Open SoC Debug Hardware Documentation

The Open SoC Debug Contributors

Oct 31, 2018

Content

This is the reference implementation of the hardware components of Open SoC Debug.

CHAPTER 1

Testing

The Open SoC Debug hardware reference implementation comes with a set of tests on different levels of abstraction.

Cocotb-based unit tests run in simulation and validate the functionality of smaller parts of the implementation, usually a single debug module or another small components. The test cases are typically stored along the module source code in a `test` subdirectory. Common functionality in all tests is contained in a software library `osdtestlib`.

1.1 Run Cocotb-Based Unit Tests

1.1.1 Prerequisites

The unit tests require Synopsys VCS as simulator. Make sure it's installed.

You *don't* need to install cocotb manually, it's installed when the tests are run.

1.1.2 Run a single test

Tests are executed using the `test/cocotb/cocotb_testrunner` tool. The testrunner reads all `*.manifest.yaml` files describing the test setup, and configures cocotb in a way to run the test with the simulator.

```
# All commands in this example expect you're in the top-level directory
# of the OSD hardware repository.
cd your_osd_hardware_toplevel_dir

# Run the tests for the STM in non-interactive mode
test/cocotb/cocotb_testrunner modules/stm

# Run tests in cocotb debug mode
# This is especially useful when tests are failing due to Python coding
# errors, as it shows backtraces.
test/cocotb/cocotb_testrunner -l DEBUG modules/stm
```

1.1.3 Interactively run a test with GUI and waveforms

During test development and bug hunting, you might want to run the unit tests in the VCS GUI to see waveforms and other runtime data. You can use the `-g` (or `--gui`) switch to show the GUI during the test run.

```
# normal log level
test/cocotb/cocotb_testrunner -g modules/stm

# debug log level
test/cocotb/cocotb_testrunner -l DEBUG -g modules/stm
```

After the GUI shows up,

- select your top-level DUT in the hierarchical view on the left,
- press CTRL-4 to open the waveform view, and then
- press F5 (or go to Simulator, Start/Continue) to start the simulation.

You can see the outputs of the run in the VCS console on the bottom. To re-run the simulation, press F5 twice.

Note: Restarting a simulation picks up all changes made to the **Python test code** automatically. If you change the **HDL code** or the test manifest files you need to close VCS and call the test runner again to see your changes.

1.1.4 Run all tests

In addition to running a single test, `cocotb_testrunner.py` can also run multiple tests at once.

```
test/cocotb/cocotb_testrunner
```

1.2 Cocotb Test Framework

`osdtestlib` is a set of commonly used functionality to test the OSD hardware implementation with cocotb.

1.2.1 osdtestlib.asserts

Assertations for common checking needs

copyright Copyright 2017-2018 by the Open SoC Debug team

license MIT, see LICENSE for details.

`osdtestlib.asserts.`**`assert_signal_value`**(*signal*, *expected_value*)

Raise a TestFailure if the signal doesn't have an expected value

1.2.2 osdtestlib.debug_interconnect

Access the Debug Interconnect in OSD

copyright Copyright 2017-2018 by the Open SoC Debug team

license MIT, see LICENSE for details.

class osdtestlib.debug_interconnect.**AliasBusDriver**(*entity*, *name*, *clock*, *signal_aliases={}*)
Extension of the cocotb.BusDriver to support aliases for signal names

class osdtestlib.debug_interconnect.**NocDiWriter**(*entity*, *clock*, *signal_aliases={}*)
Writer for the OSD Debug Interconnect implemented as NoC

send_packet
Transmit a complete packet to a chosen debug module

Parameters **packet** – debug interconnect packet

class osdtestlib.debug_interconnect.**NocDiReader**(*entity*, *clock*, *signal_aliases={}*)
Reader for the OSD Debug Interconnect implemented as NoC

read_timeout_cycles = 1000

receive_packet
Receive a packet from the debug interconnect

Parameters **set_ready** (*bool*) – Set the ready signal to tell the DUT we can receive data. If set to True, this function will handle the ready signal. If set to False, you must set the ready signal yourself. This mode is useful for toggling the ready signal during the receive operation to achieve greater coverage of edge cases.

Returns DiPacket

class osdtestlib.debug_interconnect.**DiPacket**
A single debug interconnect packet

class TYPE
Packet types

REG = 0

PLAIN = 1

EVENT = 2

RESERVED = 4

class TYPE_SUB
Packet subtypes

REQ_READ_REG_16 = 0

REQ_READ_REG_32 = 1

REQ_READ_REG_64 = 2

REQ_READ_REG_128 = 3

REQ_WRITE_REG_16 = 4

REQ_WRITE_REG_32 = 5

REQ_WRITE_REG_64 = 6

REQ_WRITE_REG_128 = 7

RESP_READ_REG_SUCCESS_16 = 8

RESP_READ_REG_SUCCESS_32 = 9

RESP_READ_REG_SUCCESS_64 = 10

RESP_READ_REG_SUCCESS_128 = 11

RESP_READ_REG_ERROR = 12

NOT_DEFINED = 13

RESP_WRITE_REG_SUCCESS = 14

RESP_WRITE_REG_ERROR = 15

class BASE_REG
Base register addresses

MOD_VENDOR = 0

MOD_TYPE = 1

MOD_VERSION = 2

MOD_CS = 3

MOD_EVENT_DEST = 4

class SCM_REG
SCM Register map

SYSTEM_VENDOR_ID = 512

SYSTEM_DEVICE_ID = 513

NUM_MOD = 514

MAX_PKT_LEN = 515

SYSRST = 516

class MAM_REG
MAM Register map

AW = 512

DW = 513

REGIONS = 514

flits
The data words this packet consists of

Getter Get the flits this packet consists of

Setter Take flits and set the packet contents based on it

set_contents(*dest*, *src*, *type*, *type_sub*, *payload*)
Populate the data fields of a packet

Parameters

- **dest** – DI address of the target module
- **src** – DI address of the sending module
- **type** – packet type
- **type_sub** – packet subtype
- **payload** – payload

equal_to(*dut*, *other_packet*, *mask=None*)
Compares a packet with another packet and outputs if both are equal

Parameters

- **dut** – device under test
- **other_packet** – debug interconnect packet which this packet is compared to
- **mask** – list of boolean variables indicating which flits of the payload are to be ignored

Returns True if the packets are equal (modulo the mask), False otherwise

class osdtestlib.debug_interconnect.**RegAccess**(*dut*, *reader=None*, *writer=None*)
Access registers of debug modules

read_register
Read a value from a specified register and return the read value to the user.

Parameters

- **dest** – DI address of the target module.
- **src** – DI address of the sending module.
- **word_width** – choose between 16, 32, 64 and 128 bit register access.
- **regaddr** – address of the register the value is to be read from.

Returns Value read from the register

write_register
Write a new value into a register specified by the user and read the response to tell the user if the write process was successful

Parameters

- **dest** – id of the target module.
- **src** – id of the sending module.
- **word_width** – choose between 16, 32, 64 and 128 bit register access.
- **regaddr** – address of the register the new value will be written to.
- **value** – value to write to the register

assert_reg_value
Assert that a register contains an expected value

test_base_registers
Test the functionality of the base registers

1.2.3 osdtestlib.exceptions

Exception definitions

copyright Copyright 2017 by the Open SoC Debug team

license MIT, see LICENSE for details.

exception osdtestlib.exceptions.**Error**
Base class for exceptions.

exception osdtestlib.exceptions.**RegAccessFailedException**(*message*)
Exception raised for errors during the register access process.

message
explanation of the error.

exception osdtestlib.exceptions.**DebugEventFailedException**(*message*)
Exception raised for errors during the evaluation of a debug packet.

message
explanation of the error.

exception osdtestlib.exceptions.**ResetFailedException**(*message*)
Exception raised for errors during the reset

message
explanation of the error.

1.2.4 osdtestlib.soc_interface

Trace generators and other tools to stimulate the SoC as DUT

copyright Copyright 2017 by the Open SoC Debug team

license MIT, see LICENSE for details.

class osdtestlib.soc_interface.**StmTraceGenerator**
Mimics CPU signals to generate stimuli for connected debug modules

trigger_event
Set the trace* signals of the STM so that it generates a debug event packet

Parameters

- **dut** – device under test.
- **trace_id** – ID of the trace event.
- **trace_value** – value of the trace event.

Python Module Index

o

Index

S

T

W