Antmicro

Busperf

2025-12-30

CONTENTS

CHAPTER

ONE

INTRODUCTION

Busperf is a tool for analyzing bus performance of the most common buses including AHB, APB and AXI, in order to identify throughput bottlenecks.

This documentation provides an overview of Busperf's features. It consists of the following chapters:

- *Analyzers* provides descriptions of available native analyzers
- *YAML bus description* describes how to write a YAML bus description
- *Busperf plugins* describes how to write a custom plugin
- *Output* explains how to interpret Busperf's output

CHAPTER

TWO

ANALYZERS

Busperf provides analyzers for common buses, including AHB, APB and AXI. This chapter provides a list and descriptions of the currently available analyzers.

2.1 Single channel buses

2.1.1 Ready/valid

Handshake name: ReadyValid

Required signals:

- `ready`
- `valid`

2.1.2 AHB

Handshake name: AHB

Required signals:

- `htrans`
- `hready`

2.1.3 APB

Handshake name: APB

Required signals:

- `psel`
- `penable`
- `pready`

2.1.4 Credit/valid

Handshake name: CreditValid

Required signals:

- `credit`
- `valid`

2.2 Multi channel buses

2.2.1 AXI read

Analyzer name: AXIRdAnalyzer

Required signals:

- `ar`
  - `id` (not required for AXI Lite)
  - `ready`
  - `valid`
- `r`
  - `id` (not required for AXI Lite)
  - `ready`
  - `valid`
  - `resp`
  - `last` (not required for AXI Lite)

2.2.2 AXI write

Analyzer name: AXIWrAnalyzer

Required signals:

- `aw`
  - `id` (not required for AXI Lite)
  - `ready`
  - `valid`
- `w`
  - `ready`
  - `valid`
  - `last` (not required for AXI Lite)
- `b`
  - `id` (not required for AXI Lite)
  - `ready`
  - `valid`
  - `resp`

CHAPTER

THREE

YAML BUS DESCRIPTION

This chapter provides examples of YAML bus descriptions for different types of buses.

3.1 Single channel bus

Example `.yaml` for `tests/test_dumps/dump.vcd`:

```
interfaces:
  "a_":
    scope: "some_module"
    clock: "clk_i"
    reset: "rst_ni"
    reset_type: "low"

    handshake: "ReadyValid"
    ready: "a_ready"
    valid: "a_valid"

  "b_":
    scope: "some_module"
    clock: "clk_i"
    reset: "rst_ni"
    reset_type: "low"

    handshake: "Custom"
    custom_handshake: "PythonReadyValid"
    ready: "b_ready"
    valid: "b_valid"
```

- "a_", "b_": names of buses
- reset_type: `low` or `high`
- handshake: possible values: `ReadyValid`, `CreditValid`, `AHB`, `APB`, `Custom`
- custom_handshake: if handshake is set to `Custom`, a name of a Python plugin should be provided

Scopes can also be nested. Example `.yaml` for `tests/test_dumps/nested_scopes.vcd`:

```
scopes:
  base: &base_scope
```

(continues on next page)

(continued from previous page)

```
    - top
    - tb

interfaces:
  "a_":
    scope: [*base_scope, "$rootio"]
    clock: "clk_i"
    reset: "rst_ni"
    reset_type: "low"

    handshake: "ReadyValid"
    ready: "a_ready"
    valid: "a_valid"

  "b_":
    scope: [*base_scope, "some_module"]
    clock: "clk_i"
    reset: "rst_ni"
    reset_type: "low"

    handshake: "ReadyValid"
    ready: "b_ready"
    valid: "b_valid"
```

3.2 Multi channel bus

Example `.yaml` for a multi channel bus:

```
interfaces:
  "ram_rd":
    scope: ["test_taxi_axi_ram", "uut"]
    clock: "clk"
    reset: "rst"
    reset_type: "high"

    custom_analyzer: "AXIRdAnalyzer"
    intervals:
      - [0, 5000000]
      - [1234567890,1324567890]
    ar:
      id:    ["s_axi_rd", "arid"]
      ready: ["s_axi_rd", "arready"]
      valid: ["s_axi_rd", "arvalid"]
    r:
      id:    ["s_axi_rd", "rid"]
      ready: ["s_axi_rd", "rready"]
      valid: ["s_axi_rd", "rvalid"]
      rresp: ["s_axi_rd", "rresp"]
      rlast: ["s_axi_rd", "rlast"]
```

(continues on next page)

(continued from previous page)

```
"ram_wr":
  scope: ["test_taxi_axi_ram", "uut"]
  clock: "clk"
  reset: "rst"
  reset_type: "high"

  custom_analyzer: "AXIWrAnalyzer"
  aw:
    id:    ["s_axi_rd", "awid"]
    ready: ["s_axi_wr", "awready"]
    valid: ["s_axi_wr", "awvalid"]
  w:
    ready: ["s_axi_wr", "wready"]
    valid: ["s_axi_wr", "wvalid"]
    wlast: ["s_axi_wr", "wlast"]
  b:
    ready: ["s_axi_wr", "bready"]
    valid: ["s_axi_wr", "bvalid"]
    bresp: ["s_axi_wr", "bresp"]
    id:    ["s_axi_rd", "bid"]
```

For multi channel buses, you need to specify the analyzer, along with signals required by that analyzer.

- custom_analyzer: possible values: `AXIRdAnalyzer`, `AXIWrAnalyzer`, `\<name of custom python analyzer\>`

CHAPTER

FOUR

BUSPERF PLUGINS

You can extend Busperf's functionality through Python plugins. A custom analyzer can be specified in the YAML file under the `custom_analyzer` and `custom_handshake` keys. To prevent name collisions between native and Python analyzers, the names of the Python analyzers should start with "Python". Some examples are available in `plugins/python`.

4.1 Analyzer requirements

A plugin should define a `create()` function that returns an analyzer object. There are 2 types of Python plugins, `custom_handshake` and `custom_analyzer`, with different uses.

4.1.1 Custom handshake

Custom handshakes are used to calculate statistics for single channel buses. The analyzer object should define the following methods:

- `get_signals(self) -> list[str]`
  - Return value - an array of string names of signals that are required by the analyzer.
- `interpret_cycle(self, signals) -> CycleType`
  - signals - array of signal values (casted to string) during clock cycle
  - Return value - interpreted state of the bus during that clock cycle. See *CycleType*

4.1.2 Custom analyzer

Custom analyzers are used to calculate statistics for multi channel buses. Their analyzer object must define the following methods:

- `get_yaml_signals(self) -> list[tuple[SignalType, list[str]]]`
  - Return value - returns types and paths to each signal defined in yaml that is required by the analyzer. See *SignalType*
- `analyze(self, clk, rst, ...) -> list[Transaction]`
  - Parameters - the method takes as arguments signals requested in `get_yaml_signals`, clk and rst signals are always included and passed as first 2 arguments, all remaining are passed in the same order as in `get_yaml_signals`. Each signal is an array of tuples `(time: int, value: str)`.
  - Return value - the method should return a list of transactions that were present on a bus. See *Transaction*

4.2 Interface types

All types that are used on the rust-python interface are made available in a `busperf` module that is created at runtime and made accessible for the plugins. It defines the following types based on Rust structs:

4.2.1 CycleType

This enum represents the state of a single channel.

```
class CycleType:
    Busy = <CycleType.Busy>                     # performing transaction
    Free = <CycleType.Free>                     # bus is not used
    NoTransaction = <CycleType.NoTransaction>   # transaction is not performed
    Backpressure = <CycleType.Backpressure>     # backpressure
    NoData = <CycleType.NoData>                 # receiver ready but no data is␣
↪available to transfer
    Reset = <CycleType.Reset>                   # reset signal active, bus in␣
↪reset
    Unknown = <CycleType.Unknown>               # invalid/unknown state
```

4.2.2 SignalType

This enum represents what data about signal(s) the analyzer expects.

```
class SignalType:
    Signal = <SignalType.Signal>              # passes every signal change's time␣
↪and value to the analyzer
    RisingSignal = <SignalType.RisingSignal>  # passes times of every rising edge␣
↪of the signal (useful for e.g. clk)
    ReadyValid = <SignalType.ReadyValid>      # passes all time a transaction is␣
↪performed on a ready/valid channel
```

4.2.3 Transaction

This type represents one transaction of the analyzed multichannel bus.

```
class Transaction:
    def __init__(
        self,
        start: int,
        first_data: int,
        last_data: int,
        resp_time: int,
        resp: str,
        next_start: int
    ):
        self.start = start              # time of command issue
        self.first_data = first_data    # time of first data being transferred
        self.last_data = last_data      # time of last data transfer
        self.resp_time = resp_time      # time of response
```

(continues on next page)

(continued from previous page)

```
        self.resp = resp                # value of the response
        self.next_start = next_start    # start time of next transaction
```

CHAPTER

FIVE

OUTPUT

For each described bus, Busperf will calculate and display several statistics.

5.1 Single channel

- `bus_name`: name of the bus
- `busy`: number of clock cycles performing transaction
- `free`: bus is not used
- `no transaction`: transaction is not performed
- `backpressure`: backpressure
- `no data`: receiver ready but no data is available to transfer
- `reset`: clock cycles with reset active
- `transaction delays`: delays in clock cycles between transactions
- `burst lengths`: lengths of bursts including delays during burst

Table matching state of the bus with Busperf statistic name:

busperf	busy	free	no transaction	backpressure	no data	unknown
axi	ready && valid	!ready && !valid	not used	!ready && valid	ready && !valid	no used
ahb	seq / no seq	idle	not used	hready	trans=BUSY	other
credit valid	credit>0 && valid	credit>0 && !valid	credit=0 && !valid	not used	not used	other
apb	setup or access phase	!psel	not used	access && !pready	not used	other

5.2 Multi channel

- `Cmd to completion`: number of clock cycles from issuing a command to receiving a response
- `Cmd to first data`: number of clock cycles from issuing a command to first data being transferred

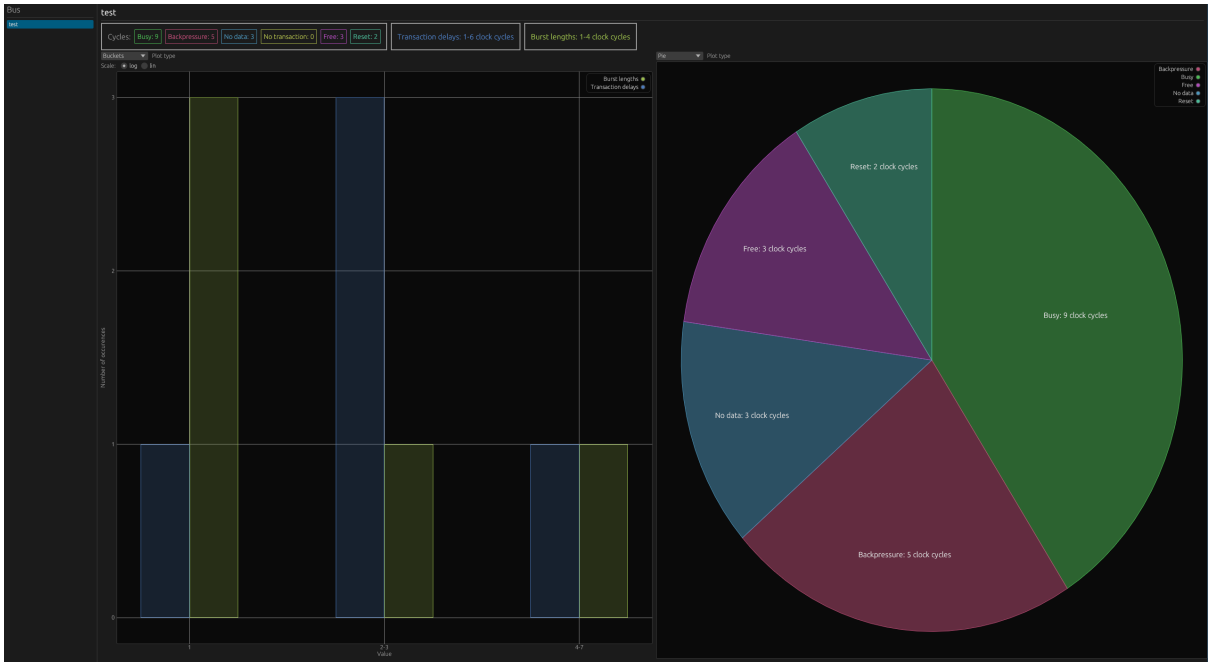
- `Last data to completion`: number of clock cycles from last data being transferred to transaction end
- `Transaction delays`: delays between transactions in clock cycles
- `Error rate`: percentage of transactions that resulted in error
- `Bandwidth`: averaged bandwidth in transactions per clock cycle

5.3 Examples

5.3.1 Single channel buses

bus name	Busy	Back-pressure	No data	No trans-action	Free	Re-set	Transaction delays	Burst lengths
test	9	5	3	0	3	2	1 x1; 2-3 x3; 4-7 x1	1 x3; 2-3 x1; 4-7 x1

bus name	Busy	Back-pressure	No data	No trans-action	Free	Re-set	Transaction delays	Burst lengths
a_	0	0	15	0	0	15	16-31 x1	No transaction on this bus
b_	0	0	15	0	0	15	16-31 x1	No transaction on this bus



5.3.2 Multi channel buses