AMBA® ATP Engine

Guide

ARM IHI 0092A (ID051320)

AMBA ATP Engine
Guide

Release Information

The following changes have been made to this guide:

Change history

Date	Issue	Confidentiality	Change
15 May 2020	A	Non-Confidential	First release

Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.
LES-PRE-20349

Confidentiality Status

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
AMBA ATP Engine Guide

Preface

This preface introduces the *AMBA ATP Engine Guide*. It contains the following sections:

About this guide

Software architecture and functional description for the AMBA ATP Engine.

Intended audience

This guide is written for hardware and software engineers who want to understand the inner workings of the AMBA ATP Engine and learn how to integrate it into third-party applications.

Using this guide

Chapter 1 ***Introduction*** Introduces the AMBA ATP Engine guide.

Chapter 2 ***Traffic Profiles*** Details the components of the AMBA ATP Engine.

Chapter 3 ***Events, Logging, and Statistics*** Describes ATP Events that can be triggered by a Traffic Profile and propagated by the Traffic Profile Manager to other Profiles which are subscribed to it.

Chapter 4 ***Adapters*** Describes the use of adapters to allow the AMBA ATP Engine to be integrated into a host platform.

Appendix A ***AMBA ATP Engine Configuration*** Defines a standard format for the ATP Engine configuration file. It is structured in a human-readable and machine-readable format.

Conventions

The following sections describe conventions that this guide uses:

- *Typographic conventions*
- *Numbers*

Typographic conventions

The typographical conventions are:

italic	Highlights important notes, introduces special terminology, and denotes internal cross-references and citations.
bold	Denotes signal names, and is used for terms in descriptive lists, where appropriate.
`monospace`	Used for assembler syntax descriptions, pseudocode, and source code examples. Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.
SMALL CAPITALS	Used for a few terms that have specific technical meanings.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. Both are written in a `monospace` font.

Additional reading

This section lists relevant publications.

Arm publications

See the Infocenter http://infocenter.arm.com, for access to Arm documentation.

- *AMBA Adaptive Traffic Profiles Specification* (ARM IHI 0082)
- *AVL AXI XVC Vector Format* (ARM IHI 0030)
- *SNOOPy Calendar Queue* (10.1109/WSC.2000.899756)

Other publications

- *The gem5 simulation system*, http://www.gem5.org

Feedback

Arm welcomes feedback on its documentation.

Feedback on this guide

If you have comments on the content of this guide, send e-mail to `errata@arm.com`. Give:

- The title, AMBA ATP Engine Guide
- The number, ARM IHI 0092A
- The page number(s) that your comments apply
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the AMBA ATP Engine guide:

1.1 About AMBA ATP Engine guide

The AMBA ATP Engine is a platform-independent module that generates synthetic traffic, as defined in the ATP specification. It can be plugged into event-driven or time-driven software modeling, simulation, and testing platforms, such as gem5, using a simple API mechanism. See *gem5 adapter* on page 4-41.

The AMBA ATP Engine provides a framework that is designed to generate synthetic traffic in a simple way. It uses standardized configuration files that can be shared across teams and is both human and machine readable.

The AMBA ATP Engine module is platform-independent that can be plugged into a host tool or run as a standalone tool.

1.2 ATP framework

The process of using the ATP framework is:

- Configure .atp files
- Execute the simulation
- Collect the resulting statistics

The AMBA ATP Engine module supports configuration of any number of ATP masters and slaves by using external profile files. Profiles elements in these files define the ATP traffic, which uses a FIFO buffer model. A master can have one or more Profiles. Slaves are defined by a single Profile.

Executing the simulation can be done through a host platform or be set up as a standalone program.

Chapter 2
Traffic Profiles

This chapter describes the components of the AMBA ATP Engine:

2.1 Traffic Profile Manager

A Traffic Profile Manager (TPM) is an entity that manages all the configured Traffic Profiles and arbitrates their activation and deactivation. The TPM requests and delivers packets from and to Traffic Profiles. In addition, the TPM relays events to and from subscribing Traffic Profiles.

Figure 2-1 shows an example of a simple TPM embedded in a host system.

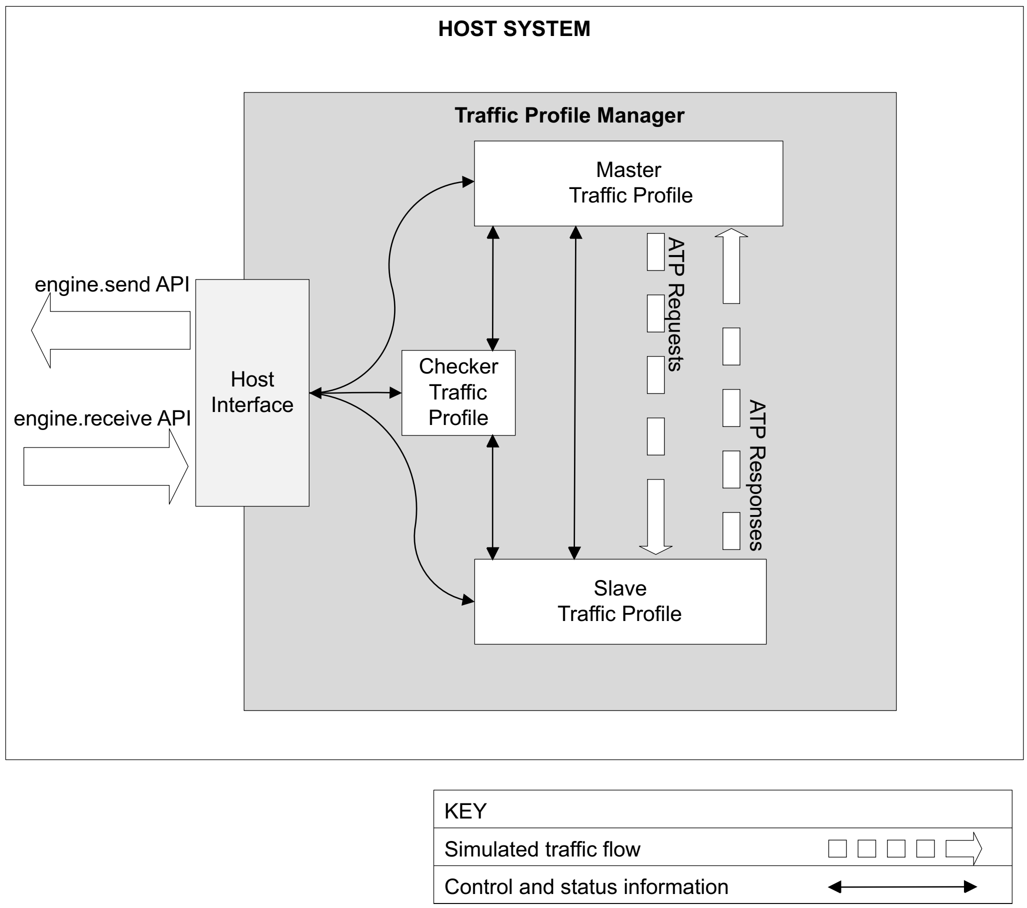

Figure 2-1 Example Traffic Profile Manager simulation

2.1.1 TPM role

Host platforms connect to the TPM in the AMBA ATP Engine. The TPM exposes several APIs that allow host platforms to:

- Send packets
- Receive packets
- Configure specific features
- Retrieve global and per-master statistics

2.1.2 Host interface

The TPM implements the interfaces that enable the host platform to request and send packets to the AMBA ATP Engine.

The Host Interface defines two APIs to send and receive information.

`send` The `send` API is called by the host platform to request packets from the AMBA ATP Engine.

`receive` The `receive` API is used to convey packets to the AMBA ATP Engine, which are then to be delivered to its profiles.

The host interface functions use stream operations. A Stream is defined as a buffered sequence of packets defined by a Traffic Profile on a *First-In First-Out* (FIFO) basis. The packets are chained through `wait_for` on TERMINATION events.

Table 2-1 shows the host interface operations.

Table 2-1 Traffic Profile Manager operations

Operation	Description
Address Stream Reconfiguration	Allows reconfiguring base and address range of a stream. The parameters for Read and Write operations from the same Stream can be configured independently.
Address Stream Reset	Allows a Stream to be reset to its initial state. The reset can also occur after termination of its profiles. All Traffic Profiles of the Stream are restarted and traffic generation proceeds as if the profiles were in their initial state.
Address Stream Termination query	Interface to query the TPM about the termination of a stream. The interface then returns the state of all the profiles of a stream and whether they have terminated.

2.2 Traffic Profile classes

Traffic Profiles are the entities that generate packets, which are defined by their ATP FIFO configuration. See *ATP FIFO* on page 2-23.

Traffic Profiles also receive responses to the packets generated. Several kinds of traffic profile entities are implemented in the AMBA ATP Engine:

Master Traffic Profile

The Master Traffic Profile implements a traffic profile that sends packet requests and receives packet responses. One or more traffic profiles, each with their corresponding FIFO, can be combined to implement a specific master behavior. The FIFOs can be in any combination of series and parallel operation.

Checker Traffic Profile

The ATP Checker records all requests and responses that are generated or received by the profile it is monitoring. It stores this information in its own ATP FIFO.

If the profile being monitored has FIFO underrun or overrun detection, it returns an error. The error indicates that the rate that requests and responses are recorded does not match configured FIFO rate.

Delay Traffic Profile

The ATP Delay Traffic Profile implements a delay block. The profile can be used to force a pause between two traffic profiles. Its behavior is to stay active for the configured amount of time, sending no packets and accepting no responses. At end of the configured delay, it terminates.

Slave Traffic Profile

The Slave Traffic Profile implements a fixed latency and bandwidth memory slave. This memory slave receives requests from, and sends responses to, its registered masters. If all ATP masters are registered to at least one ATP slave, the AMBA ATP Engine can be run as a standalone executable.

Figure 2-2 shows Traffic Profiles inheritance.

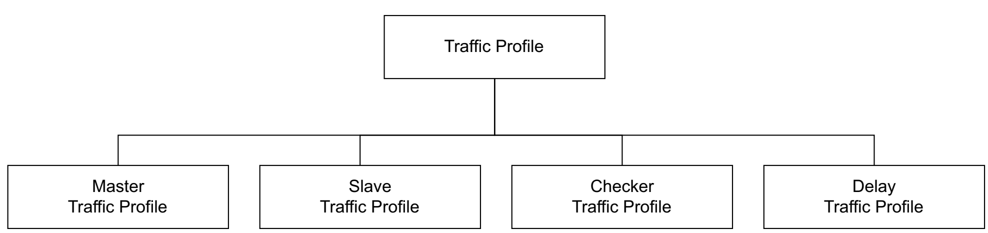

Figure 2-2 Traffic Profiles inheritance

2.2.1 Master Traffic Profile

Each Master Traffic Profile contains a Packet Descriptor object, which describes how its ATP Packets are to be generated. Several options for both address and packet size are available:

Address generation options:

- Start from a base value and generate with a configured linear increment
- Start from a base value and generate with two-dimensional increments (increment and stride)
- Random uniform address generation between configured minimum and maximum values
- Normal distribution address generation around a configured mean with a configured standard deviation

Packet size generation options:

- Constant configured value
- Random uniform size generation between configured minimum and maximum values
- Normal distribution size generation around a configured mean with a configured standard deviation

Addresses and packet sizes can also be required to be aligned by enabling the natural alignment feature. It is also possible to specify that either no-alignment or fixed-alignment used. See *Address Generation* on page 2-25.

A Master Traffic Profile also includes an ATP FIFO model. The ATP FIFO model shapes the data produced or consumed by the profiles by configuring the FIFO fill and FIFO depletion rate levels. This configuration allows setting a maximum buffered data quantity equal to the configured FIFO maximum size. ATP FIFOs are in *ATP FIFO* on page 2-23.

A Master Traffic Profile can be in one of the following states:

Terminated The profile is terminated, it has depleted all data it was scheduled to send, and has no outstanding transactions in flight which are waiting for response.

When a Master Traffic Profile sets its status to Terminated, it emits an ATP Event. This event signals that its state is Terminated. This terminated status persists until the end of the simulation, unless re-activated (reset) by the TPM. See *ATP events* on page 3-28.

Locked The profile is not terminated, and is either:

- Waiting for responses to unlock its outstanding transactions
- Waiting for responses and has terminated all its data scheduled to be sent
- Waiting for another profile to terminate
- Halted is a condition of complete lock, triggered by linking this profile to another. Linking is accomplished by switching between halted and non-halted status on the other profile being locked or unlocked.

Active The profile is ready to generate data. The Master Traffic Profile is active if it is not Locked and not Terminated.

2.2.2 Checker Traffic Profile

A Checker Traffic Profile is configured with an ATP FIFO just like a Master Traffic Profile. It is also set to monitor one or more Master Traffic Profiles. When one of its monitored Master Traffic Profiles sends or receives a Packet, the TPM invokes the `send` or the `receive` function on the Checker Traffic Profile with the same packet. The Packet being sent to the Checker Traffic Profile its ATP FIFO status level and Outstanding Transactions (OT) to change.

When the Checker Traffic Profile records an ATP FIFO underrun or overrun, one or more profiles do not match its configured ATP FIFO fill or depletion rate.

A Checker Traffic Profile can be in one of the following states:

Terminated The profile is terminated if no longer active.

When a Checker Traffic Profile changes its status to Terminated, it emits an ATP Event signaling the termination. The Check Traffic Profile persists in the terminated state until the end of the simulation, unless re-activated (reset) by the TPM. See Chapter 3 *Events, Logging, and Statistics*.

Active One or more of the monitored Master Traffic Profiles is not terminated yet.

2.2.3 Delay Traffic Profile

A Delay Traffic Profile configuration is a single latency value, expressed in seconds or SI unit of time. See *Supported SI and IEC Standard Rate Units* on page A-44.

On activation, the Delay Traffic Profile initializes a counter and then reports its expiration time at the next transmission time. This report causes the TPM to query the Delay Traffic Profile again at its expiration time. The query from the TPM will then terminate.

The Delay Traffic Profile is normally used to trigger another profile activation by making that profile wait on the Delay Traffic Profile termination event.

A Delay Traffic Profile can be in one of the following states:

Terminated The profile configured delay is expired, counting from its first activation time

When a Delay Traffic Profile switches its state to Terminated, it emits an ATP Event, signaling its termination. The Delay Traffic Profile persists in its terminated state until the end of the simulation, unless re-activated (reset) by the TPM. See *ATP events* on page 3-28

Active The Delay Traffic Profile is not waiting for any other profile to terminate. The first activation time is recorded and used to compute its termination time as first activation time plus configured delay

2.2.4 Slave Traffic Profile

Slave Traffic Profiles simulate memory slaves that receive requests and send responses to its registered Master Traffic Profiles. When all ATP Masters are registered to at least one ATP Slave, the AMBA ATP Engine can be run as a standalone executable.

A Slave Traffic Profile can be configured with:

Rate The maximum service rate that the Slave Traffic Profile can serve incoming requests

Latency Request-to-response latency

Max OT Maximum number of parallel memory accesses supported

Width Width of the memory in bytes, which determines how many accesses a request corresponds to

Master ATP Master or Masters associated with this memory slave

Slave Traffic Profiles include an ATP READ FIFO model, which is used to regulate the rate at which requests are served. On a request reception, the ATP FIFO is queried using its `send` API. If the FIFO can accommodate the request, an amount of data equal to the request size is marked as "in-flight". This marking signals that the request is accepted and queued, waiting to be processed.

When the Slave Traffic Profile is requested to send responses, it calls the `receive` API on its FIFO for an amount of data corresponding to the sizes of the responses about to be issued. In-flight data in the FIFO is marked as received and can be consumed by the FIFO drain rate. The drain rate allows more requests to be served.

A Slave Traffic Profile can be in one of the following states:

Locked A Slave Traffic Profile has reached its maximum OT. A slave that is Locked cannot accept any more requests or the Slave Traffic Profile has no responses queued to be sent

Active A Slave Traffic Profile is Active if it is not locked

2.3 ATP FIFO

The ATP FIFO model tracks data either written or read by a Master or Slave device. The model constrains that data to the size of the FIFO buffer. The constraints are:

- The maximum number of outstanding transactions allowed
- The data consumption or production rate

The ATP FIFO exposes two main APIs: `send` and `receive`. When one of those APIs is called, the FIFO also runs an update function. The update function recomputes the FIFO fill level based on its configured rate and elapsed time since last call.

Figure 2-3 shows the flow of the ATP FIFO.

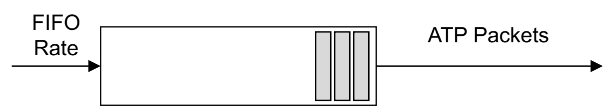

Figure 2-3 ATP FIFO

2.3.1 ATP FIFO update

The Update function computes the FIFO level update, based on the configured rate and the elapsed time since the last update call. The result can be a non-integer number. The Update function uses a "carry" function to save decimal leftovers. The carry function adds any decimal part to a store and truncates the FIFO level update to an integer value. When the carry value reaches 1, the Update function adds 1 to the current level update and the carry value is reset.

When the FIFO level update is computed, the Update function checks whether an underrun or overrun occurs. The update function checks by comparing the level update with the current FIFO fill level:

- If the ATP FIFO is a READ FIFO, then an underrun occurs when the current fill level is less than the computed level update.
  - If an underrun occurs, the FIFO level is reset to zero.
  - If an underrun does not occur, then the FIFO level update is subtracted (drain) from READ FIFO level.
- If the ATP FIFO is a WRITE FIFO, then an overrun occurs when the current fill level plus the computed level update is greater than the configured maximum fill level.
  - If an overrun occurs, the FIFO level is capped to its configured max level.
  - If an overrun does not occur, then the FIFO level update is added (fill) to WRITE FIFO level.

2.3.2 ATP FIFO send

Each time the ATP FIFO `send` API is invoked, a FIFO Update is triggered. After the Update, the `send` API checks whether is possible to accommodate for a specific data request in the FIFO, based on the FIFO type:

Read FIFO The `send` function checks if it is possible to allocate space in the FIFO to read the requested amount of data. If the current level (including any in-flight data) plus the requested data is less than the configured maximum level, then the allocation is successful.

Write FIFO The `send` function checks if it is possible to remove data from the FIFO to write the requested amount. If the current level (minus any in-flight data) is greater than the requested amount, then the removal is successful.

The `send` function also tracks of the outstanding transactions, which is the data that is in-flight the `receive` function has not been called yet.

In addition to returning the success status of the requested operation, the `send` function also computes and returns the next time a request for the same amount of data will be possible to be served. This computation is done by considering the current fill level, the maximum configured level, and the FIFO update rate.

2.3.3 ATP FIFO receive

When the `receive` API is invoked, the Update function is called. The `receive` function records the acknowledgment of data previously marked as in-flight with a `send` function call.

The outstanding transaction counter is decremented, and a specified amount of data is removed from the ATP FIFO in-flight data counter. That amount is:

- Added to the ATP FIFO level, if it is ATP READ FIFO
- Removed from the ATP FIFO level, if it is ATP WRITE FIFO

2.3.4 ATP FIFO tracker queue

The ATP FIFO Tracker Queue is an optional feature that is included in the AMBA ATP Engine. Its purpose is to give a latency measure based on how the FIFO-achieved rate differs from the configured drain or fill rate.

Tracker Queue latency is the difference between the successful issue time and the expected issue time. The successful issue time is the amount of time the `send` API takes to get the amount of data requested. The expected issue time is the amount of time the FIFO is configured to take to consume or produce that amount of data.

Figure 2-4 shows the sequence that the ATP Tracker Queue performs its function.

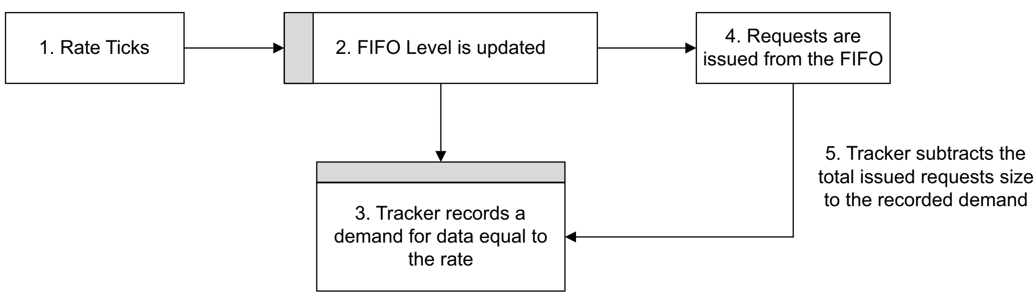

Figure 2-4 ATP Tracker Queue

2.4 Packet descriptor

The Packet Descriptor (PacketDesc class) implements methods to generate packets with specific patterns both for addresses and sizes. When random value generation is required to generate one or both the values, an object of the RandomGenerator class is also used.

The Packet Descriptor also includes a tagger object (PacketTagger) which is used to fill other fields of the generated packets such as local and global IDs.

2.4.1 Address Generation

Address generation can be configured as:

- A linear sequence of addresses with a base plus a fixed increment value (Linear)
- Following one of the patterns available in the Random Generator (Random)

The address generation can be configured with an optional range over which the generated address stream wraps around. In both cases, optional Stride configuration can be provided, which is composed of three parameters:

N Number of addresses that should be included in a Stride

Increment Increment to be applied to the addresses of the Stride

Range Range whose end the Stride addresses that should terminate at

If the Stride parameters are configured, the address generation process:

- Generate one address, using the configured method (Linear or Random).
- Generate N-1 subsequent addresses:
  - Increment the previously generated address by the quantity Increment.
  - If the end of the Stride range is reached, then stop generating addresses
  - If the end of the Stride range is not reached, continue generating subsequent addresses

Address alignment can be optionally configured, either to a specific value (must be a power of two), or set to natural.

If a value is provided, generated addresses are masked out so that they are bit-aligned to that value. See *Pattern configuration* on page A-48.

In natural alignment, a packet address is aligned to the next power of its size.

2.4.2 Size Generation

Size generation of the Packet Descriptor can be configured as a constant value, or as following one of the patterns available in the Random Generator (Random).

2.4.3 Random Generator

The Random Generator class supports two random number distributions, both based on the Mersenne Twister 64-bit generator.

Uniform Uniformly distributed random numbers that are bounded by the configured min and max values

Normal Normal distributed (Gaussian) random numbers, with configured average value `mean` and standard deviation `dev`.

2.4.4 Packet Tagger

The Packet Tagger assigns values to packet fields other than address and size:

UID Shared across the entire AMBA ATP Engine. UID is set by the TPM on all outgoing packets and uniquely identifies a packet in the AMBA ATP Engine system, and any connected adapters.

ID Can be configured with global min/max bounds, or min/max values local to a Traffic Profile.

Chapter 3
Events, Logging, and Statistics

This chapter describes ATP Events that can be triggered by a Traffic Profile and propagated by the TPM to other Profiles which are subscribed to it.

3.1 ATP events

ATP Events can be triggered by a Traffic Profile and propagated by the TPM to other Profiles that are subscribed to the ATP Event. ATP Events signal that something happened to the destination profile at the current ATP time.

Table 3-1 shows all ATP Events with their meaning.

Table 3-1 ATP events

Event	Definition
NONE	Empty event. Has no effect.
ACTIVATION	A profile has become active for the first time
TERMINATION	A profile has issued all the transactions it was configured to issue
FIFO_EMPTY	The FIFO of a profile has become empty
FIFO_FULL	The FIFO of a profile has become full
FIFO_NOT_EMPTY	The FIFO of a profile becomes non-empty after being empty
FIFO_NOT_FULL	The FIFO of a profile becomes non-full after being full
PROFILE_LOCKED	A profile is unable to send data
PROFILE_UNLOCKED	A profile can send data after being unable to
PACKET_REQUEST_RETRY	A rejected request is due to be retried
TICK	Special clock event, triggers TPM tick

All Traffic Profiles derive from the EventManager class, which implements methods to generate and receive events.

When a Traffic Profile subscribes to an Event, the event is added to a wait list in the TPM. When a Traffic Profile generates an Event, it is checked by the TPM against the wait list. The TPM then propagates the Event to all other Traffic Profiles waiting for it.

3.1.1 Master Traffic Profile Events

Table 3-2 shows the Events that a Master Traffic Profile can emit.

Table 3-2 Slave Traffic Profile Events

Event	Condition
ACTIVATION	Emitted when: • The Traffic Profile is created, if it is not configured to wait for any TERMINATION Event, • The `active` API is called, if the API detects that the profile becomes active for the first time.
TERMINATION	Emitted when the `active` API is called, if it detects that all scheduled data has been sent and there are no outstanding transactions left.
PROFILE_LOCKED	Emitted at the end of a `send` API call if the Traffic Profile was unable to accommodate the request.
PROFILE_UNLOCKED	Emitted at the end of a `send` API call if the Traffic Profile could accommodate the request.

Table 3-3 shows the Events that the Master Traffic Profile can subscribe to.

Table 3-3 Master Traffic Profile Event subscriptions

Event	Condition
ACTIVATION	A Master Traffic Profile can wait on: • One or more other Traffic Profile ACTIVATION events before starting to issue transactions. • Activation that is externally injected into the Traffic Profile Manager by using its `event` API.
TERMINATION	A Master Traffic Profile can wait on one or more other Traffic Profile TERMINATION events before starting to issue transactions
PROFILE_LOCKED	A Master Traffic Profile can wait on one or more other Traffic Profiles PROFILE_LOCKED events before issuing transactions. The Traffic Profile Master then starts waiting for a PROFILE_UNLOCKED event and will lock again after the PROFILE_UNLOCKED event occurs.
PROFILE_UNLOCKED	A Master Traffic Profile can wait on one or more other Traffic Profiles PROFILE_UNLOCKED events before issuing transactions. The Traffic Profile Master then starts waiting for a PROFILE_LOCKED event and will lock again after the PROFILE_LOCKED event occurs.

3.1.2 Delay Traffic Profile Events

Table 3-4 shows the Events that a Delay Traffic Profile can emit.

Table 3-4 Delay Traffic Profile Events

Event	Condition
ACTIVATION	Emitted when: • The Traffic Profile is created, if it is not configured to wait for any TERMINATION Event • When its `active` API is called and the Traffic Profile is detected to be active for the first time
TERMINATION	Emitted when the Traffic Profile is detected as expired on its `active` API call. The expiration is defined as when the current time is equal to or past the Traffic Profile deadline. The Traffic Profile deadline is equal to the activation time plus the configured delay.

Table 3-5 shows the Events that the Delay Traffic Profile can subscribe to.

Table 3-5 Delay Traffic Profile Event subscriptions

Event	Condition
ACTIVATION	A Delay Traffic Profile can wait one or more other Traffic Profile ACTIVATION events before starting issuing transactions.
TERMINATION	A Delay Traffic Profile can wait one or more other Traffic Profile TERMINATION events before starting issuing transactions.

3.1.3 Checker Traffic Profile Events

Table 3-6 shows the Events that a Checker Traffic Profile can emit.

Table 3-6 Checker Traffic Profile Events

Event	Condition
TERMINATION	It is emitted when the Traffic Profile is no longer waiting for the TERMINATION of any of its monitored profiles.

Table 3-7 shows the Events that a Checker Traffic Profile can subscribe to.

Table 3-7 Checker Traffic Profile Event subscriptions

Event	Condition
TERMINATION	A Checker Traffic Profile waits on the TERMINATION events of all its monitored profiles

3.1.4 Slave Traffic Profile Events

Table 3-8 shows the Events that a Slave Traffic Profile can emit.

Table 3-8 Slave Traffic Profile Events

Event	Condition
ACTIVATION	A Slave Traffic Profile emits the activation signal when it is first created.

A Slave Traffic Profile does not subscribe to any events.

3.2 Kronos

The Kronos is a *Future Event List* (FEL) system for the AMBA ATP Engine. It is available to the TPM for scheduling Events and retrieving the next event tick to set the ATP clock to.

Events in Kronos are organized using a calendar queue, which is a fast-lookup data structure specialized in handling timelines.

Kronos enables the AMBA ATP Engine to be run as a standalone executable and in adapter-driven mixed memory mode.

3.2.1 Standalone Execution

When the AMBA ATP Engine is compiled as a standalone executable, it can load ATP files and run them against an internal slave. The standalone executable is named “atpeng”, by default.

This internal slave is configured with:

- No maximum OT limit
- Latency and rate values either taken from:
  - The command line
  - Set to the defaults: 80ns and 32GB/s

The AMBA ATP Engine executable command-line syntax is as follows:

```
./atpeng [.atp file]+ <rate> <latency>
```

If any of the ATP files listed in the command line also contain one or more ATP Slaves, those slaves will also be instantiated with the default. Any master already associated with such slaves defined in the ATP files will not be associated with the default slave.

The AMBA ATP Engine simulates a system where all masters are connected to their associated slaves with an ideal link. The execution of the simulation ends when the last configured packet is successfully responded to.

3.2.2 Mixed Mode Execution

When ATP is compiled with a host platform, it supports ATP configured internal slaves running alongside any host platform defined slave memory systems.

Any master not associated with internal slaves is associated with the host platform by default. All its requests are routed to the platform adapter and incoming responses will be delivered to it.

Mixed Mode Execution is triggered when at least one internal and one external slave are both present.

The AMBA ATP Engine implements an internal routing mechanism to serve packets to internal slaves. This mechanism is triggered when its send and receive API are called by the driving adapter. The Mixed Mode execution is transparent to the adapter and no special handling is required.

3.3 Logging

Logging in ATP is delegated to the `Logger` class. The `Logger` follows the singleton design pattern, and as such it provides a get API to access and interact with the only instance of the `Logger` class.

The `Logger` class outputs formatted strings to a configured ostream object, which is set to cout by default.

The `Logger` defines various logging verbosity levels, of increasing priority. The `Logger` can be configured with one of such levels, causing all log messages logged with inferior priority not to be output to the internal ostream object.

The `Logger` logging levels are, in increasing priority order:

1. DEBUG_LEVEL
2. WARNING_LEVEL
3. ERROR_LEVEL
4. PRINT_LEVEL

The `Logger` header file also defines some macro functions which log a message with predefined priority:

1. LOG
2. WARN
3. ERROR, which also causes the AMBA ATP Engine or platform to terminate with `exit(1)`
4. PRINT

3.4 Statistics

Traffic Profile Descriptors include a `Stats` object which logs metrics when data transmission or reception occurs for a profile. The TPM can aggregate `Stats` objects by master on request, by calling its `getMasterStats` API.

Table 3-9 shows the metrics that are collected for each Traffic Profile.

Table 3-9 Collected Metrics

Name	Description	Unit
`Start`	Traffic Profile Activation time	Seconds
`Time`	Traffic Profile Termination time or Statistics collection time, whichever occurred first.	Seconds
`Sent`	Number of ATP Packets sent	-
`Received`	Number of ATP Packets received	-
`Data Sent`	Total data sent	Bytes
`Data Received`	Total data received	Bytes
`Latency`	Packet Request to Response average latency	Seconds
`Jitter`	Inter-Request Jitter is computed iteratively, based on RFC 1889 January 1996 `J=J+(|D (n-1, n) |-J)/16`	Seconds
`Send Rate`	Computed as `Data Sent`/`Time`	Bytes/Seconds
`Receive Rate`	Computed as `Data Received`/`Time`	Bytes/Seconds
`Underruns`	Total ATP FIFO underruns	-
`Overruns`	Total ATP FIFO overruns	-
`OT`	Average number of outstanding transactions	-
`FIFO Level`	Average FIFO fill level	Bytes

Chapter 4
Adapters

This chapter describes the use of adapters to allow the AMBA ATP Engine to be integrated into a host platform:

4.1 Adapters overview

The AMBA ATP Engine can be integrated into a host platform to use the memory system of the platform. Using this host platform, the ATP Master is enabled to send memory requests to the host platform and receive memory responses from it.

To integrate the AMBA ATP Engine into the target platform, an adapter layer is required. The adapter is an interface between the platform APIs and the AMBA ATP Engine APIs.

4.2 Adapter Integration API

The integration procedure is described in this section. A gem5 adapter is provided with the official ATP distribution. gem5 adapter integration is detailed in *gem5 adapter* on page 4-41.

Table 4-1 shows the AMBA ATP Engine APIs that are available to the host platform adapter to set AMBA ATP Engine options.

Table 4-1 Adapter Integration APIs

API function	API	Purpose
Initialization	`Load`	Loads an ATP file into the AMBA ATP Engine
Control	`enableProfilesAsMasters`	Special mode where all ATP Profiles are treated as single masters
	`enableTrackerLatency`	Enables the ATP Latency Tracker
	`setTime`	Sets the ATP Time
Stream Management	`addressStreamReconfigure`	Reconfigures address ranges of a chain of profiles linked by termination `waited_for` events. The reconfiguration is calculated from the base and range passed, so that they fit the new address space. The calculation starts from the root profile provided.
	`streamReset`	Resets the stream for a profile. Looks up a list of profiles linked by TERMINATION event and issues them the reset command. *Experimental*.
	`streamTerminated`	Checks if a profiles stream is terminated. Traverses a profiles stream hierarchy and returns true only if all profiles of the stream have terminated. *Experimental*.
	`uniqueStream`	Gets a unique reference to a stream. If the requested stream is already in use, this function will clone the stream. *Experimental*.

Table 4-1 Adapter Integration APIs (continued)

API function	API	Purpose
Status	`getTimeResolution`	Returns the AMBA ATP Engine time resolution See *Supported Time Units* on page A-45
	`getMasters`	Returns a set of all configured master names
	`isTerminated`	Returns True if all packets for master *m* have been transmitted
	`waiting`	Returns True if the TPM is waiting for packet responses
	`getMasterStats`	Returns a Stat object for master m containing a snapshot of all its ATP stats computed at the current ATP time
	`getMasterSlaves`	Returns a map with master to internal slaves pairings
Kronos Control	`loop`	Run the internal routing loop, which dispatches packets between internal masters and internal slaves
Packet transmission	`send`	Returns packets generated by active ATP masters with external (host platform) system destinations.
	`receive`	Routes packets generated by the host platform to ATP Profiles

4.3 Adapter Integration Flow

To integrate the AMBA ATP Engine into a host platform, it is necessary to follow the code flow:

1. Configure a Traffic Profile Manager object in the adapter by using the control APIs
2. Load ATP input files in the Traffic Profile Manager object using the `load` API
3. Get the time resolution from the Traffic Profile Manager, which is determined based on the loaded ATP files
4. Start the interaction with the AMBA ATP Engine by calling the `send` API
5. Route any Packets available to the AMBA ATP Engine by calling the `receive` API. This call can be triggered asynchronously when a packet to be sent to the AMBA ATP Engine is available

4.4 Adapter design guidelines

The interactions between the adapter and the AMBA ATP Engine should happen following the guidelines:

- The Traffic Profile Manager object should be embedded into an adapter layer which bridges the AMBA ATP Engine and the host platform.
- The AMBA ATP Engine should exchange information with the adapter layer in the form of Google Protocol Buffer objects. ATP Packets and Stats are Google Protocol Buffer objects and should be exchanged as such.
- Where conversion from ATP Packet format to host packet format is needed, that should be implemented in the adapter layer.
- Any additional information that is required to merge the ATP Packet stream into the host platform, must be resolved at the adapter layer.
- The AMBA ATP Engine relies on the host platform to provide time ticks. Discrete time is acceptable, but non-increasing time ticks are not.
- The host platform should detect whether the AMBA ATP Engine has exhausted all its traffic and it is no longer waiting for packets from the host.

When the TPM does not signal a next transmission time through its `send` API, the cause could be:

- The TPM could be locked waiting for packets from the host.
- The TPM could have depleted all its scheduled traffic.

Checking the TPM status with the `waiting` or `isTerminated` API is needed in this case to ascertain whether AMBA ATP Engine has terminated.

4.5 gem5 adapter

ProfileGen is an adapter for the gem5 simulator that is implemented as a MemObject derived class. `ProfileGen` encapsulates the Traffic Profile Manager and allows gem5 to send and receive memory request and response packets from the AMBA ATP Engine.

`ProfileGen` connects to other gem5 objects by instantiating a configurable number of master ports that are dedicated to the individual ATP masters. `ProfileGen` sends and receives packets belonging to such masters.

It also implements a local packet buffer used to store ATP packets when their associated master ports are unavailable for transmission to the gem5 connected system.

4.5.1 Usage

`ProfileGen` is to be connected to the system by using its master ports. The typical way to do so is to directly connect its master ports to an XBar.

`ProfileGen` can be used as a reference for integrating the gem5 adapter into gem5 scenarios.

4.5.2 Configuration Parameters

Table 4-2 gem5 configuration parameters

Parameter	Definition
`config_files`	Comma-separated list of ATP files
`exit_when_done`	Flag to enable the simulation to terminate when all ATP masters have depleted their packets
`exit_when_one_master_ends`	Flags to enable the simulation to terminate when one ATP master depletes its packets
`trace_atp`	Flag to enable tracing ATP-generated packets
`trace_atp_file`	File name where to store the ATP-generated packets trace
`trace_gem`	Flag to enable gem5 generated packets in time
`trace_gem_file`	File name where to store gem5 generated packets trace
`trace_m3i`	Flag to enable tracing of ATP-generated packets in m3i format [1.4, 3]. M3i traces are generated per master and named after them
`trace_m3i_bus`	Bus width to be used to fill the m3i file trace
`out_of_range_addresses`	Flag to allow the reception of packets with out-of-range addresses from the AMBA ATP Engine
`uid_routing`	Flag to enable fast routing based on ATP UID packet field
`profiles_as_masters`	If set, all Traffic Profiles are treated as single masters
`tracker_latency`	Flag to enable the ATP Latency Tracker queue

Appendix A
AMBA ATP Engine Configuration

This appendix defines a standard format for the ATP Engine configuration file. It is structured in a human-readable and machine-readable format.

A.1 Configuration file structure

The configuration file syntax follows the Google Protocol Buffer text file format. Each object is enclosed in curly brackets {} and properties of the object are specified with the property: value syntax.

The configuration file is organized as a flat list of Traffic Profiles, preceded by a global configuration section defined in *Global Configuration* on page A-46.

Each Traffic Profile includes:

- A FIFO Configuration block that specifies the parameters of the AMBA Traffic Profile FIFO
- A Pattern Configuration block that specifies what kind of packets and how the Traffic Profile should generate them

See *FIFO configuration* on page A-47 and *Pattern configuration* on page A-48.

A.1.1 Field type specifiers

Optional Specifying a value for this field can be omitted.

Repeated This field can be specified multiple times, it is then processed as array of values.

Required Specifying this field value is mandatory.

A.1.2 Supported SI and IEC Standard Rate Units

Table A-1 shows the rate unit specifiers that can be used:

Table A-1 Unit specifiers

Specifier	Definition
TB/s	10^{12} Bytes/s
GB/s	10^{9} Bytes /s
MB/s	10^{6} Bytes /s
kB/s	10^{3} Bytes /s
TiB/s	2^{40} Bytes /s
GiB/s	2^{30} Bytes /s
MiB/s	2^{20} Bytes /s
KiB/s	2^{10} Bytes /s
B/s	Bytes /s
Tbit/s	10^{12} bits/s
Gbit/s	10^{9} bits /s
Mbit/s	10^{6} bits /s
kbit/s	10^{3} bits /s
Tibit/s	2^{40} bits /s
Gibit/s	2^{30} bits /s
Mibit/s	2^{20} bits /s
Kibit/s	2^{10} bits /s

A.1.3 Supported Time Units

Table A-2 shows the time units supported in time-based parameters.

Table A-2 Supported Time Units

Unit	Definition
CYCLES	[default]
PS	picoseconds
NS	nanoseconds
US	microseconds
MS	milliseconds
S	seconds

A.2 Configurations

A.2.1 Global Configuration

Table A-3 shows the global configuration parameters can be used to change the AMBA ATP Engine behavior.

Table A-3 Global configuration

Parameter	Type	Requirement	Definition
`lowId`	uint64	Optional	Packet ID low ID value: all packets contain an ID with this value lower bound
`highId`	uint64	Optional	Packet ID high ID value: all packets contain an ID with this value upper bound
`tracing`	bool	Optional	Enables ATP packet tracing to file
`trace_dir`	string	Optional	Specifies the directory name to store ATP packet traces

A.2.2 Traffic profile

Traffic Profiles are identified in the configuration file by the profile tag. Table A-4 shows the fields each profile has.

Table A-4 Traffic Profile

Parameter	Type	Requirement	Definition
`type`	Type	Optional	Profile type, which configures the profile as type 'READ' or 'WRITE'. This affects the FIFO behavior.
`master_id`	string	Required	Master identifier. Multiple profiles can share a master ID.
`fifo`	FifoConfiguration	Optional	Configure the FIFO of a Traffic Profile
`pattern`	PatternConfiguration	Optional	Packets to be generated
`delay`	DelayConfiguration	Optional	Its presence indicates that the profile being configured is a Delay type
`slave`	SlaveConfiguration	Optional	Its presence indicates that the profile being configured is a Slave type
`wait_for`	string	Optional, repeated	Profiles that should complete before this one can activate or events related to the specified profile
`name`	string	Optional	Profile identifier. If it is not set, it will be generated as profile<#>

A.2.3 Delay Configuration

Table A-5 shows Delay Configuration parameters.

Table A-5 Delay Configuration

Parameter	Type	Requirement	Definition
`time`	string	Required	Can be a floating-point value and can be expressed in any of the supported time units. See Table A-2 on page A-45.

A.2.4 Slave Configuration

Table A-6 shows slave configuration parameters.

Table A-6 Slave Configuration

Parameter	Type	Requirement	Definition
`Rate`	string	Required	Maximum rate the slave can process data at. Can be a floating-point value and can be expressed in any of the supported rate units. See *Supported SI and IEC Standard Rate Units* on page A-44.
`latency`	string	Optional	Static latency with which the slave will respond to requests. Can be a floating-point value and can be expressed in any of the supported time units. See *Supported Time Units* on page A-45.
`random_latency`	RandomDesc	Optional	If present, response latencies are randomly generated according to RandomDesc configuration
`random_latency_unit`	string	Optional	If random latencies are configured, determines their time unit: can be expressed in any of the supported time units. See *Supported Time Units* on page A-45. Default to ns.
`TxnLimit`	uint64	Optional	Limit to the number of outstanding transactions that the slave can accept. Defaults to 1
`TxnSize`	uint64	Required	Granularity with which transactions are processed, that is, size of the slave data unit
`master`	string	Repeated	Masters assigned to this slave

A.2.5 FIFO configuration

Table A-7 shows FIFO configuration parameters.

Table A-7 FIFO configuration

Parameter	Type	Units	Requirement	Definition
`Start`	StartupLevel	Enum	Optional	The starting level of the FIFO. Defaults to EMPTY for READ FIFO and FULL for WRITE FIFO
`Start`	StartupLevel	Enum		
`Full`	uint64	Bytes	Required	The maximum number of bytes that the FIFO can contain. A value of 0 means unbounded number of bytes.
`TxnLimit`	uint64	Integer	Optional	This is the limit to the number of outstanding transactions that the profile can generate. Defaults to 1. A value of 0 means unbounded number of outstanding transactions.
`total_txn`	uint64		Optional	This is the total number of transactions this profile will generate before deactivating itself. A value of 0 for `total_txn` indicates unbounded number of transactions.

Table A-7 FIFO configuration (continued)

Parameter	Type	Units	Requirement	Definition
`Rate`	uint64	Integer	Required	Fill or Depletion rate of this profile. Can be expressed: • In units of ATP time • As a rate with unit specifiers compliant to SI or IEC standards See *Supported SI and IEC Standard Rate Units* on page A-44.
`FrameSize`	uint64	Bytes	Optional	Specifies the number of bytes transferred by the FIFO
`FrameTime`	uint64	Time	Optional	Specifies the amount of time the FIFO will be active for

A.2.6 Pattern configuration

Table A-8 shows address pattern configuration parameters.

Table A-8 Pattern configuration

Parameter	Type	Requirement	Definition
`cmd`	PacketCommand	Optional	Packet command type
`address`	Address	Optional	Address is configured as: `base` Address generation is sequential, starting from this value. `increment` The value `address` is increased by this value. `start` If `start` is specified, when `address` exceeds `Xrange`, then generation restarts from `start`, instead of `base`.
`stride`	Stride	Optional	Stride is configured as: `Stride` Value that address increments. `Xrange` Maximum allowed address value.
`random_address`	RandomDesc	Optional	If present, addresses are randomly generated according to RandomDesc configuration. Random addresses can be also configured by specifying • Random address generation type • Base address • Range
`size`	uint64	Optional	If present, all generated packets will have this configured size
`random_size`	RandomDesc	Optional	If present, packet sizes are randomly generated according to RandomDesc configuration

Table A-8 Pattern configuration (continued)

Parameter	Type	Requirement	Definition
`alignment`	uint64	Optional	Address alignment to specified size in bytes when addresses are randomly generated. By default, the alignment is "natural", that is, addresses are aligned to generated packet sizes. `alignment` has no effect for non-random address sequences.
`lowId`	uint64	Optional	Packet ID low ID value: all packets generated by the FIFO will contain an ID with this value lower bound. `lowId` supersedes the global configuration.
`highId`	uint64	Optional	Packet ID high ID value: all packets generated by the FIFO will contain an ID with this value upper bound. `highId` supersedes the global configuration

A.2.7 Random descriptor

Table A-9 shows the parameters used for Random descriptors.

Table A-9 Random descriptor

Parameter	Type	Requirement	Definition
`type`	Type	Required	Random distribution type, supported Types are: • UNIFORM • NORMAL • POISSON • WEIBULL
`uniform_desc`	UniformDesc	Optional	Uniform distribution configuration, takes min possible value and max possible value parameters
`normal_desc`	NormalDesc	Optional	Normal distribution configuration, takes mean and std_dev parameters
`poisson_desc`	PoissonDesc	Optional	Poisson distribution configuration, takes mean parameter
`weibull_desc`	WeibullDesc	Optional	Weibull distribution configuration, takes scale and shape parameters