a time. As a result, there is no guarantee that every Break Event will be detected when it occurs without additional logic and connections, since the Initiator is not actively monitoring all Targets simultaneously. This limitation introduces potential gaps in communication, as Break Events from certain Targets may go unnoticed.

To address this challenge comprehensively, this document will propose a solution for reliably detecting and communicating Break Events to the Initiator, leveraging the distinct advantages of an FPGA-based MUX design.

4. Specification changes needed and constraints.

Before describing the implementation details, it is important to outline some required changes to the specification to ensure its functionality. Additionally, a few suggestions could be included to further optimize and improve system performance.

4.1. Required Changes.

4.1.1. Additional Timeouts:

The current specification provides a timeout mechanism for the Snooper to detect a stuck bus following a downstream channel select operation. This is defined by the parameter tMSC_WDT:

Symbol	Parameter	Min	Max	Units	Comments
tMSC_WDT	Mux Switch Control Watchdog Timeout	100	-	ms	Time for a snooper to detect a stuck bus following a downstream channel select operation

Figure 5. Current Watchdog Timer available in the M-PESTI specification [3].

Nevertheless, depending on the implementation needs, leveraging the Watch Dog Timer (WDT) for both the Discovery Phase and Active Phase may introduce potential latency concerns. Waiting 100ms for the MUX position to change during every Active Phase exchange could result in delays that prevent the latency requirements of individual Targets from being met, especially in critical scenarios such as a fault notification. For example, if a Target needs to convey fault information to the Initiator within a specific timeframe, such delays could compromise the system's responsiveness.

To address this issue, it is imperative to introduce two distinct Watch Dog Timers that vary based on the phase the Target is operating in. These timers are calculated to account for different payload sizes and phase-specific timing requirements:

Discovery Phase WDT: Calculated based on a 2048-byte payload and the 10ms tDPTAR time maximum, ensuring timely communication during payload exchanges.

Active Phase WDT: Calculated based on the tAPRTO parameter, covering timing needs during arbitration and processing phases.

Table 1. Additional Watchdog Timers.

Symbol	Parameter	Min	Max	Units	Comments
tMSC_DPWDT	Mux Switch Control Discovery Phase Watchdog Timeout	110	-	ms	Timer for a snoopers to detect a stuck bus in the Discovery Phase following a downstream channel select operation.
tMSC_APWDT	Mux Switch Control Active Phase Watchdog Timeout	250	-	us	Timer for a snoopers to detect a stuck bus in the Active Phase following a downstream channel select operation.

4.1.2. Additional information shared through the MSC_STAT_VAL:

The current MSC_STAT_VAL provides limited information, which restricts Snooper's ability to share data with the Initiator.

MSC_STAT_VAL (PESTI Snooper Target to Initiator)

7	6	5	4	3	2	1	0
RSVD							NOT_ACK

Figure 6. Current MSC_STAT_VAL available in the M-PESTI specification [3].

Since there is a need for additional information to be shared from the Snooper to the Initiator, it is important to recall that prior to MSC_STAT_VAL, the MSC_CTRL_VAL is sent by the Initiator. The MSC_CTRL_VAL serves to indicate the position to which the MUX is intended to be switched. By defining additional fields within the MSC_STAT_VAL, the Snooper can relay detailed information about the specific Target that the Initiator has requested to switch to. These newly defined fields serve the following purposes:

1. **HELD LOW:** This field samples the value of the Target line and is asserted when the line is held low.

The purpose of this bit is to prevent the Initiator from attempting communication with a Target that is currently holding the line low. This condition could indicate that the Target is not ready to communicate or a simple presence from a non-PESTI capable device. While the WDT could eventually release the Initiator from such a lock, it would incur a latency penalty. By sharing this information, the system can avoid unnecessary delays and maintain efficient communication.

2. **BREAK EVENT DET:** This field is asserted when a Break Event (e.g., a break release) is detected on the line.

The purpose of this bit is to inform the Initiator that a Break Event has occurred, which could signify various states depending on the context. At the start, sharing this information with the Initiator is critical to ensure it knows which Targets are available to begin communication. It is important to clarify that once the information is shared and then, the Initiator issues a command to that Target the bit is cleared until a new Break Event is detected.

Table 2. Suggested new format for the MSC_STAT_VAL.

Bit	7	6	5	4	3	2	1	0
FIELD	RSVD					BREAK_EVENT_DET	HELD_LOW	NOT_ACK

There are other uses that are given after the initial Break Event detection. Since a Target may assert a Break Event when resetting, this bit is leveraged to share that information with the Initiator.

4.2 Optional changes.

4.2.1 Support for more than 8 targets:

The current MSC_CTRL_VAL allows switching between 8 positions:

MSC_CTRL_VAL (Initiator to PESTI Snooper Target) Data Byte Value = 02h

7	6	5	4	3	2	1	0
RSVD	RSVD	RSVD	DS_ALL_CH_SEL	DS_CH_SEL[2:0]		ENABLE	

Figure 7. Current MSC_CTRL_VAL available in the M-PESTI specification [3].

The current command restricts the fan-out to a maximum of 8 Targets, which limits the scalability of the Fan-Out Methods. Alternatively, additional bits could be allocated to increase the number of positions, allowing the Initiator to communicate with a larger set of Targets. Sharing this information with the Initiator is critical to ensure it can properly address and interact with all available Targets in the system.

Table 3. Suggested new format for the MSC_CTRL_VAL.

Bit	7	6	5	4	3	2	1	0
FIELD	RSVD	DS_CH_SEL[4:3]		DS_ALL_CH_SEL	DS_CH_SEL[2:0]		ENABLE	

4.2.2 Proposed Payload Format for the Snooper:

Several pieces of information could be valuable when exchanging the payload from the Snooper to the Initiator. These include the number of PESTI wires available, which indicates how many devices can be connected to the FPGA-based MUX, FET MUX or Switch, and a configuration bit to differentiate between implementations using the FET MUX/Switch method and those using the FPGA-based method. Sharing this information with the Initiator is critical to ensure it has full visibility into the system's configuration and connectivity, enabling efficient communication with all available devices.

Table 4. Suggested Payload for the Snooper.

Byte/Bit	7	6	5	4	3	2	1	0
00h	Payload Format Version = 02h							
01h	Discovery Payload Size							
02h	Vendor ID[15:8]							
03h	Vendor ID[7:0]							
04h	Device Class = 0xFE							
05h	Device ID[15:8]							
06h	Device ID[7:0]							
07h	Number of available PESTI Wires[4:0]					Configuration (0=FET MUX/Switch Method, 1=FPGA- Based MUX)	PEC SUPPORT	VENDOR ID FORMAT
08h to N	Optional Vendor Specific Region (Varies) + **Padding (0 – 7 Bytes)							
Nh	**Checksum							

5. FPGA-based MUX with Snooper solution.

The challenges outlined earlier in this document, including the snooper's ability to discern bus direction and transaction state, can be addressed through various approaches. In this paper we

opted to illustrate one solution. We worked through several solutions and distilled down the minimum required specification changes. This paper introduces a mechanism for enabling the FPGA snooper to “learn” when to transition between Virtual Wire output and input states. This method, referred to as the "Training Mechanism", leverages the snooper’s primitive protocol awareness to count Virtual Wire output bytes and to adapt its behavior dynamically.

FPGA MUX Method

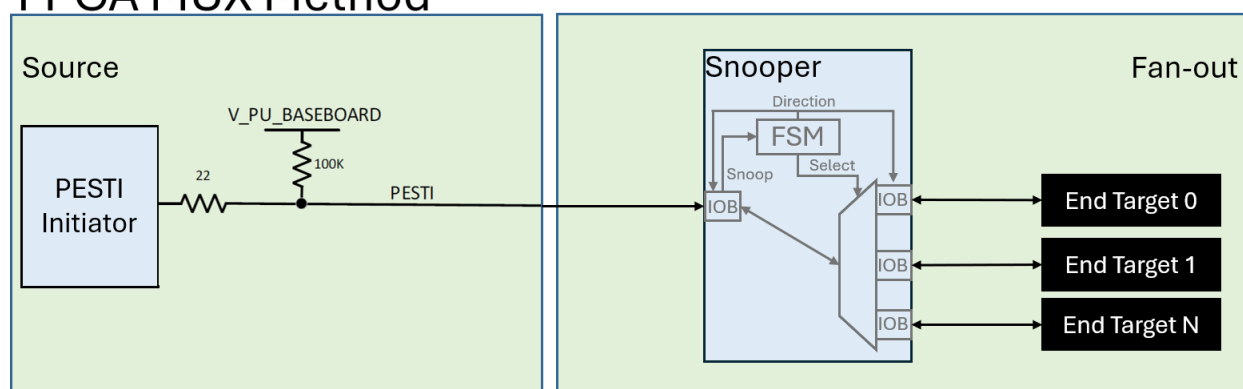


Figure 8. M-PESTI Functional Diagram with an FPGA-based MUX with an integrated Snooper.

The snooper is designed with sufficient comprehension of the bus protocol to interpret bus commands. It utilizes byte counts and/or timeouts to identify all fixed-length bus cycles, ensuring predictable operation in scenarios where transaction lengths are predefined. However, the primary variable that the snooper must “learn” during operation is the number of virtual wires expected by each target. This adaptive capability is critical for supporting dynamic and variable bus transactions while maintaining efficient and accurate operation.

Prior to diving into the training sequence, it is important to cover some additional details.

5.1. Snooper Does Not Occupy a Position in the MUX.

The M-PESTI specification suggests that the snooper should occupy Position 0 of the multiplexer on its Fan-Out Techniques examples, enabling the initiator to communicate exclusively with the snooper. However, the specification does not explicitly mandate this arrangement. Additionally, the specification defines bit 0 of MSC_CTRL_VAL as ENABLE. When ENABLE is set to 0, the multiplexer or switch is disabled, allowing the initiator to

communicate solely with the snoopers while all downstream multiplexer or switch channels remain disabled.

If the number of downstream targets is limited to eight (as defined by DS_CH_SEL[2:0]), the snoopers occupying Position 0 reduces the number of usable target positions to seven. By contrast, if the snoopers does not occupy a dedicated position within the multiplexer, all eight downstream target positions remain available, which could be advantageous in systems requiring a higher number of targets.

For the purposes of this white paper, the proposed solution will leverage the ENABLE bit to control snoopers operation without requiring it to occupy Position 0 of the multiplexer. This approach preserves the full complement of eight downstream target positions, optimizing system functionality while maintaining compliance with the M-PESTI specification.

5.2. At the start:

The Snooper is expected to hold the line low for more than 50 microseconds, asserting the Break Event. Once ready to receive, it releases the line. Following this, the Initiator, regardless of DPEN or APEN states, will issue command 0x0 and decode the payload. This step allows the Initiator to determine whether the PESTI line is connected to a Fan-Out Controller or directly to a Target. If the payload indicates a Fan-Out Controller, the relevant values are stored, and the remaining payload is discarded. If the payload indicates a direct connection to a Target, the payload must be discarded after reception, and the Initiator must wait for the DPEN signal to reissue command 0x0.

5.3. Break Release Detection and information sharing mechanism.

The Fan Out Controller communicates that a Break Event was detected to the Initiator using the MSC_STAT_VAL byte. This byte is sent after the Initiator requests a channel change via the MSC_CTRL_VAL byte. Once the channel is selected, the FPGA-based MUX relays the current break status to the Initiator, ensuring it is informed of Target states and the bit is cleared. The MSC_STAT_VAL byte format can be structured as defined in Table 2. Suggested new format for the MSC_STAT_VAL.

5.4. FPGA-based MUX functionality requirements.

The following assumptions underpin the design proposal:

1. **Payload Exchange Capability:** The snoopers are capable of exchanging payload data, ensuring it can participate effectively in communication and protocol operations. The payload can be requested at any time if the Snooper is the only one connected to the Initiator.
2. **Payload Version Identification:** The snoopers utilize the payload to distinguish themselves as a multiplexer rather than an End Target. This differentiation is critical for ensuring proper identification and operational functionality within the system.
3. **Sending Commands:** Before sending any command—except for command 0x0 at the start—the Initiator must issue command 0x2 along with MSC_CTRL_VAL to specify the target it wants to communicate with. This prevents the Initiator from sending multiple consecutive commands to the same target. Instead, after sending a command, if another is required, the Initiator must reissue command 0x2, reselect the same target, and then send the next command.
4. **Abort Mechanism:** During a transaction, either the Initiator or a Target may assert a Break Event to abort the exchange. Upon detecting the Break Event, both parties cease communication. Concurrently, the Snooper identifies the Break Event, disconnects the Target from the Initiator, and awaits command 0x2 to switch to a different Target.
5. **Virtual Wire Byte Count Storage:** The snoopers are designed to store the byte count for Virtual Wire output associated with each Target. This capability enables the snoopers to manage and track data flow for individual Targets efficiently.
6. **Training for Virtual Wire Detection:** Prior to issuing any command 0x1 to a Target, the snoopers must undergo a "training" phase to share the number of virtual wires expected by the Target to the Snooper. This step is essential because the Snooper must dynamically adjust the bus direction between Virtual Wire output and input states. The specification defines the minimum Active Phase Turn Around Time (tAPTAR) as 0.1 microseconds. However, achieving the minimum turnaround time presents a significant challenge for systems operating at a clock frequency of 4 microseconds, necessitating careful design considerations and timing optimizations.

These assumptions establish the foundational requirements for the proposed design and highlight key operational constraints that must be addressed to ensure compliance with the specification and effective system performance.

5.5. Training Mechanism:

The training sequence, managed by the initiator's hardware (HW), is a critical step that occurs early in the boot process. Once training is complete, the system incurs no additional latency or bandwidth penalties. The most critical timing challenge during this process arises in the Active Phase, where the specification allows only 0.1 microseconds to switch the bus direction between Virtual Wire out and Virtual Wire in states. To address this, the proposed approach involves "training" the snoopers to determine the number of virtual wire bytes expected for each target.

5.5.1. Training from the Initiator's perspective:

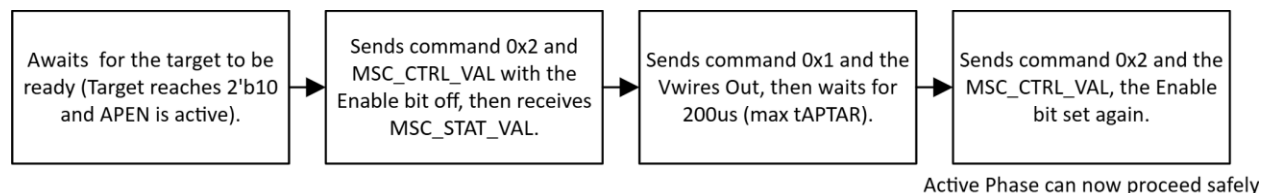


Figure 9. Flow diagram of the training sequence for one target from the Initiator's perspective.

Additional comments:

1. The maximum idle time between transactions (t_{IDLE}) is 1000 ns (1 microsecond), and the minimum Active Phase Bus Idle time (t_{APBI}) is 10 microseconds.
2. The initiator is guaranteed to observe at least one Virtual Wire In byte. Once the initiator detects one UART clock period (4 microseconds) of the MPESTI line floating, it can confidently conclude that the Virtual Wire In transaction has completed.

Detailed Flow of events:

1. Initial state:
 - a. At the outset, only the FPGA Snooper is connected to the Initiator. No commands have been issued, and therefore no targets are connected. The Initiator waits to see a Break Event.

- b. Once the Break Event is detected by the Initiator, it issues its first command, CMD 0x00. Since no CMD 0x02 commands have preceded it, the Snooper responds with its own payload, confirming its identity as a Snooper. Optionally, the payload can also include additional information, such as the number of downstream targets supported. See Table 4. Suggested Payload for the Snooper. for the suggested payload structure for the Snooper.
- 2. Start state:
 - a. The Initiator operates as usual, waiting for each Target's payload to be successfully received. Key indicators such as ASTAT==1, APEN==1 and DSTAT == 2'b10 signal readiness (specific conditions may vary).
 - b. Once the readiness event occurs, the training sequence for the respective Target is armed.
- 3. Metadata: MPESTI provides two methods for the Initiator to determine the number of virtual wires required for each Target:
 - a. Method A: During discovery, the Field Replaceable Unit (FRU) sets the number of virtual wires in a register, which the Initiator then reads.
 - b. Method B: Targets using Payload Version 0 include the number of virtual wires in a specific payload field, allowing the Initiator to extract this information directly.
- 4. Training Routine:
 - a. Disable Bus Access: The Initiator de-asserts the ENABLE bit (bit 0) in the MSC_CTRL_VAL register to disable the multiplexer. This ensures that only the snooper remains connected to the Initiator, while all downstream targets are disconnected.
 - b. Disable Error Detection: To prevent unnecessary error flags, the APERR mechanism is temporarily disabled during the training phase.
 - c. Initiate Virtual Wire Exchange: With the ENABLE bit off, the Initiator sends Virtual Wire Out bytes to the Snooper as if it was sending it to the target selected through the MSC_CTRL_VAL.
 - d. Wait for the timeout: The Snooper will not reply after receiving the Virtual Wire Out bytes. Therefore, the Initiator waits for 200 microseconds (tAPTAR) to reissue command 0x2.

- e. Complete Training: The training sequence is performed when a new target is ready to begin its Active Phase, ensuring the snooper has accurate data for each target intending to exchange Virtual Wires.
5. Post-Training Operation:
 - a. Once training is complete, the Initiator re-asserts the ENABLE bit (bit 0) in the MSC_CTRL_VAL register, restoring normal bus access for all Targets.
 - b. All subsequent transactions follow standard active phase procedures (CMD2, CMD1). The Snooper dynamically adjusts the bus direction immediately after the stop bit of the final VWOUT byte, based on the byte count determined during training.
 6. Retraining Requirement:
 - a. If a Target undergoes a firmware update or resets, the Snooper must be retrained. The Initiator will repeat the training sequence to ensure the snooper is updated with the correct Virtual Wire byte counts for the affected Target.

5.5.2. Training from the Snooper's perspective.

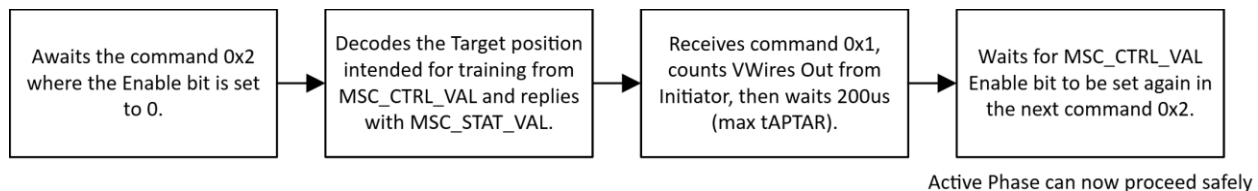


Figure 10. Flow diagram of the training sequence for one target from the Snooper's perspective.

Detailed Flow of Operations:

1. Initial State:
 - a. At the outset, the Snooper maintains holds the line low for more than 50 microseconds. Once the Snooper is ready to receive, the line is released, signaling the Snooper's readiness to receive commands and enabling the Initiator to begin communication.
2. Idle State:
 - a. The Snooper remains idle while waiting for the Initiator's command 0x0 to request the Payload. The format of the expected Payload is defined in Table 4. Suggested Payload for the Snooper.

3. Start State:

- a. **Payload Transmission:** Upon receiving command 0x0, the Snooper transmits its Payload, which includes configuration and identification data. This ensures that the Initiator detects the presence of an FPGA-based MUX with an integrated Snooper, instead of a standalone Target.
- b. **Ready for Position Switching:** After completing the payload transmission, the Snooper becomes ready to switch to any target position and either exchange payloads or initiate training to enable the Active Phase.

4. Training Routine:

- a. **Training Activation:** At any time, the Snooper may receive the MSC_CTRL_VAL signal. If bit 0 is set to LOW (0), exclusive communication between the Snooper and the Initiator is established for training purposes.
- b. **Target Identification:** The Snooper interprets the MSC_CTRL_VAL signal to determine which specific Target the Initiator will be referring to during the training sequence.
- c. **Snooper Reply:** The Snooper responds with MSC_STAT_VAL, confirming its readiness for the training process, and prepares to receive command 0x1.
- d. **Byte Counting:** Following the receipt of command 0x1, the Snooper begins counting the number of bytes to identify the number of Virtual Wire Out bytes associated with the specified Target. Once completed, this data is stored internally by the Snooper.
- e. **Transmission Delay:** After successfully receiving the Virtual Wire Out bytes, the Snooper waits for 200 microseconds (tAPTAR).
- f. **Repetition for Multiple Targets:** The Initiator trains the Snooper by repeating the training sequence for each specific Target, ensuring the Snooper is properly configured for Virtual Wire exchanges with all available Targets.
- g. **Transition to Virtual Wire Exchange:** Once the Snooper has been successfully trained, Virtual Wire exchanges can commence.

5. Post-Training State:

- a. **Switching Positions:** Before initiating subsequent exchanges, the Snooper waits for command 0x2 to switch to a different position or to remain in the current position with the Enable bit set to 1, signaling the start of Virtual Wire exchange.
 - b. **Command Readiness:** The Snooper becomes fully prepared to handle any incoming commands appropriately.
- 6. **Retraining Requirement:**
 - a. **Retraining Process:** If retraining is required, the Snooper and Initiator establish communication again. During this process, MSC_CTRL_VAL must have bit 0 set to LOW (0) in the next transaction to reinitialize the training procedure and then step 3 is repeated.

6. Conclusion

The integration of M-PESTI MUX functionality within a remote FPGA represents a compelling opportunity to enhance system efficiency and flexibility. By leveraging the programmable nature of an FPGA—already a necessary component on the remote PCB—to serve as both the snooper and the MUX, significant advantages can be realized, including reduced board space, optimized FPGA pin usage, and lower overall system costs. This approach exemplifies the power of embedding functionality directly into existing hardware to streamline design and minimize complexity.

While implementing an FPGA-based MUX in a multi-master system does pose challenges—particularly with regard to managing bidirectional data flow—the solutions outlined in this white paper demonstrate that these obstacles are not only surmountable but can be addressed with innovative techniques. By providing the Snooper with guidance on bus data direction and leveraging the bounds established by the M-PESTI protocol, the system can infer and adapt to data flow requirements effectively. For more complex scenarios, such as Virtual Wire transactions and asynchronous break requests, the Snooper can be trained to learn the unique behaviors and requirements of each target, enabling robust and reliable operation.

Although the inclusion of an FPGA-based MUX is not a strict requirement of the M-PESTI protocol, its adoption offers a valuable alternative that goes beyond the traditional FET MUX or

switch-based solutions. By embedding the MUX within the FPGA, designers gain greater flexibility, reduced hardware dependency, and a more compact and cost-effective system architecture. This approach underscores the versatility of the M-PESTI protocol and opens the door to more adaptable and efficient implementations, empowering users to achieve higher levels of innovation and performance in their designs.

7. Glossary

FPGA: A field-programmable gate array (FPGA) is a configurable I/C.

M-PESTI: Modular-Peripheral Sideband Tunneling Interface (M-PESTI).

8. References

- [1] Texas Instruments, December 2015. [Online]. Available:
<https://www.ti.com/lit/ds/symlink/sn74cbt3251.pdf?ts=1753156836935>. [Accessed 14 August 2025].
- [2] D. Greaves, University of Cambridge, 2017. [Online]. Available:
<https://www.cl.cam.ac.uk/teaching/1617/SysOnChip/materials.d/sp3-partition/zhp7e8c2a2a6.html>. [Accessed 14 August 2025].
- [3] Jeff Kennedy et al., Open Compute Project, 3 March 2024. [Online]. Available:
<https://drive.google.com/file/d/1QQ3GfDzTk5SxdgS-O8sTWjwuyzBsCmw2/view>. [Accessed 25 August 2025].

9. License - Creative Commons

This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

International License found here: <https://creativecommons.org/licenses/by-sa/4.0/>

10. About Open Compute Foundation

The Open Compute Project (OCP) is a collaborative Community of hyperscale data center operators, telecom, colocation providers and enterprise IT users, working with the product and solution vendor ecosystem to develop open innovations deployable from the cloud to the edge. The OCP Foundation is responsible for fostering and serving the OCP Community to meet the market and shape the future, taking hyperscale-led innovations to everyone. Meeting the market is accomplished through addressing challenging market obstacles with open specifications, designs and emerging market programs that showcase OCP-recognized IT equipment and data center facility best practices. Shaping the future includes investing in strategic initiatives and programs that prepare the IT ecosystem for major technology changes, such as AI & ML, optics, advanced cooling techniques, composable memory and silicon. OCP Community-developed open innovations strive to benefit all, optimized through the lens of impact, efficiency, scale and sustainability. Learn more at www.opencompute.org.