Chapter 6: Packet Assemble Block (PAB) Engine

This chapter uses the following sections to describe the Packet Assembly Block (PAB) engine, with its flexible packet reassembly and editing capabilities.

- Overview describing PAB engine usage.
- Features of the engine.
- Functional Description.
- Engine Configuration – Where in the Axxia Software Environment to find and configure PAB attributes, along with a description of the attributes found in the Axxia Software Environment (ASE).
- Input Task Parameters – Parameters passed to the PAB engine that define processing for a task.
- Output Task Parameters – Parameters passed to downstream engines and task error flags.

Overview

The Packet Assembly Block (PAB) is an accelerator engine used to flexibly reassemble and transmit multiple segmented packets. The PAB engine supports commands that perform these key tasks.

- Assembly – Control the placement of the segments when reassembling the packet. With this flexible configuration, you can perform segment-level packet editing.
- Transmit – Select whether to transmit the entire packet, part of a reassembled packet, or transmit the packet and create a copy.
- Discard – Decide what part of the packet reassembly to discard. This frees memory for new reassembly tasks.
- Troubleshoot – Obtain values about packet reassembly priorities, memory usage, and state.

The PAB accelerator engine enables reassembly of multiple segments into the same reassembly buffer based on different requirements. The segment put into the reassembly buffer can be programmed with the required packet size and the starting position located in the shared buffer. The reassembly buffer holds up to 64 KB.

The PAB engine can simultaneously reassemble up to 2^{24} packets. It uses the cache and memory to hold reassembly context data during the assembly process. It also uses the cache to store a dynamic, reassembly context packet state in a table called the Connection Database (CDB).

The PAB engine communicates with the Memory Management Engine (MME) to allocate, deallocate, and manage dynamic memory blocks. The PAB engine communicates with the Timer Manager (TMGR) to request timer starts and to receive time-out events for packet reassemblies that require time-sensitive transmission. The PAB transmits reassembled packets to other Axxia accelerator engines, for example, the Modular Packet Processor (MPP), where the transmitted task contains access to the packet reassembly. The host processor initializes the PAB engine using the Axxia Configuration Ring.

Features

The PAB engine has the following features.

- Operates as a general holding buffer that can assemble, transmit, retransmit, and delete packets from incoming data streams.
  - Supports inserting or extracting segments from anywhere in the same single assembly buffer.
  - Supports gaps in the same single buffer.
  - Specifies to the bit level how to support protocols such as the High Speed Medium Access protocol (MAC-hs).

- Supports the following traditional reassembly functions.
  - IP defragmentation
  - AAL5
  - AAL2
  - Service Specific Segmentation and Reassembly (SSSAR)
- Supports the following generalized holding buffer and sliding window protocols for the transmit and retransmit buffer functionality.
  - TCP origination and termination
  - MAC-d protocol
- Buffers and schedules packet data based on events, rather than rates or weights, especially when a varying amount of data needs to be sent.
- Supports millions of reassembly contexts and simultaneous reassemblies.
- Reassembles packet segments.
- Transmits partial or complete segments.
- Edits segments to different locations in the reassembly buffer.
- Stops packet reassembly if the timer expires.
- Checks to see when the reassembly exceeds maximum reassembly size and ends a reassembly context.
- Checks sequence numbers for internal integrity checking.
- Checks the status of a reassembly context.
- Monitors reassembly context resource usage.

Functional Description

The PAB engine uses a set of operation commands, an ASE parameter configuration, and input task parameters to perform the following functions.

- Identify (`Enqueue`) a range of segment data that the PAB engine adds to the reassembly context.
- Send (`transmit`) all or part of the reassembled packet. The PAB engine can make multiple copies of the reassembly context. The PAB engine passes all or part of the reassembly context in a task to the next engine.
- Remove (`discard`) selected data from the reassembly context to recover resources.
- Debug (`getStatus`) reassembly – Pass a task to the next engine that does not impact the reassembly context and provides the reassembly state and resource information.
- Perform bit-level editing within the reassembly context.

The PAB engine uses the following configuration options to control the action performed when the PAB engine performs one of its commands.

- Define the placement of the segment in the reassembled packet.
- Identify the data range from the reassembled packet for the PAB engine to pass in a `Transmit` command.
- Time-out a reassembly.
- Check sequence numbers for increased reassembly context integrity.

The following glossary terms are used throughout the PAB engine chapter.

- Initial segment – First segment *sent* to a new reassembly context.
- First segment – The first segment of the packet or data added to the reassembly.
- Terminating a reassembly context – Removing its data from memory and removing the reassembly context from the connection database (CDB).

Identify Segment Data for Reassembly

A segment data can be identified as the whole incoming packet data or partial incoming packet data. The PAB engine adds a segment to a reassembly context each time you use an *Enqueue* command with the **startOffset**, **firstOffset** and **`length`** parameters to define the incoming packet data range as the *new* segment to add to the reassembly.

For more details about the Enqueue commands, see the *Enqueue Commands* heading within the *Functional Description* section for the PAB chapter of this document.

PAB Enqueue Commands

The following Enqueue commands include these PAB engine operations.

- `enqueue` – Adds a segment to an existing reassembly context.
- `enqueueTransmit` – Appends an incoming segment to a particular reassembly ID and transmits the entire reassembly.
- `enqueueTransmitCopy` – Appends a new segment to a particular reassembly ID and transmits a copy of the entire reassembly task to the next engine.
- `enqueueTransmitDiscard` – Appends a new segment to a particular reassembly ID, transmits the entire reassembly, and removes entire reassembly (de-allocates resources).

Enqueue Process (Add Segment)

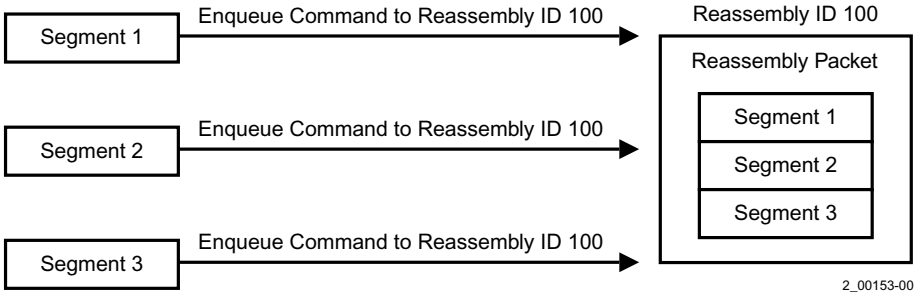
The PAB engine Enqueue commands add segments to an existing reassembly context. You can control the placement of each segment in the reassembly context space. You can overlap segments or leave gaps. This means that in addition to reassembling segments, you can perform byte- and bit-level segment editing.

You can also define the following different byte offsets.

- One to delete from the start of the initial segment.
- An offset to delete from all other Enqueued segments in the reassembly.

Enqueue command parameters enable you to add an incoming task to any point in the reassembly. Optionally, a range of data can be selected for enqueuing by using the input parameters. The result of assembling segments becomes a reassembly packet. The following figure illustrates multiple segments added to the reassembly, the `enqueue` commands, and the reassembled packet.

Figure 1 Enqueue Command Creates a Reassembly Packet



The previous figure shows three tasks, and the PAB engine performs one task for each segment. The PAB engine receives each input task with an `enqueue` command. The command parameter (**reassemblyIndex**) is 100. The result of the three operations is a reassembly packet made up of segments one, two, and three.

Transmit Reassembled Packet

Use the PAB engine `Transmit` command to send the packet data (in a task) to the next accelerator engine in the Virtual Pipeline sequence.

Using Transmit Commands

The following brief descriptions explain the PAB Engine Transmit operations.

- `transmit` – Sends the packet data (in a task) to the next accelerator engine in the Virtual Pipeline sequence.
- `transmitCopy` – Sends packet data more than once (without potential conflicts) or send multiple versions of a packet, each with different (edited) headers.

  When using the `transmitCopy` command, if the PAB engine sends data more than once from the same reassembly, it creates two or more tasks that point to the same packet data. If different tasks reference the same packet in memory, there is potential for overwriting data because downstream engines process the tasks independently. You also can overwrite data when you edit packet data in the PAB engine after it is sent.

  The `transmitCopy` – Duplicates the packet data for the reassembly in memory and transmits the duplicated packet data from a reassembly ID with a new task. Because the original data remains in the reassembly, the state does not change. The `transmitCopy` command includes information about the offset and length of the reassembly data sent.
- `transmitDiscard` – Sends the original reassembly pointers in an output task and discards reassembly packet data, starting from the beginning of the reassembly through the last byte transmitted. The command sends a packet from a particular reassembly ID and includes the offset and the reassembly length.

NOTE The PAB engine uses special handling for the `discard` and `transmitDiscard` commands that specify a non-zero **lengthBitOffset**. If the **lengthBitOffset** ends at a non-byte boundary, the PAB engine discards only up to the prior byte boundary. This means the discard is rounded down. If the **lengthBitOffset** does not end on a byte boundary, the discard generates unexpected results.

Transmit Reassembly Process

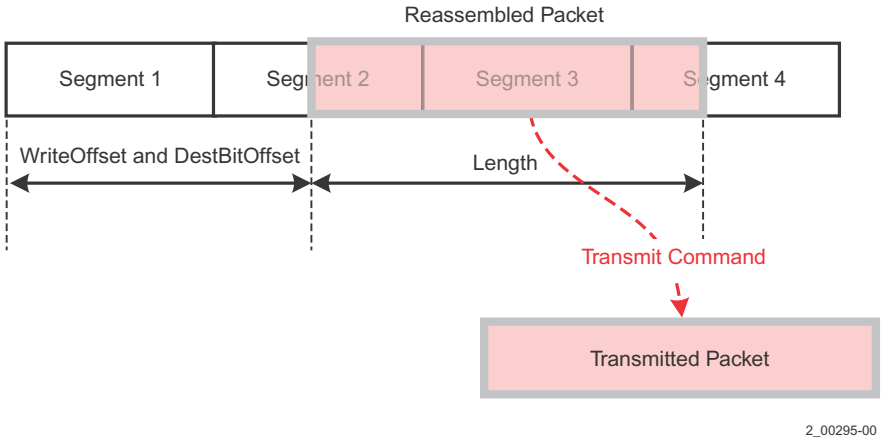
The PAB engine uses the Transmit command to send a task to the next engine using a specific reassembly ID. You can select part of a reassembly or the complete reassembly as the packet sent within an Output Task.

You can use the input task parameters commands to define the packet offset and length information. Instead of defining the range of data for the segment, the input parameters used by the `transmit` operation defines the range of reassembly data sent in a task by the PAB engine.

The `transmit` command does not discard information from the reassembly. To start a new reassembly, use the Transmit with Discard (`transmitDiscard`) command to discard the reassembly.

The following figure shows four segments in a reassembly. The transmitted packet consists of the end of segment two, segment three, and beginning of segment four. The **writeOffset**, **destBitOffset** and **length** input parameters identify the packet information range to send.

Figure 2 Transmit Command Sends Partial Reassembled Packet Packet



Discard Reassembly Context

Each active reassembly context takes up memory in the Namespace table created to hold the PAB engine connection database (CDB). You should always release Namespace entry reassembly contexts to free memory for any of the following reasons.

- The PAB engine completed the reassembly and sent the reassembled packet (task).
- An error happened during reassembly and you should discard the reassembled packet.
- The reassembly context timed out or exceeded the maximum defined size (**sizeThreshold**) input parameter.

Use the `discard` command to delete a reassembly context or a subset of reassembly data. The `discard` command can delete all or part of the reassembly data. It starts deleting data from the beginning of the reassembly and deletes the number of bytes defined in the **length** input parameter. A **length** value of zero deletes the entire reassembly.

PAB Discard Commands

The Discard commands include the following PAB engine operations.

- `discard` – Delete a reassembly context (discard all) or a subset of the reassembly data.
- `transmitDiscard` – Sends the portion of the reassembly selected by the input task parameters and deletes the reassembly context.
- `enqueueTransmitDiscard` – Appends a new segment to a particular reassembly ID, sends the entire reassembly as a task to the next engine, and removes it (de-allocates resources) from the reassembly.
- `cleanup` – Discards the reassembly without performing any Sequence checking or verify the `discardOnTimeout` parameter matches the value for this reassembly ID in the CDB.
- `stickyDiscard` – Marks the reassembly as dirty. The command triggers the PAB engine to free memory for the reassembly ID and drop all subsequent enqueued data segments. The PAB engine maintains the reassembly in this state until the PAB receives one of the following PAB functions.
  - A complete Transmit with Discard
  - Cleanup
  - Enqueue with Transmit and Discard
  - Discard

Discard on Exception

You can also set the input task parameters to discard a reassembly as part of an exception error. When you exceed a set time (**timerValue**) or when the reassembly exceeds the maximum size (**sizeoutThreshold**), you can discard the assembly by selecting the **discardOnTimeout** or **discardOnSizeout** input parameter. If the PAB engine sends another segment to the reassembly context, it is treated as if it is the *initial segment* sent to that existing reassembly context. When you *do not select* **discardOnTimeout** or **discardOnSizeout** flag, the time-out or sizeout event triggers the PAB engine to send the packet to the next engine in the engine sequence.

Bit-level Editing Operation

The PAB engine supports bit-level editing of packet segment data by using the two bitFields input task parameters for a PAB command within the virtual pipeline instance. The PAB engine can perform the following bit-oriented operations.

- Insert a bit stream into an existing assembly at an arbitrary starting bit position, relative to the beginning of the assembly.
- Transmit a bit stream from an assembly beginning with an arbitrary starting bit position.
- Advance an assembly head pointer and length status to delete data from the front portion of the assembly context.

Timeout Reassembly

To maintain timers, the PAB uses the Timer Manager (TMGR) engine. A current time counter synchronizes with Axxia Time and maintains the order of timer requests and events.

The PAB engine can time reassemblies for time-out error cases or trigger packet transmission for time-sensitive applications. The PAB engine associates timers with specific reassembly indexes.

NOTE When using timers for tasks other than clean up (discard assembly and re-initialize statistics when the PAB engine detects an unknown state), an accelerator engine (for example, MPP or CPU) sending commands to the PAB engine has difficulty determining the current reassembly context status.

NOTE When starting multiple timers for a reassembly context, the PAB engine stores only the most recent value in the reassembly Connection Database.

Checking Sequence Numbers

Use sequence numbers to ensure that the Axxia device does not discard tasks sent to the PAB engine. The PAB engine can verify the sequence number of each segment it receives for reassembly. You can enable sequence number checking (**seqCheckEnable**) and supply a sequence number (**seqNumber**) using the PAB engine input task parameters.

Configure sequence number checking by using the ASE hierarchy path: **AXX5500 > VirtualPipelines > [StartEngineName: EIOA | MPP | CPU | Expander] > VirtualPipeline > EngineSequence > PAB Dialog Tab**.

Because the sequence number continues when the PAB engine creates a new reassembly context with this reassembly ID, it enables the feature to cover the initial `enqueue` command to a reassembly context. If the check fails, the PAB engine action marks the reassembly context as an error. It frees up memory allocated to this reassembly context (`discard`) and drops future **enqueue**s to this reassembly context. Optionally, the PAB engine can send a task to provide an exception notification.

To configure the reassembly sequence checking for different segments, see the *Sequence Number Checking* heading in the *Engine Configuration* section for this chapter.

Task and Resource Management

The PAB engine manages input tasks using the following configuration elements.

- Buffer Management – Set memory thresholds globally and for different task priorities.
- Task Receive Queues – Define backpressure units, queue arbitration, queue priorities, and discard policies.
- Task Trace – Log tasks to trace multiple events.

The PAB engine supports buffer management to manage memory usage for reassembly contexts. The PAB engine can be configured to discard a particular reassembly based on input task priority if the overall amount of memory for a specific priority reaches the configured threshold. The PAB engine also supports orderly input task dropping (priority-based) when an input task receive queue depth reaches the configured threshold.

The PAB engine has two task receive queues, the high and low priority task receive queue. Tasks are transmitted to the PAB engine with a 3-bit task-priority value. The PAB engine maintains a table, configurable through the ASE, that maps the received task priority to a PAB task receive queue.

Each engine can trace multiple events with detailed input parameter and task information.

For the reassembly context memory, you configure one or more Namespaces to create the **PABReassemblies** engine table. These tables hold the context information for the packet reassemblies.

You can define one or more PAB engine memory tables to use for reassemblies by using this hierarchy in the ASE Outline View: **AXX5500 > Namespaces > Namespace > PABReassemblies** and configuring the **Namespace** attributes.

To allocate memory for the PAB reassemblies for a PAB engine, see the *Reassembly Namespaces* heading in the *Engine Configuration* section for the PAB chapter of this document.

Obtain Reassembly Status and Memory Usage

The PAB engine `getStatus_xxxx` commands enable you to send a task that does not affect the reassembly context data. It forces the PAB engine to generate an output task with the state and priority information for the requested assembly context. This information helps to troubleshoot you PAB applications during development. The PAB engine can use the following `getStatus_xxxx` commands to obtain debugging information.

- `getStatus_reasmState` – Checks the connection database state for a specify reassembly ID.
- `getStatus_reasmPrioMemState` – Checks the PAB memory usage for the priority used by a particular reassembly, along with the total memory usage for all task priorities.
- `getStatus_PrioMem` – Provides you the memory usage for a specified task priority, along with the total memory usage.

To determine the memory usage with a specific reassembly and specific task priority reassemblies, see the *Get Reassembly Status* heading within the *Statistics and Monitoring* section for the PAB chapter of this document.

Engine Configuration

For the PAB engine to process packets accurately, you must configure these elements of each PAB engine in an engine sequence.

- PAB engine buffer management thresholds
- PAB reassembly resource namespaces
- PAB engine (input task parameters) in the Virtual Pipeline instance

The PAB engine uses the following configuration options to control the action performed when the PAB engine performs a core function.

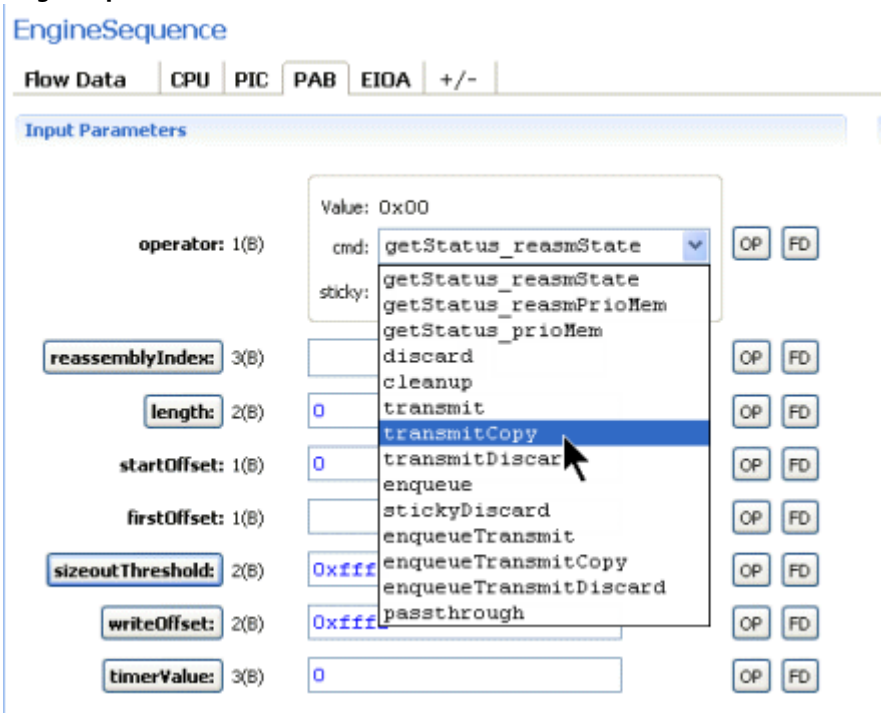
- Define the placement of the segment in the reassembled packet.
- Identify the range of data from the reassembled packet for the PAB engine to pass in a `transmit` command.
- Time-out a reassembly.
- Check sequence numbers for increased reassembly context integrity.

The PAB engine uses a set of operation commands, an ASE parameter configuration, and input task parameters to perform the following core functions.

- Identify (`enqueue`) a range of segment data that the PAB engine adds to the reassembly context.
- Send (`transmit`) all or part of the reassembled packet. The PAB engine can make multiple copies of the reassembly context. The PAB engine passes all or part of the reassembly context in a task to the next engine.
- Remove (`discard`) selected data from the reassembly context to recover resources.
- Debug (`getStatus`) reassembly – Pass a task to the next engine that does not impact the reassembly context and provides the reassembly state and resource information.
- Perform bit-level editing within the reassembly context.

Select the PAN input and output task parameters by using the ASE hierarchy path: **AXX5500 > VirtualPipelines > [StartEngineName: EIOA | MPP | CPU | Expander] > VirtualPipeline > EngineSequence > PAB Dialog Tab**.

Figure 3 PAB Engine Operation Commands



To configure the buffer management thresholds, see the *Buffer Management* heading in the *Engine Configuration* section for the PAB chapter of this document.

To select specific input parameters to control the key PAB commands, see the *PAB Command Input Parameters* heading in the *Engine Configuration* section for the PAB chapter of this document.

Task Management

The PAB engine manages input tasks using the following configuration elements.

- **Buffer Management** – Set memory thresholds globally and for different task priorities.
- **Task Receive Queues** – Define backpressure units, queue arbitration, queue priorities, and discard policies.
- **Task Trace** – Log tasks to trace multiple events.

The PAB engine supports buffer management to manage memory usage for reassembly contexts.

The PAB engine has two task receive queues, the high priority and low priority task receive queue. Tasks are transmitted to the PAB engine with a 3-bit task-priority value. The PAB engine maintains a table, configurable through the ASE, that maps the received task priority to a PAB task receive queue.

Each engine can trace multiple events with detailed input parameter and task information.

Buffer Management

The PAB engine supports buffer management to manage memory usage for reassembly contexts. The PAB engine can be configured to discard a particular reassembly, based on input task priority, if the overall amount of memory for a specific priority reaches the configured threshold. The PAB engine also supports orderly input task dropping (priority-based) when an input task receive queue depth reaches the configured threshold.

You can configure the PAB engine buffer management thresholds using this hierarchy in the ASE Outline View: **AXX5500 > Engines > PAB**.

The following PAB engine parameters must be configured to support buffer management.

Reassembly buffer management attributes.

AXX5500 > Engines > PAB	
itlFifoWm	Input task queue backpressure threshold that signals full. Data Type: Integer Default: 15
bmTotalThresh	Global reassembly buffer management threshold (bytes). Set to less than the total amount of memory available for PAB reassembly storage. Buffer management occurs only for a task when the following events happen together. ■ Memory in use for its priority exceeds the per-priority threshold *and* ■ Total amount of memory in use by PAB (across all priorities) exceeds the `bmTotalThresh` setting. Data Type: Integer Default: 0x100000
bmThresh0 bmThresh1 bmThresh2 bmThresh3 bmThresh4 bmThresh5 bmThresh6 bmThresh7	Reassembly buffer management threshold (bytes) for each input task priority. Data Type: Integer Default: 0x20000
maxOparamLen	Maximum number of output parameter bytes received with the input task (selected from the input task parameters, as specified by the input template merge) that are supplied to the PAB with each new task. The value cannot be reduced while the application runs; however the value can be increased. Data Type: Integer Default: 80 (0x50)

Task Receive Queues

The PAB engine has two task receive queues, the high priority and low priority task receive queue. Tasks are transmitted to the PAB engine with a 3-bit task-priority value. The PAB engine maintains a table, configurable through the ASE, that maps the received task priority to a PAB task receive queue.

The arbitration style between the high and low priority task receive queues can be configured as *strict priority*, *rotating*, *simple round robin,* or *weighted round robin*. The **taskQueueArbitration** attribute defines the arbitration mode of the task priorities. You can select from four types of arbitration methods to configure the priorities of task receive queues. The priorities values range from 0 through 7. Priority 0 is the highest priority, and priority 7 is the lowest.

The following examples assume a four queue setup and describe each task queue arbitration method.

Strict Priority – All higher priority tasks are serviced before lower priority tasks. The lower-numbered queues have higher priority than higher numbered queue.

Assume the PAB engine is currently servicing queue two. When there is task in higher priority queue one, it services queue one next.

Rotating – Prioritizes the next queue in the sequence, even if queues were previously skipped because they were empty.

Assume the PAB engine is about to service queue two, however queues two and three do not contain tasks to service. The PAB engine services queue four. On the next service round, the PAB engine begins with the queue three, strictly maintaining the servicing sequence as if all queues had queued tasks.

Round Robin – Prioritizes the next queue with available tasks. The PAB engine begins servicing queues with the next queue in sequence after the previously serviced queue.

Assume the PAB engine is currently servicing queue two, however queues two and three do not contain tasks to service. The PAB engine services queue four. On the next service round, the PAB engine begins with queue one, which is the next queue after queue four.

Weighted Round Robin – Define the weights for each task receive queue. Use round robin algorithm and weight to determine how long the currently active queue is serviced.

Assume the PAB engine is currently servicing queue two. When the weight is *greater than zero*, the next serviced queue remains queue two. When the weight equals zero, it services the next queue which has pending tasks to be active in a round robin way. See the following note about setting additional attributes.

NOTE To use the Weighted Round Robin arbitration mode, you must set the **highPrioQueueWrrWeight** (high priority queue weight) and **lowPrioQueueWrrWeight** (low priority queue weight) attributes.

To configure the task receive queues to arbitrate the task priorities and loads, using this hierarchy in the ASE Outline View: **AXX5500 > Engines > PAB > TaskReceive**.

PAB global buffer and queue discard thresholds.

AXX5500 > Engines > PAB > TaskReceive	
thresholdGroupUnits	Threshold units are tasks or 256-byte blocks that define the unit of resources for the **Threshold Group - Used** attribute. ■ `Tasks` ■ `BlockSize256B` Default: `Tasks`
taskQueueArbitration	How the task queues are prioritized. The priorities values range from 0 through 7. Priority 0 is the highest priority, and priority 7 is the lowest. Default: `WeightedRoundRobin`
backpressureHysteresis	Queue level at which to send a backpressure OFF message. ■ `DivideBy16` – Set the level 1/16 below current backpressure ON level. ■ `DivideBy64` – Set the level 1/64 below current backpressure ON level. Default: DivideBy16
highPrioQueueWrrWeight	For `WeightedRoundRobin` arbitration, the weight given to the high priority queue. Range: 0 to 15 Default: 15
lowPrioQueueWrrWeight	For `WeightedRoundRobin` arbitration, the weight given to the low priority queue. Range: 0 to 15 Default: 7

AXX5500 > Engines > PAB > TaskReceive	
thresGroupId	The queue discard group to use for this queue. The Timer has only one threshold group. Read Only
Buffer Thresholds	Buffer thresholds apply to each queue and depend on the number of available 2K-byte blocks of global memory. ■ Discard Random – Randomly discard incoming tasks when the number of global 2K-byte blocks available is below this value. Default: 32 ■ **Discard All** – Discard all incoming tasks when the number of global 2K-byte blocks available is below this value. Default: 16 ■ **Red Slope** – Random Early Discard (RED) slope controls the drop profile. Range: 0.0 to 1.0 Default: 1.0
Threshold Group - Used	■ **Backpressure** – When the number of units in the queue exceeds this value and when the queue is configured as a backpressure source, send a backpressure ON message. When the queue usage decreases to less than the threshold set by the **backpressureHysteresis** attribute, send a backpressure OFF message. ■ **Discard Random** – When the number of units in the queue exceeds this value, randomly discard incoming tasks. ■ **Discard All** – When the number of units in the queue exceeds this value discard all incoming tasks. ■ **Red Slope** – Random Early Discard (RED) slope controls the drop profile.

NOTE Priority zero tasks are never discarded based on queue thresholds. Priority zero tasks can be discarded based on buffer thresholds

Task receive queue threshold values that control when the MME engine fetches additional memory for operations.

AXX5500 > Engines > PAB > TaskReceive > PrefetchMMBBlocks	
num256byteBlocks	The number of 256-byte memory blocks that this accelerator engine attempts to maintain for use by the task receive queue. Default: 24
num2KbyteBlocks	The number of 2-KB memory blocks that this accelerator engine attempts to maintain for use by the task receive queue. Default: 24
num16KbyteBlocks	The number of 16-KB memory blocks that this accelerator engine attempts to maintain for use by the task receive queue. Default: 8
num64KbyteBlocks	The number of 64-KB memory blocks that this accelerator engine attempts to maintain for use by the task receive queue. Default: 4

Task Trace Configuration

Each engine can trace multiple events with detailed input parameter and data information. In addition to the global configuration that enables the trace, the following fields must be configured for each engine to support trace functionality.

You can configure how much task information to log by using this hierarchy in the ASE Outline View: **AXX5500 > Engines > PAB > TaskTrace**.

Configure the logging of engine task trace entries.

AXX5500 > Engines > PAB > TaskTrace	
numLogEntries	Number entries in this trace log instance. Range: 0 to 2^{31}– 1 Default: 0
traceEnabled	Enable this PAB instance for tracing. Default: FALSE
logWrap	When the trace log fills to capacity, add the next entries at the start of the PAB trace log. Default: FALSE
logEntrySize	Set the size of each **traceLog** entry. Range: 32 | 128 Default: 32 bytes
traceAll	Trace all tasks processed by this engine. Default: TRUE
taskPassThrough	TRUE = Pass the **traceEnable** attribute from the input task to the output task. **NOTE** Only an intermediate engine can set this attribute equal to TRUE. Engines such as MPP, CPU, EIOA, NCA, MMB, and Expander do not use this attribute. FALSE = Set the **traceEnable** attribute in the output task to the **traceLog** record status. Default: TRUE
sequenceNumberBase	Specify the lowest sequence number assigned to this trace instance. Default: 0x0 Read Only

Reassembly Namespaces

You configure the PAB reassembly context memory space within the ASE menu .

You can define one or more PAB engine memory tables to use for reassemblies by using this hierarchy in the ASE Outline View: **AXX5500 > Namespaces > Namespace > PABReassemblies** and configuring the following **Namespace** attributes.

Packet Assembly Block engine namespace attributes.

AXX5500 > Namespaces > Namespace(ID) > PABReassemblies	
calculated	Compiler calculates the *stripped* feature when enabled. Default: TRUE (selected)
striped	Tables within the same namespace can share the same cache line. Default: FALSE
entrySize	Size of each table entry. Range: 128 | 256 When the PAB engine uses a timer, select 256 bytes; otherwise, select 128 bytes. Default: 128

Use one of the four `Enqueue` commands to create a PAB reassembly context. To start a new reassembly context, the enqueue command must be assigned a reassembly index, a pointer into the reassembly table that represents an entry not currently being used for another reassembly. The segment is not required to be the first segment of the packet being reassembled. A new reassembly can be created by enqueueing any segment of the packet.

The following attributes must be configured *after* the **PABReassemblies** namespace is created.

Individual namespace attributes.

AXX5500 > Namespaces > Namespace)	
name	Name assigned for this namespace.
fplManaged	Select TRUE to manage the namespace entries with an FPL program. The maximum number of FPL-managed namespaces is 16. When selected, the *fastpath* code in the MPP engine must use HASH engine function calls to permit the FPL program to manage the reassembly. Default: FALSE
numEntries	Number of entries to reserve for each namespace (multiples of 20). Because each reassembly requires a PAB namespace entry, the total number of defined entries must be enough for the maximum number of simultaneous reassemblies the application supports.
baseId	The base ID of the namespace. If **auto-assigned** selected, the ASE computes this value after building the configuration image. Deselect this field to control what base ID values to use for the **PABReassemblies** namespace. Default: Assigned by the ASE
managementDomain	Domain currently managing the namespaces. Read Only

PAB Command Input Task Parameters Usage

Each of the PAB engine commands use the input task parameters to specify what part of the packet reassembly on which to operate.

The PAB engine uses the input task parameters to control the following functions.

- Select what part of the reassembly to insert (`enqueue`) a new segment.
- Select what part of the reassembly to transmit.
- Select the portion of the reassembly to discard.

Select Position to Enqueue Segment in Reassembly

When a segment is added to a reassembly context, you can use a PAB `enqueue` command with the `writeOffset` input parameter to define the segment starting point in the reassembly context.

You can add segments to a reassembly context in any order. The following examples show common procedures.

- Add a segment that creates a *hole* by adding the third segment to the reassembly context before the second segment arrives.
- Add a segment that *overlaps* another segment and overwrites existing data in the reassembly context.

Use this technique as an alternative method of editing packet data.

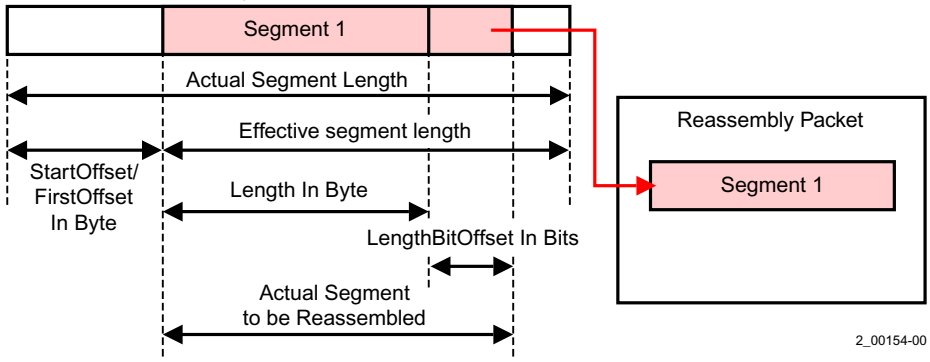
The following input task parameters can be configured for the different `enqueue` commands.

- **length**
- **startoffset**
- **firstOffset**
- **writeOffset**
- **lengthBitOffset**

- **destBitOffset**

The requirements for each parameter are described in the following paragraphs. The special conditions that affect how the PAB engine implements the `enqueue` command parameters is described with each input parameter.

Figure 4 Actual Segment Assembly



length

This input parameter directs the PAB engine to enqueue the entire segment, assuming the **lengthBitOffset** value is also zero. The PAB engine uses the **length** value, the **startOffset**, **firstOffset**, **lengthBitOffset**, and the segment length values to determine which part of the segment to enqueue and where to enqueue it. The PAB applies the **startOffset** data to *every* segment except the *initial segment* that begins the reassembly. For the initial segment, the PAB uses the **firstOffset** instead of the **startOffset** parameter. The previous figure illustrates the actual segment size the PAB engine reassembles as a function of the actual segment size, **startOffset**, **lengthBitOffset**, and **length** parameters.

NOTE When `length+lengthBitOffset=0`; the length for reassembly is the effective segment length. The default value is zero.

startOffset

The **startOffset** parameter defines the number of bytes to exclude from the original segment. For the initial segment of the reassembly, the PAB engine uses the **firstOffset** value and ignores the **startOffset** value. Normally the **startOffset** value does not exceed the actual segment length.

The following conditions apply for the use of **startOffset**:

- When the **startOffset** value exceeds the actual segment length, the actual segment length for reassembly is zero. The PAB engine treats the input task as containing no data.
- If the actual segment length exceeds the **startOffset** value and the value of adding `startOffset+length+lengthBitOffset` exceeds the actual segment length, the actual segment to be reassembled is the effective segment length. In Figure 4, the `Actual_segment_length-startOffset` is called the effective segment length. A sticky interrupt status bit is set to record this unexpected event.

The default value is zero.

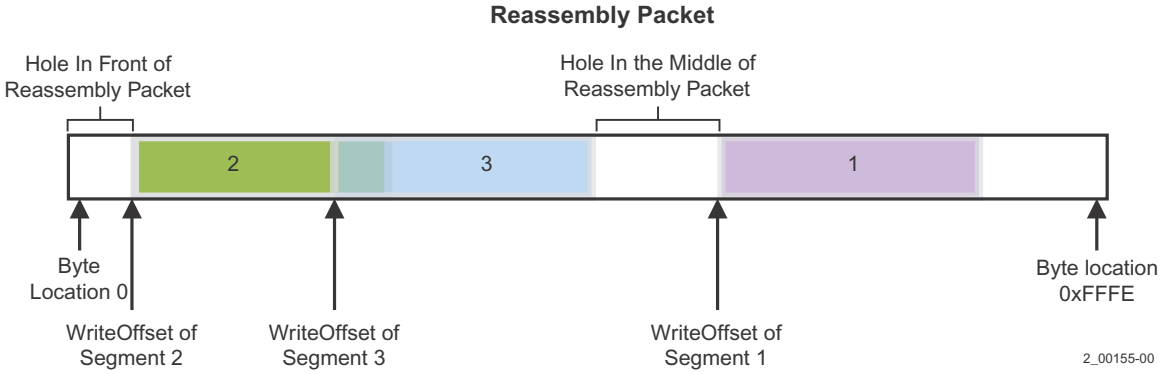
writeOffset

The writeOffset default value (0xFFFF) defines the enqueued segment at the end of the current reassembly. Other values specify the location where the PAB engine writes the first segment byte. Because a segment can be enqueued in any location of the reassembly packet, the following three possible results exist:

- An empty space remains in front of a reassembly packet.
- One or more gaps remains in the middle of a reassembly packet.
- Part of a previously enqueued segment is overwritten.

The following figure illustrates these condition.

Figure 5 Potential writeOffset enqueue Commands Results



In the preceding figure, there are three `enqueue` commands associated with Segment One (purple), Segment Two (green), and Segment 3 (blue). Segment One is the initial segment enqueued to the reassembly packet. Segments Two and Three are subsequent `enqueue` commands to the reassembly. Segment One **writeOffset** results in a hole in the reassembly packet. Because Segment Three **writeOffset** exists within the Segment Two data range, Segment Three overwrites data in Segment Two. The **writeOffset** can result in a hole in front of the reassembly as well as the middle of the reassembly.

When it is necessary to transmit two or more times from a single active reassembly, the PAB engine restricts the minimum value for the **writeOffset** parameter. It requires the minimum **writeOffset** value to exceed or equal a specific value. This value points to the location immediately after the last byte of the previously transmitted packet reassembly.

For a reassembly of 1000 bytes and using a `transmit` command with the **length** of 500 bytes. In this case, the minimum value of the following **writeOffset** must exceed or equal to 500. This prevents data from within a reassembly from being referenced by two or more output tasks, potentially resulting in overwriting data.

lengthBitOffset

Use the **lengthBitOffset** parameter and the **length** parameter to specify the segment length to enqueue in bit units. You leave this field set to zero for byte-level reassembly and specify the reassembly segment length with the **length** input parameter, in byte units. The default value is zero.

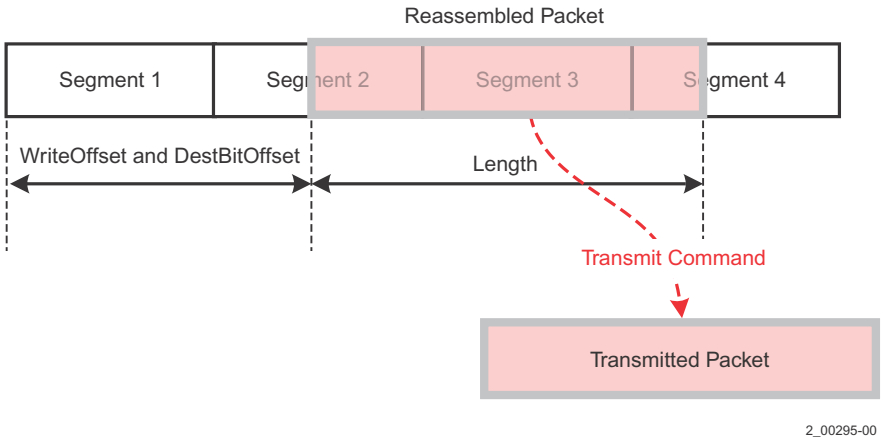
destBitOffset

Use the **lengthBitOffset** parameter and the **writeOffset** parameter to specify the write location of the segment to be enqueued in bit units. If the **writeOffset** parameter is set to `0xFFFF`, the PAB engine ignores the **destBitOffset** parameter. Leave this field set to zero for byte-level reassembly and specify the write location of the reassembly segment with the **writeOffset** command parameter, in byte units. The default value is zero.

Select Reassembled Data To Transmit

When the packet data reassembly is complete, it can be transmitted as a packet to the next engine in the engine sequence. When the packet is transmitted, the `startoffset`, `length` and `writeoffset` parameters can be defined for data packet sent in an output task.

Figure 6 Transmit Command Sends Partial Reassembled Packet Packet



The following PAB input parameters can be configured for the Transmit commands.

- **length**
- **writeOffset**
- **lengthBitOffset**
- **destBitOffset**

The following sections describe the command parameters.

length

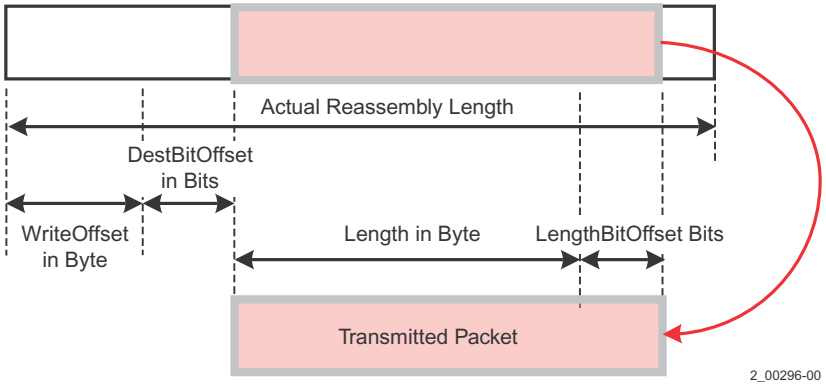
The **length** parameter specifies the PAB engine transmits the entire reassembly from the specified WriteOffset and **destBitOffset**, assuming the **lengthBitOffset** value is zero. When **length** is non-zero, the PAB uses the value with the **writeOffset**, **destBitOffset**, and **lengthBitOffset** values to determine the reassembly part to transmit. The default value is zero.

There are two special conditions:

- When `writeOffset+destBitOffset+length+ lengthBitOffset` exceeds the actual reassembly length, the PAB engine transmits no data and sets the sticky interrupt status bit to indicate this exception.
- When `length+lengthBitOffset` equals zero, the PAB engine transmits the entire reassembly starting from the location specified by the sum of `writeOffset+destBitOffset`.

The following figure illustrates the relationship between these command parameters.

Figure 7 Length Command Parameter Relationships in Reassembly Packet



writeOffset

The default value `0xFFFF` instructs the PAB engine to transmit the reassembly packet from byte location zero. The **writeOffset** and **destBitOffset** parameters indicate the location the PAB will transmit the reassembly packet from. There are two possible exceptions the PAB can generate with **writeOffset** parameter:

- If the sum of **writeOffset** and **destBitOffset** is greater than the actual reassembly length, the PAB generates an exception and does not transmit data.
- If the sum of the **writeOffset**, **destBitOffset**, **length**, and **lengthBitOffset** exceeds the actual reassembly length, the PAB engine generates an exception and does not transmit data.

lengthBitOffset

The default value is 0. Use this parameter with the **length** parameter, to specify the reassembly packet length to be transmitted, in bit units. Leave this field as zero for byte level reassembly and use the **length** parameter to specify the reassembled packet segment length, in byte units.

destBitOffset

Use the **lengthBitOffset** parameter with the **writeOffset** parameter to specify from where to transmit the reassembly. If the **writeOffset** parameter is `0xFFFF`, the PAB engine ignores information in the **destBitOffset**. Leave this field as zero for byte-level reassembly to use the **writeOffset** parameter to specify the destination for the packet, in byte units.

The default value is zero.

Controlling the Packet Discard Process

The `discard` command discards data from the front of a reassembly packet. The **length** and **lengthBitOffset** parameters determine the number of bytes orbits to be discarded from the reassembly packet. If these parameters specify to discard the entire reassembly, the PAB engine terminates reassembly. If the `discard` command runs and data remains in the reassembly, the PAB engine treats the first byte of data not discarded as being at offset zero for future PAB commands and sets the `MinEnqueueOffset` to zero. Application software must track this value for future commands on the reassembly to specify a non-default write offset for it. This means there is no need to track the value if the software has no requirement to specify a write offset.

The following command parameters must be configured for this command.

- **length**
- **lengthBitOffset**

The parameters are described in the following sections.

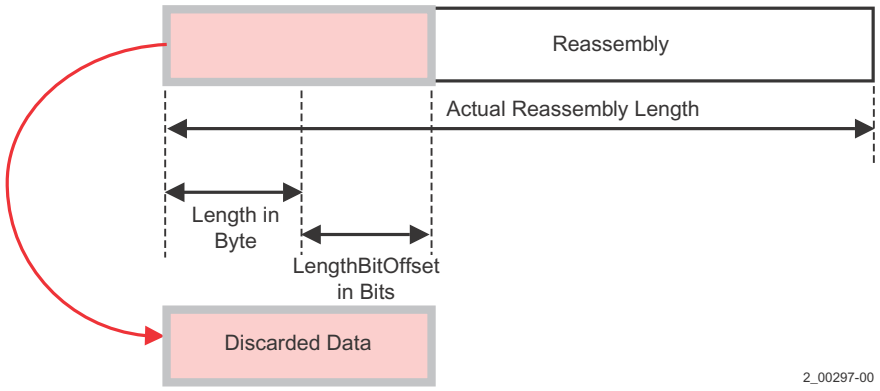
length

The default value zero indicates the PAB should discard the entire reassembly. For other values, the **length** and **lengthBitOffset** parameters specify the number of bytes and bits the PAB removes from the beginning of the reassembly.

A special condition applies to the **length** parameter when used with the `discard` command:

If the `Length + LengthBitOffset` value exceeds the actual reassembly length, the PAB engine discards the entire reassembly and generates an exception event, because it attempted to access non-existing data.

Figure 8 Discard Data for Reassembly



lengthBitOffset

Use the **lengthBitOffset** parameter and the **length** parameter, to specify the amount of the reassembly for discard, starting from the beginning of the reassembly, in bit units. Leave this field as zero for byte level reassembly. If this value is zero, the **length** parameter value specifies the reassembly amount the PAB discards, in byte units. The default value is zero.

NOTE The PAB uses special handling for `discardTransmit` with Discard commands that specify a non-zero **lengthBitOffset**. If the **lengthBitOffset** ends at a non-byte boundary, the PAB discards only up to the prior byte boundary. This means the discard is rounded down. If the **lengthBitOffset** does not end on a byte boundary, unexpected results are generated.

Configuring Timer and Examples

To start a timer for a reassembly context, you provide a value for the **timerValue** as an input task parameter in the PAB engine within a Virtual Pipeline instance and select a **timerControl** action in the **advancedFlags** field. The options available using the **timerControl** selection include the following timer actions.

- Enable or disable the timer
- Start the timer based on the initial segment or any designated segment (every, current, last, and so forth) the PAB enqueues for addition to the reassembly.

The following table describes the different **timerControl** actions.

Table 1 Selecting timerControl Actions

Action (Bit Value)	PAB Reassembly Action
`none` (00)	Do nothing (default setting)
`stop` (01)	Stop Timer: The PAB engine does not time out the reassembly even when the timer value expires.
`restart` (10)	Start Timer for the initial segment of the reassembly: Use this action when the time-out needs to start on the arrival of the initial segment in the reassembly, not just any segment.
`restartOnFirst` (11)	Start on first reference to an inactive reassembly context: Use this option when the time-out needs to start on the arrival of the initial segment in the reassembly, rather than any segment.

When a reassembly is terminated by a command (`discard`) or an exception (`oversize PDU`) the PAB engine resets the timer. For example, the PAB engine can start a reassembly and the timer. When the oversize exception happens because the reassembly size exceeds the **sizeoutThreshold**, the PAB engine resets the reassembly. In this case, the PAB does not generate an output task due to the reassembly time-out.

The following examples illustrate use of the **timerControl** parameter.

Example 1 – Time-out Value Based on Initial Segment Arrival

The time-out value depends on the arrival of the initial segment into the reassembly. There are three **enqueue** commands with the **advanceFlags** fields **timerControl** set to `11` and **discardOnTimeout** check box deselected. This results in the reassembly transmitted when the timer started by the first enqueue expires.

- Segment 1: `enqueue` with **timerControl** set to `11`
- Segment 2: `enqueue` with **timerControl** set to `11`
- Segment 3: `enqueue` with **timerControl** set to `11`

Example 1 – Time-out Value Based Based on Last Segment Arrival

The time-out value depends on the arrival of the last segment onto the reassembly. There are three segments for the reassembly and on the last segment the PAB engine performs a **transmitDiscard** command.

- Segment 1: `enqueue` with **timerControl** set to `10`
- Segment 2: `enqueue` with **timerControl** set to `10`
- Segment 3: `transmitDiscard` with **timerControl** set to `01`

NOTE Because the reassembly is terminated with a `discard` command, setting the **timerControl** to 01 is not mandatory.

Sequence Number Checking

The PAB engine writes the **seqNumber** and **seqCheckEnable** input task parameter values to the Connection Database (CDB) for every command except the `getStatus_xxxx` and `passThrough` commands. When the PAB engine reuses a particular reassembly index for a new reassembly context, by default, it uses the previous sequence numbering for that reassembly index.

You can change the sequence checking behavior by setting the **blockSeqCheck** input task parameter. By selecting this check box, you can turn off sequence error checking for this command. If you select the **seqCheckEnable** *and* the **blockSeqCheck** check box, the PAB engine resets the reassembly context sequence check and starts a new count for the reassembly context in the CDB.

Selecting both the **blockSeqCheck** and **seqCheckEnable** check box input parameters, plus the current state of the reassembly controls the PAB engine sequence checking. The sequence number update occurs only when the CDB state of the previous segment shows sequence checking enabled.

You can use the combination of the **blockSeqCheck** and **seqCheckEnable** to define a sequence number check for a particular reassembly ID. The following table shows the sequence checking actions performed when selecting the **blockSeqCheck** and **seqCheckEnable** input parameter check boxes.

Table 2 Input Parameter Control of Sequence Checking

SeqCheckEnable	BlockSequenceCheck	Update Reassembly CDB with the seqNumber Value	Perform Sequence Check
Not selected	Not selected	No	No
Not selected	Selected	Yes, except for these commands: **getStatus_xxxxxx** **passThrough**	No
Selected	Not selected	Yes, except for these following commands: **getStatus_xxxxxx** **passThrough**	Yes, except for these commands: **cleanup** **passThrough** **getStatus_xxxxxx**
Selected	Selected	Yes, except for these commands: **getStatus_xxxxxx** **passThrough**	No

NOTE To initialize the reassembly sequence number, select the **blockSeqCheck** input parameter check box.

NOTE To run the sequence check for the incoming segment, select the **seqCheckEnable** check box.

Input Task Parameters

This section describes the PAB task input parameters.

Select the input and output PAB task parameters by using the ASE hierarchy path: **AXX5500 > VirtualPipelines > [StartEngineName: EIOA | MPP | CPU | Expander] > VirtualPipeline > EngineSequence > PAB Dialog Tab**.

The following table shows the input parameters and their descriptions.

Table 3 PAB Input Parameter Usage

Input Parameter Name	Size	Description
operator (cmd)	1B	Defines the PAB command with 5 MSB bits and 3 LSB sticky bits.
reassemblyIndex	3B	The Reassembly ID, default value of `0xffffff`.
length	2B	Length information in unit of bytes. The meaning of the Length is different for different command. ■ Length for `enqueue` commands ■ Length for `transmit` commands ■ Length for `discard` commands To configure the **length** in the `enqueue` commands, see the *Select Segment Position in Reassembly* heading in the *Engine Configuration* section for the PAB chapter of this document. To configure the **length** in the `transmit` commands, see the *Select Reassembled Data To Transmit* heading in the *Engine Configuration* section for the PAB chapter of this document. To configure the **length** in the `discard` commands, see the *Select Reassembled Packet Data To Transmit* heading in the *Engine Configuration* section for the PAB chapter of this document. Default is `0x0000`.
startOffset	1B	The number of bytes the PAB engine removes from the reassembled segment. For the initial segment to arrive in the reassembly context, use the **firstOffset** instead of the **startOffset** input parameter. To configure the two offset parameters in the `enqueue` commands, see the *Select Segment Position in Reassembly* heading in the *Engine Configuration* section for the PAB chapter of this document. Default value of zero.
firstOffset	1B	Number of bytes the PAB engine removes from the initial segment to arrive in the reassembly context. Default is `0x0000`.
sizeoutThreshold	2B	Maximum size of reassembly in bytes. Value of `0xFFFF` indicates that there is no limit in reassembly size. Default is `0xFFFF`.
writeOffset	2B	This information is used for both `enqueue` and `transmit` commands: ■ When the `enqueue` command is used, the **writeOffset** indicates the byte location in which this segment is to be placed for reassembly. The value of `0xFFFF` means append at the end of current reassembly for the `enqueue` command. To configure the **length** in the `enqueue` commands, see the *Select Segment Position in Reassembly* heading in the *Engine Configuration* section for the PAB chapter of this document. ■ When the `transmit` command is used, the **writeOffset** indicates the beginning byte location to be transmitted from the reassembly. For the `transmit` command, value of `0xFFFF` means transmit from the beginning of the reassembly. To configure the **length** in the `transmit` commands, see the *Select Reassembled Data To Transmit* heading in the *Engine Configuration* section for the PAB chapter of this document. ■ For the `enqueueTransmit` command, the **writeOffset** is used for enqueueing, indicating the byte location where the segment is to be placed and the entire reassembly is transmitted. Default value is `0xFFFF`.
timerValue	3B	Specify the time delta for the timer. Value of 0 indicates no timer. Default value is `0x000000`.

Input Parameter Name	Size	Description	
advancedFlags	1B	**Bit**	**Field**
		7	**exceptionNotify**
		6	**bmIgnore** – Ignore all buffer management when selected. Default: FALSE
		5	**discardOnTimeout** – Discard the packet when timer expires. Default: FALSE
		4	**discardOnSizeout** – Discard the packet when reassembly exceeds the **sizeoutThreshold** value. Default: FALSE (not selected)
		3:2	**timerControl** – Use the timer to control packet reassembly time constraints and transmit requirements. ■ 00=No timer used ■ 01= Stop ■ 10= Start unconditionally ■ 11= Start timer if the task references an inactive reassembly. Default: `none`
		1	**seqCheckEnable** – Use sequence numbers to ensure packet reassembly integrity when selected. Default: FALSE (not selected)
		0	**blockSeqCheck** – Use to initialize the reassembly sequence number when selected. Default: FALSE (not selected)
bitFields	1B	**Bit**	**Field**
		7:5	**lengthBitOffset**. This offset has different meaning with different combination. ■ **lengthBitOffset** in `enqueue` command ■ **lengthBitOffset** in `transmit` command ■ **lengthBitOffset** in `discard` command
		4:2	■ **destBitOffset** in `enqueue` command ■ **destBitOffset** in `transmit` command
		1:0	Reserved
		Default value of 0	
seqNumber	1B	Segment Sequence number. Default: 0	
taskOrdering	1B		

PAB Commands

As part of each input task, the first 8 bits of the input task parameters is called the **operator**, and the 5 MSBs (Most Significant Bit) of the operator is designated the PAB command. The remaining three bits are sticky bits used for inter-engine communication. The PAB command field specifies the PAB engine instruction. Fourteen valid instructions can be passed using 14 unique binary patterns. Do not pass any pattern other than these valid 14 binary values. The following table shows the instruction name, its binary code, and a synopsis of its functionality.

ATTENTION Behavior for command encodings (binary values) not described in this table is undefined.

Table 4 PAB Commands (operator)

Command Category	Binary Value	PAB Command Name	PAB Command Functionality
Get Status	00000	`getStatus_reasmState`	Returns to the PAB client the PAB memory usage (256-byte units) for the task priority specified by the selected reassembly.
	00001	`getStatus_reasmPrioMem`	Returns to the PAB client the PAB memory usage (256-byte units) for the task priority specified by the selected reassembly and the memory usage (256-byte units) for all task priorities.
	00010	`getStatus_PrioMem`	Returns to the PAB client the PAB memory usage (256-byte units) for the task priority specified by the PAB input task and the memory usage (256-byte units) for all task priorities.
Discard	00100	`discard`	Discard data from the indicated reassembly, starting at the beginning of the reassembly, for the specified length. If all of the data is discarded, the PAB engine deletes the reassembly.
	00101	`cleanup`	Discard all reassembly data, initialize CDB state. Used only when the state of the assembly is unknown, and you want to clear it.
	10100	`stickyDiscard`	Marks the assembly as dirty. Frees all memory and causes all subsequent enqueued data to be dropped until there is a full `transmitDiscard`, `enqueueTransmitDiscard`, standalone `discard`, or `cleanup` command.
Transmit	01000	`transmit`	Transmit from the specified offset for the specified length, retain reassembly packet data, send pointers to the original reassembly packet data in the output task.
	01010	`transmitCopy`	Same as `transmit` command, but duplicates the reassembly packet data in memory and send the pointers to the duplicate data in the output tasks.
	01100	`transmitDiscard`	Same as `transmit` command, send original reassembly pointers in output task, but discard reassembly packet data starting from the beginning of the reassembly through the last byte transmitted.
Enqueue	10000	`enqueue`	Enqueue input data to reassembly, at the specified offset, for the specified length.
	11000	`enqueueTransmit`	Same as Enqueue command, followed by Transmit command, send original reassembly pointers (updated by Enqueue) in output task.
	11010	`enqueueTransmitCopy`	Same as Enqueue with Transmit command, but duplicate packet data in memory and send the pointers to the duplicate data in the output task.
	11100	`enqueueTransmitDiscard`	Enqueue as above, then transmit the complete reassembly and discard source data.
Do nothing	11101	`passthrough`	Pass input data and pointers to the output task unaltered.

advancedFlags Input Task Parameters

This section describes the following **advancedFlags** input task parameters for the PAB engine in more detail.

reassemblyIndex

The **reassemblyIndex** parameter is the 24-bit reassembly ID to which the segment is to be reassembled. The value of the reassembly ID should be within the list of the configured reassembly IDs. The default value of `0xFFFFFF` *does not* have any configuration meaning.

sizeoutThreshold

The **sizeoutThreshold** parameter specifies the maximum number of bytes for a reassembly. It is also the highest byte number location allowed on a particular reassembly. If a reassembly segment `enqueue` command attempts to write to an address higher than the **sizeoutThreshold**, the PAB engine terminates the reassembly and discards the data from the reassembly. This means the next segment enqueued with the same reassembly ID starts a new reassembly. The default value of `0xFFFF` specifies that no **sizeoutThreshold** exists.

To optimize this command for transmitting, manually reset the **sizeoutThreshold** value to less than the reassembled packet size when enqueueing a segment. This triggers a sizeout exception and forces the PAB engine to transmit the packet.

The PAB engine transmits the reassembly packet data if you selected the **discardOnSizeout** check box.

timerValue

The **timerValue** parameter specifies a duration used to set a timer by the Time Manager (TMGR) requested by the PAB engine (choose a **timerControl** selection). The following items explain how the 24 bits of time information exists in the **timerValue** input task parameter.

- Units (Bit 23) – Seconds for a value of one; zero for microseconds.
- Left Shift Operand (Bits 22:16) – For a signed value, a negative value means right-shifting. If the time unit is seconds, a valid left shifting value must be less than or equal to ten. A left shifting value greater than ten specifies the maximum delay the Timer Manager can provide.
- Timer Value (Bits 15:0).

The default value of zero indicates the PAB engine does not use a timer for this command.

exceptionNotify

There are two kinds of commands the PAB can receive from another engine:

- Commands where the PAB engine creates an output task.
- Commands that the PAB engine acts on, but does not generate an output task.

The **exceptionNotify** parameter (**advanceFlags**) enables the PAB engine to generate an output task during exception processing when it normally would not create an output task.

A default value of zero (deselected) indicates the PAB does not generate an output task when it detects an exception event on a particular reassembly. When selected, the PAB engine generates an output task when an exception occurs.

NOTE The **exceptionNotify** parameter does not generate an output task due to time-out or reassembly sizeout. Select this check box if the Virtual Pipeline sequence terminates in an accelerator engine that can correctly interpret the exception event, for example the MPP or CPU.

bmIgnore

The **bmIgnore** parameter controls buffer management checking. When you select the check box, the PAB engine performs buffer management checking for the incoming task. When deselected, the PAB engine performs no buffer management checking.

discardOnTimeout

When you select the **discardOnTimeout** check box, the PAB engine *does not* generate a task to the next engine when a time-out happens during the reassembly. If a time-out happens, the PAB engine removes all existing segments from the reassembly. This means the PAB engine treats the next enqueued segment as the initial segment for the reassembly. When the check box remains deselected, the PAB engine generates a time-out output task and sends it to the next engine when a reassembly times out.

discardOnSizeout

When you deselect the **discardOnSizeout** check box, the PAB engine generates a task when the number of bytes in a reassembly exceeds the **sizeoutThreshold** value. The PAB also sends the oversize packet to the next engine. When you select the check box, the PAB engine *does not* generate a task for the next engine.

NOTE When the PAB engine detects a timed out or oversize reassembly, it removes all segments from the reassembly. The PAB engine treats the next enqueued segment as the initial segment for the reassembly.

timerControl

You use the 2-bit **timerControl** selection list with the **timerValue** to perform the following operations

- Enable or disable the timer.
- Start the timer based on the initial segment or a designated segment (every, current, last, and so forth) to arrive in the reassembly.

The following table describes the different **timerControl** actions.

Table 5 Selecting timerControl Actions

Action (Bit Value)	PAB Reassembly Action
`none` (00)	Do nothing (default setting)
`stop` (01)	Stop Timer: The PAB engine does not time out the reassembly even when the timer value expires.
`restart` (10)	Start Timer for the initial segment of the reassembly: Use this action when the time-out needs to start on the arrival of the initial segment in the reassembly, not just any segment.
`restartOnFirst` (11)	Start on first reference to an inactive reassembly context: Use this option when the time-out needs to start on the arrival of the initial segment in the reassembly, rather than any segment.

When the PAB engine terminates a reassembly by a command or an exception happens, the PAB engine automatically resets the timer.

For example, the PAB starts a reassembly and the timer. A sizeout exception happens when the size of the reassembly exceeds the **sizeoutThreshold** and the PAB resets the reassembly. In this case, the PAB engine does not generate an additional output task when the reassembly times out.

seqCheckEnable and blockSequenceCheck

You control the reassembly sequence number checking in the PAB engine with the **seqCheckEnable** and **blockSeqCheck** check boxes. The deselected check boxes are the default values (zero or FALSE).

NOTE To initialize the reassembly sequence number, select the **blockSeqCheck** input parameter check box.

NOTE To run the sequence check for the incoming segment, select the **seqCheckEnable** check box.

To properly select the check box combination for your application, see the *Sequence Number Checking* heading in the *Engine Configuration* section for the PIC chapter of this document.

seqNumber

When the previous PAB command performs sequence number checking, the PAB engine checks the value specified in the **seqNumber** parameter when you have selected the **seqCheckEnable** check box. The default value is 0.

sticky

Sticky bits are bit-wise ORed for the following **operator** commands.

- `enqueue`
- `enqueueTransmit`
- `enqueueTransmitCopy`
- `enqueueTransmitDiscard`

The PAB engine saves the current sticky bits in the Connection Database for the reassembly. It initializes anew reassembly with the sticky bits of the initial segment sticky value. The PAB engine sends the sticky bits of a reassembly with its output task as part of the PAB status byte. Software can use sticky bits to pass or accumulate information from input tasks to output tasks. The PAB does not modify or interpret sticky bits except as described within the command descriptions. The sticky bits stored in the CDB are not affected by the `getStatus` and `passthrough` commands.

Output Task Parameters

The Output Parameters for a PAB task have the following different formats.

- A non-`getStatus` PAB command – The output task containing 12 bytes of state information and the reassembled packet (if any) are sent to the next engine in the Virtual Pipeline instance.
- A `getStatus` PAB command – The output task contains the 12 bytes that pass information about the reassembly state and memory resource usage to the next engine. Each of the getStatus commands sends a different set of reassembly information.

Output Parameters For non-getStatus PAB Commands

The PAB task output parameters include:

- A status byte that contains the reassembly sticky bits (from the Connection Database (CDB), and updated by the current task) and status bits indicating when the task experienced an exception or referenced a reassembly with an exception.
- The 3-byte reassembly index value.
- Four bytes of CDB current state for the reassembly, as updated by running the current task. If the CDB Active bit is 0, the remaining fields are undefined. If the CDB is in the *exception state*, where the first status byte exception code is a non-zero value, the priority field is undefined.

The following table shows the output parameters for PAB commands not returning status information in the output task.

Table 6 Output Parameters

Field	Size	Description
Status	1B	Status as defined in the next Status Byte Encoding table.
Reassembly ID	3B	Reassembly index specified by the **reassemblyIndex** input task parameter.
CDB State	4B	CDB State as defined in the PAB Output Parameter CDB State Encoding table.

The following table shows the bit assignments for the PAB Status byte in the output parameters.

Table 7 Status Byte Encoding

Bits	Field
7:5	Sticky Bits
4	Prior Exception. 0 - Exception status due to an exception encountered by this task 1 - Exception status due to a prior exception recorded in the CDB for this reassembly
3:0	Exception code. The bit encoding is defined in the PAB Exception Priority and Descriptions table.

The following table lists the reporting priority, name, and description for each of the PAB engine exception codes.

Table 8 PAB Exception Codes Priorities and Descriptions

Priority for Reporting	Exception Code Name	Description
0	No exception	OK – Reassembly and reassembly packet produced no exceptions.
1	Head Drop	Low priority input task queue reached its discard threshold.
2	Task Error	PAB engine received an input task with the error bit set. The PAB engine drops received tasks that have the task error bit set.
3	Buffer Management	Input task receive queue reached its buffer management threshold.
4	Sticky Discard	PAB engine received a sticky `discard` command for this reassembly ID.
5	Sequence Check Error	PAB engine detected a sequence error for this reassembly ID.
6	Reassembly Corrupted	This is a special condition when an enqueue occurs with the **writeoffset** parameter is less than the minimumEnqueueOffset (bytes). You need special precautions when the PAB engine sends a partial reassembly with data remaining in the reassembly. For example, when there are 128 bytes in a reassembly and the PAB issues a `transmit` command with the **length** of 100 bytes, the minimumEnqueueOffset value for the next enqueue command is 100. Any attempt to write to a byte location less than 100 is prevented and sets this exception bit. The following condition also sets the Reassembly Corrupted bit: ■ enqueue or transmit with {Length, LengthBitOffset} parameters AND ■ { WriteOffset, DestBitOffset} parameters greater than or equal to ((64 K-1) * 8) If any of the parameters have specially interpreted default values, use the interpreted value. This means when **length** and **lengthBitOffset** are zero, the PAB uses the current reassembly length as the {**length**, **lengthBitOffset**} value.
8	Enqueue and Memory Allocation Error	System is out of memory.
9	Time-out	A reassembly timed out.
10	Sizeout	A reassembly exceeded its **sizeoutThreshold** parameter.
11	Timer Start Failure	A PAB request sent to the Timer Manager engine failed. **NOTE** A Timer Start Failure exception has a higher priority than exceptions encoded ten through one.
12-15	Reserved	

When the PAB encounters multiple exceptions, it reports *only* the highest priority exception. It reports the `Timer Start Failure` exception (priority #11 exception), followed by exceptions with priorities ten through one.

The following table shows the information corresponding to the bit encoding of the PAB CDB **State** output parameter.

Table 9 PAB Output Parameters Connection Database Reassembly State Encoding

Bit(s) of Reassembly State Field	Connection Database Information
31	Active – Set for live reassembly. Zero means all remaining information is undefined.
30:27	Spare
26:24	Priority
23	Spare
22	Sequence number check done for reassembly
21	Discard reassembly when time-out happens
20	Timer stopped
19	Spare
18:16	Bit length – Specifies the number of bits beyond the number of bytes in the total reassembly length contain actual data You can encode reassembly offsets as bytes, bits, or bytes plus bits. You also can encode the length in the same manner; the total (precise) length of the reassembly does not need to be an even byte multiple.
15:0	Length (expressed in bytes). This represents the highest enqueued byte of the reassembly.

Output Parameters For getStatus PAB Commands

When the PAB engine transmits a task because of an explicit `getStatus` command, the PAB engine outputs a task with no packet data. Although the task does not contain packet data, it does contain PAB resource and usage and state information in specific formats. The parameter format of the output task depends on which of the three `getStatus` commands were selected. The following three tables displays the formats for the output tasks generated for each of the `getStatus` commands.

Table 10 Output Task Parameters for getstatus_reasmState Command

Field	Size	Description
Status	1B	Status as defined in the Status Byte Encoding table.
Reassembly Index	3B	Reassembly index specified by the task
CDB State	4B	Same format as the PAB Output Parameter CDB State Encoding table.
Sequence Number	1Bt	Sequence number for this reassembly
Reserved	1 bit	Reserved
Timer Generation Field	3 bits	CDB state for this reassembly
Reserved	2 bits	Reserved
Timestamp	3B	Axxia time this entry was written to CDB
Reserved	4B	Reserved

Table 11 Output Task Parameters for getstatus_reasmPrioMem Command

Field	Size	Description
Status	1B	Status as defined in the Status Byte Encoding table.
Reassembly Index	3B	Reassembly index specified by the task
Spare	3 bits	Spare
Priority Memory Usage	29 bits	PAB memory usage (256-byte units) for the indicated reassembly priority.
Spare	3 bits	Spare
Total Memory Usage	29 bits	PAB memory usage (256-byte units) total for all priorities.
Reserved	4B	Reserved

Table 12 Output Task Parameters for getstatus_prioMem Command

Field	Size	Description
Spare	35 bits	There is no reassembly or CDB-specific state to report.
Priority Memory Usage	29 bits	PAB memory usage (256-byte units) for the priority specified by the PAB input task parameter.
Spare	3 bits	Spare
Total Memory Usage	29 bits	PAB memory usage (256-byte units) total for all priorities.
Reserved	4B	Reserved

To configure the output task parameters for non-`getStatus` commands in the ASE, see the *Output Parameters For non-getStatus PAB Commands* heading in the *Output Task Parameters* section for the PAB chapter of this document.