CTU CAN FD IP CORE

Testbench Architecture

LOGIC DESIGN SERVICES l.t.d.

February 1, 2026

Document Version	Author	Date	Change description
0.1	Ondrej Ille	04-2021	Initial version
0.2	Ondrej Ille	03-2025	Replace GHDL with NVC
0.3	Ondrej Ille	06-2025	Proof-read, add compilation guide. Update figures and add "Functional coverage agent". Remove "integrated" VIP mode as experimental.

Contents

1. Introduction

This document describes test-bench of CTU CAN FD. It provides guide to integrate main CTU CAN FD test-bench into other (e.g. SoC level) test-bench, and it explains types of tests available. CTU CAN FD contains following tests / test-benches:

1. Main test-bench with following types of tests:
   - Compliance tests - Verify compliance of CTU CAN FD to ISO11868-1 2015. Contains all tests from ISO 16845-1 2016. To run these tests, you need Compliance test library compiled and linked to simulation via PLI. This library is a submodule of CTU CAN FD repository.
   - Feature tests - Verify features of CTU CAN FD thah are not directly related to compliance with ISO11898-1 2015 (e.g. TX/RX buffers, Interrupts, special modes, frame filtering, etc.).
   - Reference tests - Each test applies stimulus recorded from reference implementation of CAN protocol, checks that CTU CAN FD can receive such sequence, and accepts frame correctly (black-box testing of cooperability).
2. RX buffer unit test - RX buffer has its own block-level test-bench. It is used to verify corner-cases of RX Buffer FIFO.

This document focuses on main CTU CAN FD test-bench, and further reffers to it only as test-bench. It has following features:

- Testbench is written in VHDL, compliant with VHDL 2008.
- Reference model of CAN bus communication that is used in compliance tests, is written in C++ 17. Reference model is part of Compliance test library. Compliance library is comiled as shared object library (.so), and linked to the simulation. Test-bench communicates with Compliance library via VPI interface (GHDL specific) or VHPI interface (IEEE 1076 standard). There is a separate library for each supported simulator. For compiling Compliance test library, reffer to documentation in commercial delivery of Compliance library. Compilation is required for configuring the path of CTU CAN FD VIP inside TB.
- All test functionality is abstracted to CTU CAN FD VIP.

1.1 Test environment

CTU CAN FD development uses following dependecies/tools:

- NVC - VHDL simulator
- GTKWave - waveform viewer.

- Vunit - Unit test framework for VHDL.

Reffer to CTU CAN FD repository for ready-made docker image with all the dependencies installed.

1.2 Supported simulators

CTU CAN FD test-bench currently supports following simulators:

- NVC - Use at least version 1.16.2
- VCS - Use at least version V2023-12

2. Testbench architecture

Test-bench consists of following parts:

- CTU CAN FD VIP - contains all test code, test sequences, libraries, packages and agents. CTU CAN FD VIP communicates with Compliance test library via a PLI interface.
- CTU CAN FD (DUT) - contains RTL.

Further in this document, CTU CAN FD VIP is reffered to only as VIP. CTU CAN FD design is reffered to as DUT. Block diagram of CTU CAN FD test-bench is shown in Figure 2.1.

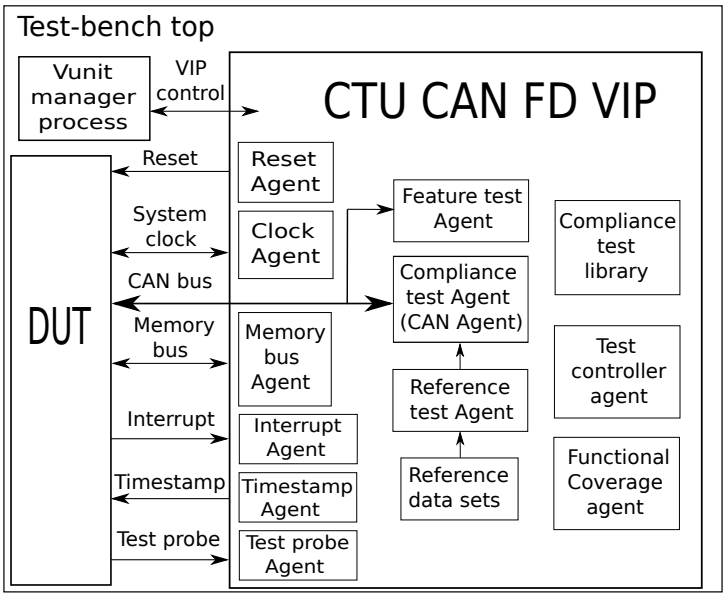

Figure 2.1: Test-bench block diagram

2.1 Compiling the testbench

To compile the design and testbench execute following steps:

1. Compile all ".vhd" files from "rtl/slf_rtl.yml" source list file into a "ctu_can_fd_rtl" library.
2. Build the compliance test library. See [3] for details.

3. Choose which "tb_top" wrapper you will be running. There are following wrappers available:
   (a) Vunit - Choose if you will be running with VUnit
   (b) Simple - If you will be running without Vunit
4. Compile all ".vhd" files from "test/slf_tb_dependencies_<tb_type>.yml" source list file to "ctu_can_fd_tb" library. <tb_type> is either "vunit" or "simple" based on previous step.
5. Compile all files from "test/slf_tb_common.yml" to "ctu_can_fd_tb" library.
6. Compile all files from "test/slf_tb_top_<tb_type>.yml" to "ctu_can_fd_tb" library.

2.2 VIP Interface

CTU CAN FD VIP is connected to DUT via interfaces shown in Table 2.1.

Interface	Signals	Connected to	Description
Reset	***res_n***	Reset agent	Control of asynchronous reset of DUT.
System clock	***clk_sys***	Clock agent	Control of DUTs clock.
DFT support	scan_enable	Test port agent	Control of DUTs scan mode.
CAN bus	***can_tx***	Compliance , Reference, Feature test agents	Connection to CAN bus (driving CAN RX and monitoring CAN TX of DUT).
	can_rx		
Memory bus	***scs***	Memory bus agent	Chip select
	swr		Write enable
	srd		Read enable
	sbe		Byte enables
	write_data		Write data to DUT.
	read_data		Read data from DUT.
	address		Memory/Register address.
Interrupt	***int***	Interrupt agent	Monitoring of DUTs interrupt output.
Test probe	***test_probe***	Feature test agent	Monitoring of DUT "test port" for various test features. Required only for feature tests.
Timestamp	***timestamp***	Timestamp agent	Control of DUTs timestamp input.
VIP control	***test_start***	Test Controller agent	Request to start test.
	test_done		Indication test has finished.
	test_success		Test result (1 - passed, 0 - failed).

Table 2.1: CTU CAN FD VIP interface signals

The behavior of VIP is following:

- VIP drives ***res_n*** of DUT.
- VIP generates clock signal for DUT (period is given by ***cfg_sys_clk_period*** generic of VIP).
- VIP monitors ***can_tx*** pin of DUT and drives ***can_rx*** pin of DUT (generates and monitors CAN frames).
- VIP generates memory transactions on its Memory bus to access registers of DUT.
- VIP monitors ***int*** pin of DUT.

- VIP monitors ***test_probe*** pin of DUT.
- VIP drives ***scan_enable*** pin of DUT.
- VIP is integrated in TB as on Figure 2.1 and simulation is controlled by Vunit manager (See tb_top_ctu_can_fd.vhd)

2.3 Test execution flow

Control of VIP by test-bench is following:

1. Testbench sets ***test_start*** = '1'.
2. Testbench waits until ***test_done*** = '1'.
3. Testbench checks ***test_success***. If ***test_success*** = '1', the test passed, otherwise test failed.

All tests follow basic test sequence: ***test_start*** = '1' is interpreted by Test controller agent. Test controller agent invokes different agents based on type of test:

1. Compliance tests - Control over TB is handed over PLI to Compliance test library (shared object library linked to simulation). Compliance test library forks its own test thread, and executes test sequence in this thread. Thread communicates with rest of the test-bench via PLI (see 2.8.1), and controls Clock agent, Memory bus agent and Compliance test agent (CAN agent). When test sequence ends, it signals this back to Test controller agent that passes the result of test back to ***test_done*** and ***test_success***.
2. Feature tests - Test controller agent requests Feature test agent to start running the test. Feature test agent uses all the other agents connected to DUT, and executes test sequence. After the test sequence, feature test agent gives control back to Test controller agent that passes the result back to ***test_done*** and ***test_success***.
3. Reference tests - Test controller agent requests running the test from Reference test agent. Reference test agent applies reference test sequences to DUT via Compliance test agents driver. When Reference test agent sequence ends, it gives control back to Test controller agent that passes the result back to ***test_done*** and ***test_success***.

2.4 Communication mechanisms

Agents in VIP communicate together via communication channel implemented in "tb_communication_pkg.vhd". Communication channel provides message passing mechanism ("send" function). Each agent implements single "receiver" of messages ("receive_start" and "receive_finish" functions). Messages can be sent by any process at any time, however only one message can be sent at a time (it is not possible to send multiple messages at the same time), over single channel. Destination agent is selected with each message being sent. Communication is synchronous ("send" function returns after the message has been received by destination agent). CTU CAN FD VIP uses single channel ("default_channel" signal) for communication.

2.5 Report mechanisms

Test-bench contains package (tb_report_pkg.vhd) that is used to report, and execute checks in the implemented tests. Any call to "error_m", "check(false,...)" or "check_false(true,...)" will make any test fail (***test_success*** will stay 0 when ***test_done*** goes high at the end of test).

VIP contains own log verbosity mechanism. There are 4 verbosity levels:

verbosity_debug All logs are shown, including "debug_m" calls.

verbosity_info Only "info_m", "warning_m", and "error_m" calls are logged. Calls to "check(true,...)"/"check_false(false,...)" are also logged.

verbosity_warning Only "warning_m" and "error_m" calls are logged.

verbosity_error Only "error_m" calls are logged.

With any verbosity level, calls to "check(false,...)"/"check_false(true,...)" are always logged, since this means a test fail condition occured. Verbosity level used by VIP can be configured by a call to "set_log_verbosity" function.

2.6 Random number generation

VIP contains pseudo-random number generator in "tb_random_pkg.vhd". VIP initializes random number generator in any test based on ***seed*** generic of VIP. The Vunit framework used to run CTU CAN FD development generates random value for ***seed*** generic. Randomization is applied in majority of feature tests, and compliance tests. CAN frame fields which have predefined value in ISO16845-1 2016 for each test, are not randomized (to meet conditions of ISO 16845-1 2016).

2.7 Agents

2.7.1 Clock agent

Clock agent generates ***clk_sys*** clock. Period, jitter and duty cycle of generated clock can be configured. Clock agent provides option to wait for one clock cycle. Clock agent is used by all test types. The clocks generated by clock agent are used to clock the DUT.

2.7.2 Reset agent

Reset agent generates DUTs reset (***res_n***). DUT is reset in beginning of each test. Polarity of a reset can be configured.

2.7.3 Memory bus agent

Memory bus agent generates memory transactions compatible with DUTs RAM-like interface (see [1]). An example of transfers on this interface is shown in Figure 2.2. This interface is compatible with Avalon interface. 8, 16 and 32 bit accesses are supported. Read and Write accesses are supported. Read accesses are always blocking (see access functions in "mem_bus_agent_pkg.vhd"). Write accesses can be blocking or non-blocking. Memory bus agent supports burst accesses. Memory bus agent contains FIFO where accesses can be posted, and then executed in bulk.

2.7.4 CAN agent

CAN agent is used by two test types: compliance tests and reference tests. CAN agent drives DUTs ***can_rx*** and monitors/checks whether DUTs ***can_tx*** signals are as expected. Sequences which are driven/monitored by CAN agent, are produced by either of:

- Compliance test library

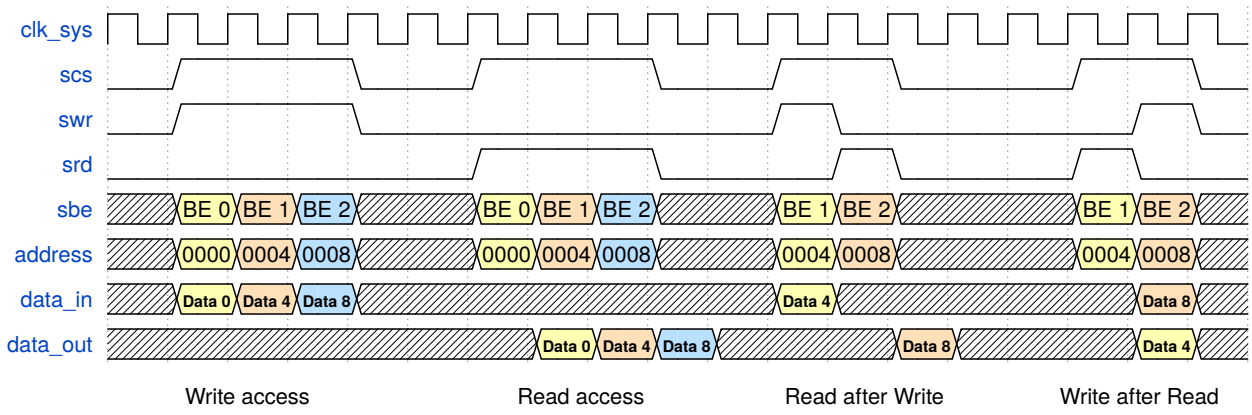

Figure 2.2: Memory bus agent transactions

- Reference test agent (the data are defined by reference data sets).

CAN agent consists of two parts:

Driver Drives sequences to ***can_rx*** of DUT.

Monitor Monitors sequences on ***can_tx*** of DUT.

Driver and monitor each contain FIFO which hold items to be driven and monitored. If there are multiple items in FIFO, they are driven/monitored one after another, therefore creating sequence of bits (similar to UVM sequence, and sequence item). Such sequence represents CAN frames. Each driven item consists of:

value Logic value which is put on ***can_rx*** when this item is being driven.

time Duration for which this item is driven.

Each monitored item consist of:

value Logic value which is checked on ***can_tx*** during monitoring of this item.

time Duration for which this item is monitored. Value should be multiple of sample_rate.

sample_rate Sampling rate used to monitor this item. Monitored item is not checked permanently, but in discrete moments separated by sampling rate. If ***can_tx*** does not match value of currently monitored item in the moment of sampling, mismatch counter is incremented and test fails.

ISO 11898-1 2015 model in compliance test library translates CAN frames to sequences of driver and monitor items. To send CAN frame to DUT, compliance test library translates bits of the frame into sequence of driver items, and drives them via CAN agents driver. Similarly, to check transmitted frame, compliance test library translates the expected CAN frame to sequence of items monitored by CAN agents monitor. Typically, compliance test library translates single bit on CAN bus to single driven/monitored item. Sampling rate of monitored items is equal to single time quanta (since ISO 16845 defines that time quanta should be used as granularity of checking ***can_tx*** value).

Driver and monitor typically operate simultaneously. An example scenario is:

- Transmit frame to DUT, and check that DUT issues dominant acknowledge at correct time.

If both driver and monitor contain the same CAN frame (monitored frame was converted to all Recessive bits with ACK bit dominant), then the example above is achieved. Alternatively, monitor can be delayed from driver by configurable time. This feature allows compensating input delay of DUT.

Typical operation of CAN agent is following:

1. Flush driver and monitor FIFOs (to be sure there are no remaining items).
2. Insert sequences to driver and monitor FIFOs.
3. Configure monitor delay.
4. Start driver and monitor.
5. Wait until driver and monitor are finished (during this time, communication channel is blocked).
6. Issue "check result" command to monitor. This will print error into simulator log, if any mismatches occured in monitored sequence (causing test to fail).

An example of CAN agent operation in which Driver transmits a frame to DUT and monitor checks that DUT issues ACK in correct moment is shown in Figure 2.3.

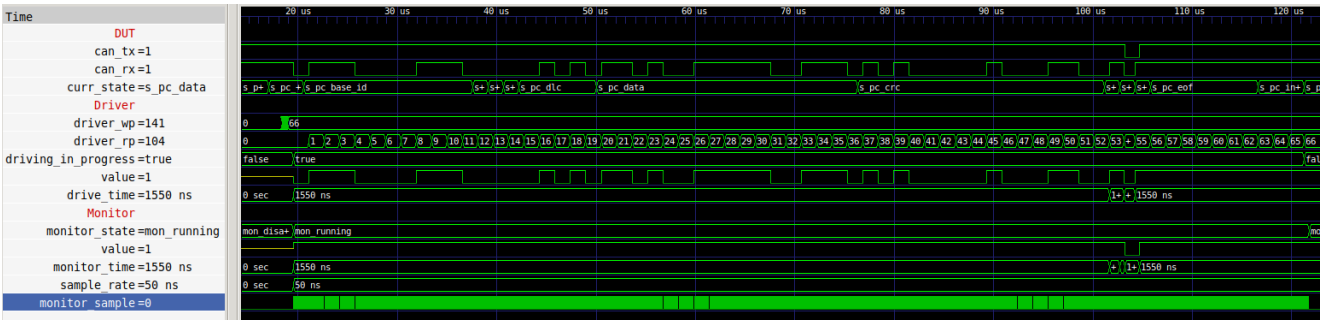

Figure 2.3: CAN agent example

2.7.5 Timestamp agent

Timestamp agent drivers ***timestamp*** signal of VIP. Timestamp agent generates up-counting sequence of values, synchronous to ***clk_sys***. Counting step, and number of cycles needed to advance to next value (prescaler) can be configured. Timestamp agent is used by feature tests which verify timestamping of RX frames or time triggered transmission.

2.7.6 Interrupt agent

Interrupt agent monitors ***int*** input of VIP. It is used to check whether DUTs interrupt is asserted or de-asserted. Polarity of interrupt is configurable.

2.7.7 Test probe agent

Test probe agent watches DUTs ***test_probe*** output. This agent uses ***test_probe*** to observe CTU CAN FDs signals indicating sample point and start of bit. Test-probe agent provides functions for synchronizing with DUTs start of bit or sample point. Test probe agent is used by feature tests. Test probe agent also drives ***scan_enable*** input of DUT.

2.7.8 Feature test agent

Feature test agent is active only in feature tests. When testbench invokes features test agent, the agent calls test specific sequence ("*_ftest.vhd" files contain test sequences), based on name of the test (**test_name** generic). Feature test agent has following capabilities:

- Contains another instance of CTU CAN FD. This instance is reffered to as Test node, and DUT communicates with this node as part of feature tests.
- Signal delayers allowing to configure arbitrary ***can_tx*** -> ***can_rx*** delay for each node (DUT and Test Node).
- Ability to force bus level (value received by both nodes on CAN bus).
- Ability to force ***can_rx*** of single node (either DUT or Test Node).
- Ability to check value of ***can_tx*** of each node.

These capabilities are used by feature test sequence to verify functionality of DUT. Feature tests use higher level API (higher than direct register access), to access functionality of DUT (see "feature_test_agent_pkg.vhd").

2.7.9 Reference test agent

Reference test agent is used by reference tests. It executes test sequence from dedicated reference data set (reference_data_set_*_pkg.vhd). Each reference data set contains 1000 frames which were transmitted by a reference CAN implementation and recorded.

2.7.10 Functional coverage agent

Functional coverage agent contains Functional Cover points in PSL language. Functional coverage agent measures functional coverage of the test-bench. Functional coverage agent peeks into the design via VHDLs external names.

2.8 Test types

2.8.1 Compliance tests

Functional diagram of the test-bench during compliance tests is shown in Figure 2.4. Compliance tests execute all tests from ISO 16845-1 2016. Therefere, compliance tests provide claims of CTU CAN FD compliance towards ISO 11898-1 2015. CAN bus bit rate used by these tests is configured via VIPs generics. Several compliance tests have limitations with regards to allowed bit rate. To see these limitations, reffer to test list files in "test/tlf_compliance_*.yml"). These limitations are given by architecture of CTU CAN FD compliance testing solution. It is intetion to remove these limitations in the next development of CTU CAN FD. Several tests override the default bit rate to meet conditions of the test given by ISO11898-1 2015 (e.g. test 7.6.23 calculates new bit rate from configured one, since test requires it to use certain bit rate ratios).

When a compliance test is started, testbench gives control to compliance test library via PLI interface. Compliance test library forks a thread where the test runs. Therefore, there are two contexts in compliance tests:

- Simulator context - Simulation is executed in this context, events are scheduled, and PLI callbacks are executed.
- Test context - Test sequence from compliance test library is executed in this context.

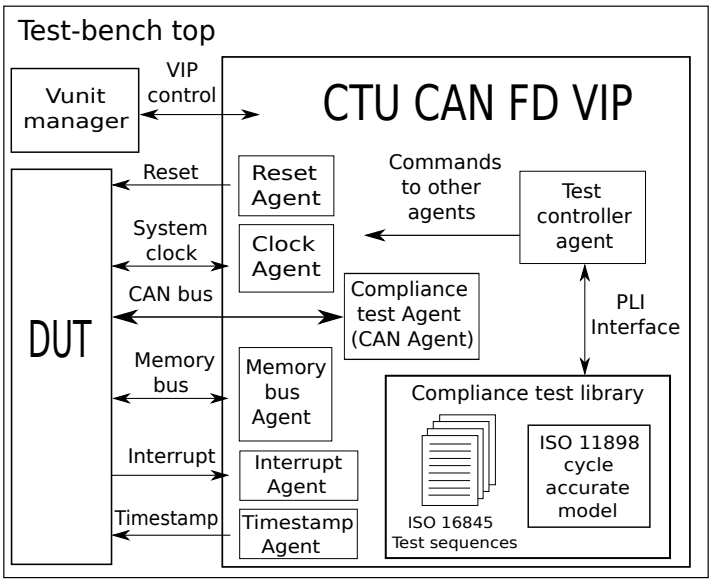

Figure 2.4: Compliance test

Test sequence running in test context communicates with the simulation via shared memory interface. The shared memory interface guarantees that PLI handles will only be accessed from simulator context. Thus, the PLI handles will not be corrupted.

Compliance test library contains model of ISO11898-1 2015. The model is a golden reference used to generate test sequences that are executed by CAN agent inside digital simulator. Reference model has following features:

- Full support of ISO 11898-1 2015 (all three variants: CAN FD enabled, CAN FD tolerant, Classical CAN)
- Cycle accurate representation of CAN frame.
- Can lenght / shorten bits to verify DUTs synchronization.
- Error insertion (all error types and positions can be modelled) and glitch insertion.

For more detailed architecture of compliance test library, reffer to [3].

PLI Interface

As PLI interface, VIP supports VHPI interface (IEEE 1076 standardized). There are two variants of VHPI: One for NVC, and one for VCS. PLI interface consists of signals that are used to communicate between testbench and compliance library. Table 2.2 lists these signals. Compliance test library, acts as master on this interface. It pushes transactions to a shared memory location (inside Compliance test library), and simulator side of this interface "picks-up" these request with VPI/VHPI callback on ***pli_clk***. Simulator then drives them to PLI signals in VIP. Test controller agent then interprets these signals, and sends commands to target agent via standard communication channel. This approach guarantees that internal structures of digital simulator are modified only from simulator context. PLI interface provides means for accessing functionality of agents within TB. Compliance library can therefore control clock/reset generation, transactions to DUT, CAN agent, etc.

Signal	Description
pli_control_req	TB is requesting run of compliance test from compliance library. Set by VIP in early in compliance test run.
pli_control_ack	Compliance test library acknowledge for ***pli_control_req***.
pli_req	Transaction request from compliance library
pli_ack	Transaction acknowledge to compliance library.
pli_cmd	Type of command/transaction being sent.
pli_dest	Transaction destination agent.
pli_data_in	Transaction data input.
pli_data_in_2	Transaction data input 2.
pli_str_buf_in	Transaction string buffer input.
pli_data_out	Transaction data output.
pli_clk	PLI clock.

Table 2.2: PLI interface signals

2.8.2 Reference tests

Reference tests use CAN agent to apply a bit-sequent to DUTs **can_rx**. This sequence was recorded from reference CAN controller implementation upon transmission of random frame. After this sequence is applied, test reads received CAN frame from DUT, and checks it matches CAN frame which was supposed to be received. This approach provides "black-box" like testing functionality. Reference tests contain 10 data sets, each with 1000 pre-recorded CAN frames. Data set is chosen by "test_name" generic of VIP. Each frame from data-set is applied by following sequence:

1. Store bit sequence from data set to CAN agents driver.
2. Start CAN agent driver.
3. Wait till driver finishes.
4. Read CAN frame received by DUT and compare it with reference frame from data-set. This frame corresponds to bit sequence from point 1.

2.8.3 Feature tests

Feature tests verify various "features" of CTU CAN FD as: Interrupts, register map, special modes, TX/RX buffers, etc. These features are usually not directly related to ISO11898-1 2015, and they are specific to CTU CAN FD. Functional diagram of TB during feature tests is shown in 2.5.

In feature tests, DUT communicates on CAN bus with another instance of CTU CAN FD located inside Feature test agent (Test node). This setup allows invoking various situations in DUT. An example of such test sequence is following:

- Test reads size of DUTs RX buffer.
- Test invokes transmission of CAN frames by Test Node. Amount of frames transmitted is selected to achieve overflow of RX buffer in DUT.
- During transmission of frames, test monitors that RX buffer overflow occurs upon reception of frame which should fill RX buffer memory (not before), therefore veryfing that overflow occurs properly.

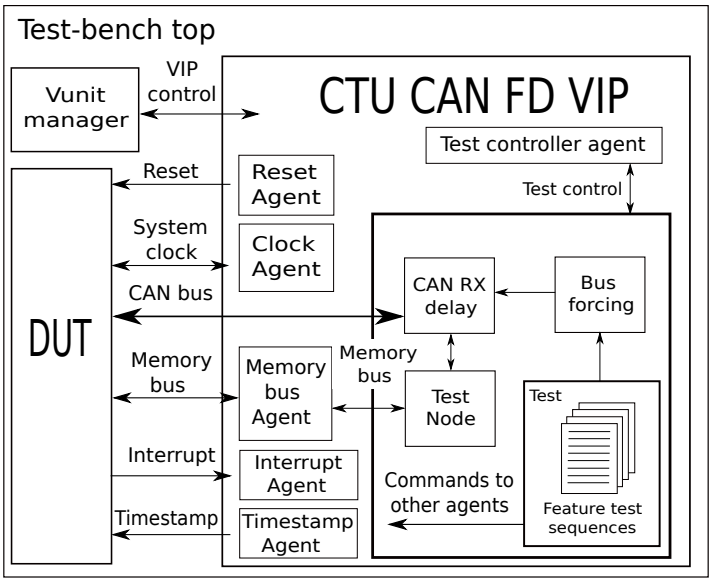

Figure 2.5: Feature test

Bibliography

[1] CTU CAN FD - System architecture

[2] CTU CAN FD - Datasheet

[3] ISO 16845 Compliance test library - `https://github.com/Blebowski/iso-16845-compliance-tests`