CTU CAN FD IP CORE

Testbench Architecture

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Measurement

September 8, 2024

Document Version	Author	Date	Change description
0.1	Ondrej Ille	04-2021	Initial version

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-benches, and it explains types of tests which are present within this test-bench. CTU CAN FD contains following tests / test-benches:

1. Main test-bench with following types of tests:
   - Compliance tests - Verifies compliance of CTU CAN FD to ISO11868-1 2015. All tests from ISO 16845-1 2016 are implemented. To run these tests, you need Compliance test library compiled and linked to simulation via PLI. This library is part of open-source CTU CAN FD repository only in form of compiled binary (works only with GHDL simulator in CTU CAN FD Gitlab environment). To compile this library and link it to TB in other simulator, access to its source code is needed. This library can be provided based on commercial agreement.
   - Feature tests - Verify features / corner-cases of CTU CAN FD which are not directly related to compliance with ISO11898-1 2015 (e.g. TX/RX buffers, Interrupts, special modes, frame filtering, etc.). These tests are open-source.
   - Reference tests - Each test applies stimulus recorded from reference implementation of CAN protocol, and checks that CTU CAN FD can receive such sequence and accepts frame correctly (black-box testing of cooperability). These tests are open-source.
2. Unit tests - Each test has its own test-bench. These tests are executed in development of CTU CAN FD to achieve higher functional coverage of CTU CAN FD verification. These test-benches are not intended for integration into other test-benches, and their description is beyond the scope of this document.

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

- Written in VHDL, compliant with VHDL 2008. Reference model of CAN bus communication which is used in compliance tests, is written in C++ 17, and it is part of Compliance test library linked to simulation as shared object library (.so). Test-bench communicates with Compliance library via VPI interface (GHDL specific) or VHPI interface (IEEE 1076 standard). Communication interface is chosen by using relevant .so file. 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.
- Can be run stand-alone (as in CTU CAN FD development), or integrated as part of other, e.g. UVM or OSSVM system level test-bench.
- All test functionality is grouped in CTU CAN FD VIP, making it easy to integrate this block into other test-bench.

- Configurable bit rate on CAN bus which is applied when running compliance tests (ISO 16845-1 requires testing various CAN bus bit rates, therefore for full compliance, it is recommended to run the tests with various bit rates as described in ISO 16845-1) and feature tests. Reference tests run with 2Mbit/500Kbit bit rate (common bit rate in automotive CAN).

1.1 Test environment

CTU CAN FD development uses following dependecies/tools:

- GHDL - VHDL simulator
- GTKWave - waveform viewer.
- Vunit - Unit test framework for VHDL.

These dependencies are required only if test-bench is about to be executed in native CTU CAN FD development environment (developing CTU CAN FD in its original repository). In such cases, reffer to CTU CAN FD repository for ready-made docker image with all dependencies installed.

If CTU CAN FD is being integrated into other system level test-bench with commercial simulator, none of the tools are required. In such case, following resources are sufficient:

- Simulator with VHDL 2008 support
- Testbench source files (see "tb" folder in CTU CAN FD delivery package) and list file "tb_src.lst". Reffer to "Delivery package" in CTU CAN FD Gitlab page. These shall be compiled in "ctu_can_fd_tb" library.
- CTU CAN FD design source files (see "rtl" folder in CTU CAN FD delivery package) and list file "rtl_lst.txt". These shall be compiled in "ctu_can_fd_rtl" library.
- Compliance tests library compiled as shared object file (.so) and linked to simulation.

If compiled Compliance library is not present, then it is still possible to run the TB, however, compliance test types are not available in such case.

2. Testbench architecture

Test-bench consists of two parts:

- CTU CAN FD VIP - contains all test code, test sequences, libraries, packages and agents. Compliance test library is linked to simulation as part of CAN FD VIP.
- 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. There are two options how to use VIP:

- Stand-alone - Verification of DUT as stand-alone IP. TB top wrapper (tb_top_ctu_can_fd.vhd) binds DUT to VIP, and controls simulation via Vunit directives (Vunit is required in stand-alone mode). VIP is used in this mode in development of CTU CAN FD.
- Integrated - DUT is integrated as part of larger design, and VIP is instantiated as part of system level test-bench. Reffer to Chapter 3 for instructions on how to integrate VIP into SoC level test-bench.

Block diagram of CTU CAN FD test-bench (with VIP used stand-alone) is shown in Figure 2.1.

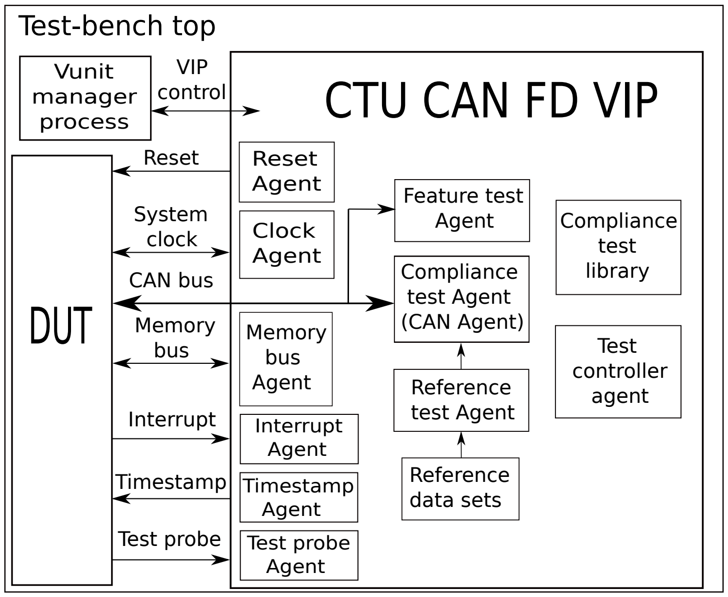

Figure 2.1: Test-bench block diagram

2.1 VIP Interface

CTU CAN FD VIP is connected to DUT (CTU CAN FD) via interfaces as 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

2.2 VIP Modes of operation

VIP can operate in two modes: Stand-alone and Integrated.

2.2.1 Stand-alone mode

When operating in Stand-alone mode, ***stand_alone_vip_mode*** generic of VIP shall be set to “true”. In Stand-alone mode, 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)

2.2.2 Integrated mode

When operating in Integrated mode, ***stand_alone_vip_mode*** generic of VIP shall be set to "false". In Integrated mode, the behavior of VIP is following:

- VIP does not directly drive reset of DUT. DUT is likely reset by System-wide reset on SoC level. Test-bench which integrates VIP shall watch VIP*s* ***res_n*** output, and assert reset of DUT when reset is asserted on ***res_n*** output of VIP (e.g. via combination of mirror and force/release statements available in System verilog and VHDL 2008). Note that ***res_n*** is active low.
- VIP does not generate clock signal for DUT. DUT is likely clocked by some kind of clock controller, or oscillator model which is part of SoC design. Since ***clk_sys*** signal of VIP si bi-directional, VIP watches the clock on this interface and synchronizes its operation to actual clock of DUT (***cfg_sys_clk_period*** generic of VIP shall be set to actual period of this clock).
- VIP does not directly drive ***can_rx*** of DUT. It is likely that ***can_rx*** pin of DUT is connected to some form of GPIO multiplexor or pad model (within a complex SoC simulation). Test-bench which integrates VIP, shall monitor the ***can_rx*** output of VIP and directly connect it (without any simulation delay) to pad which corresponds to CAN RX signal within a simulated system.
- VIP monitors ***can_tx*** output of DUT. Test-bench which integrates the VIP, shall drive **can_tx** by mirrored value of **can*_tx*** DUT output (e.g. by mirror or hierarchical reference).
- VIP monitors ***int*** pin of DUT. Test-bench which integrates the VIP, shall drive ***int*** input of VIP by mirrored values of DUTs ***int*** output.
- VIP monitors ***test_probe*** output of DUT. Test-bench which integrates the VIP, shall drive ***test_probe*** input of VIP by mirrored value of DUTs ***test_probe*** output.
- VIP does not directly drive ***scan_enable*** pin of DUT. ***scan_enable*** pin is likely driven by SoC level DFT controller. Test-bench which integrates VIP shall watch ***VIPs scan_enable*** output, and assert ***scan_enable*** input of DUT if this signal goes high.
- VIP is integrated in custom test-bench. It is likely part of System-Verilog testbench top. VIP control interface shall be driven by TB which integrates the VIP. Reffer to Chapter 3 for integration manual of VIP.

2.3 Test execution flow

Control of VIP by test-bench which integrates it, shall be following:

1. Set ***test_start*** = '1'.
2. Wait until ***test_done*** = '1'.
3. Check ***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 agent based on type of test:

1. Compliance tests - Control over TB is relinquished 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 which passes the result of test back to ***test_done*** and ***test_success***.
2. Feature tests - Test controller agent requests from 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 which 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 reads Reference test files and applies them to DUT via Compliance test agents driver. When Reference test agent sequence ends, it gives control back to Test controller agent which passes the result back to ***test_done*** and ***test_success***.

2.4 TB 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 message has been received by destination agent). CTU CAN FD VIP uses single channel (“default_channel” signal) for communication.

2.5 TB report mechanisms

TB contains package (tb_report_pkg.vhd) which is used for reporting and checking in 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 to signal 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 condition causing test to fail 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. It is therefore recommended to drive ***seed*** generic to a seed used within TB that integrates the VIP. If seed is not set, then VIP is not randomized and tests will have the same coarse of actions each time they are executed.

Following items are randomized within VIP:

- CAN frame contents, where applicable.
- Bit rate on CAN bus in feature tests which have "btr_" prefix in name. In other tests, bit rate is given by VIP generics (see Table 3.2).
- Moments at which TX commands are issued to TXT buffers.
- Moments at which frames are polled from RX buffer.
- Transmitter delay in trv_delay and ssp_cfg feature tests.

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. When VIP operates with ***stadalone_vip_mode***=true, then clocks generated by clock agent are used to clock DUT. If VIP operates **with stadalone_vip_mode**=false, then clocks of CAN agent are ignored, and rest of VIP sychronizes to ***clk_sys*** pin of VIP (***clk_sys*** pin is used as input).

2.7.2 Reset agent

Reset agent generates DUTs reset (***res_n***). DUT is reset in beginning of each test. Polarity of 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 into which accesses can be posted, and then executed in bulk.

2.7.4 Compliance test agent

Compliance test agent (in compliance test library reffered to as "CAN agent"), is used by two test types: compliance tests and reference tests. Compliance test agent is used to drive sequences to DUT and monitor/check whether DUTs responses are as expected. Sequences which are driven/monitored by compliance agent, are produced by compliance test library (or they are defined by reference data sets, in case of reference tests). Compliance test agent is connected to DUTs ***can_tx*** and ***can_rx*** signals. Compliance test agent consists of two parts:

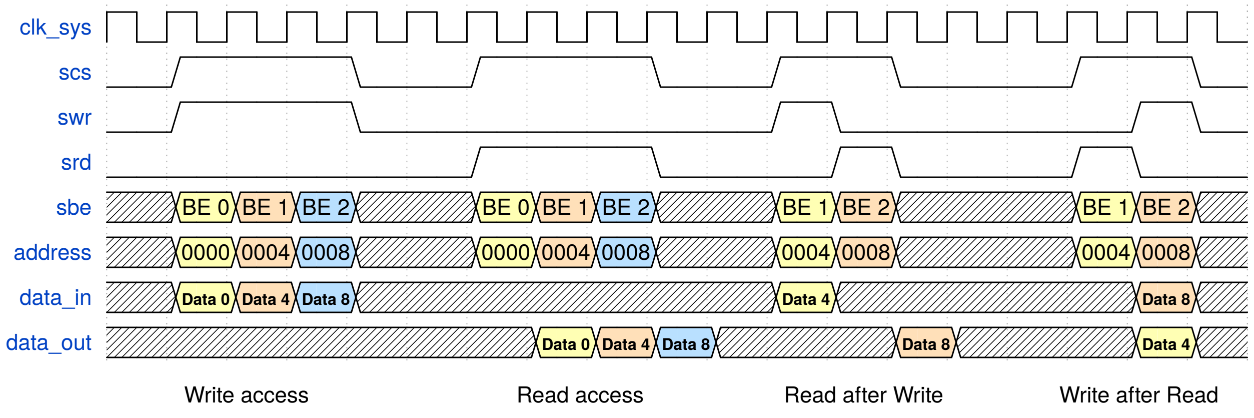

Figure 2.2: Memory bus agent transactions

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 for monitoring of 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 makes the test fail when it ends).

ISO 11898-1 2015 model in compliance test library translates CAN frames to sequences of driver and monitor items. Sending frame to DUT, is implemented by translating bits of the frame into sequence of driver items, and driving it via CAN agents driver. Similarly, checking of transmitted frame is implemented as monitoring sequence of items 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 chosen to be 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 can operate simultaneously. This is used in following scenario: Transmit frame to DUT, and check that DUT will issue dominant acknowledge at correct time. In such case, both driver and monitor are started at the same time. If they both contain the same CAN frame (monitored frame must be converted to all Recessive bits with ACK bit dominant), then such behavior is achieved. Alternatively, monitor can be delayed from driver by configurable time. This feature allows compensating input delay of DUT.

Typical use-case 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.

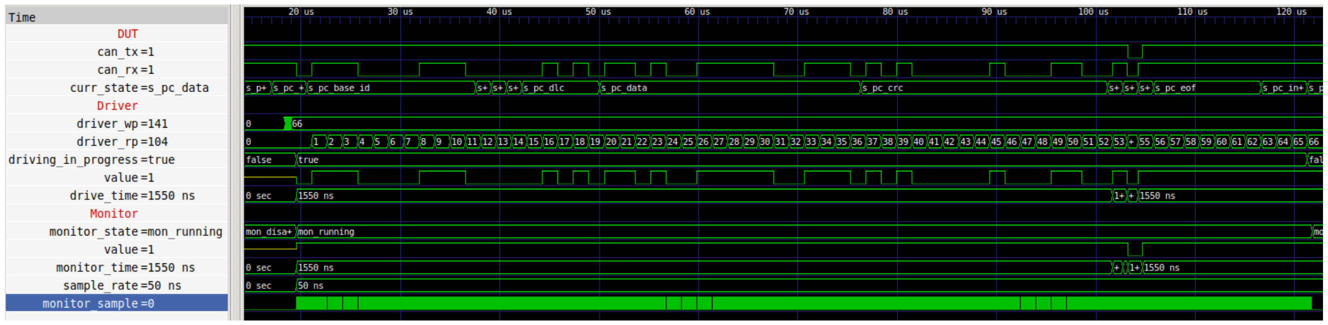
```
Time
DUT
can_tx =1
can_rx =1
curr_state =s_pc_data
Driver
driver_wp =141
driver_rp =104
driving_in_progress =true
value =1
drive_time =1550 ns
Monitor
monitor_state =mon_running
value =1
monitor_time =1550 ns
sample_rate =50 ns
monitor_sample =0
```

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, as well as 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 can be configured.

2.7.7 Test probe agent

Test probe agent spies on DUTs ***test_probe*** output. Test probe is used 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. Upon invoking by test controller agent, it 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.

Capabilities are used by feature test sequence to verify certain 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 (and recorded) from reference CAN implementation.

2.8 Test types

2.8.1 Compliance tests

Functional diagram of compliance tests is shown in Figure 2.4. Compliance tests execute all tests from ISO 16845-1 2016, therefere providing complete compliance testing of CTU CAN FD towards ISO 11898-1 2015. CAN bus bit rate used by these tests is configured via VIPs generics (therefore must be chosen at compile time). Several tests have limitations with regards to allowed bit rate (reffer to 3.7 for these limitations). Also, 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, it gives control over TB to compliance test library via PLI interface. Compliance test library forks a thread in which it exectues the test. Therefore, there are two contexts in compliance tests:

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

Test sequence running in test context communicates with simulation via shared memory interface, which guarantees that internal handles of simulator will only be accessed from simulator context, therefore not corrupting any internal memory structures of simulation (signal handles, etc.).

Compliance test library contains model of ISO11898-1 2015, which serves as golden reference for generation of test sequences which are then executed by CAN agent inside digital simulator. Reference model has following features:

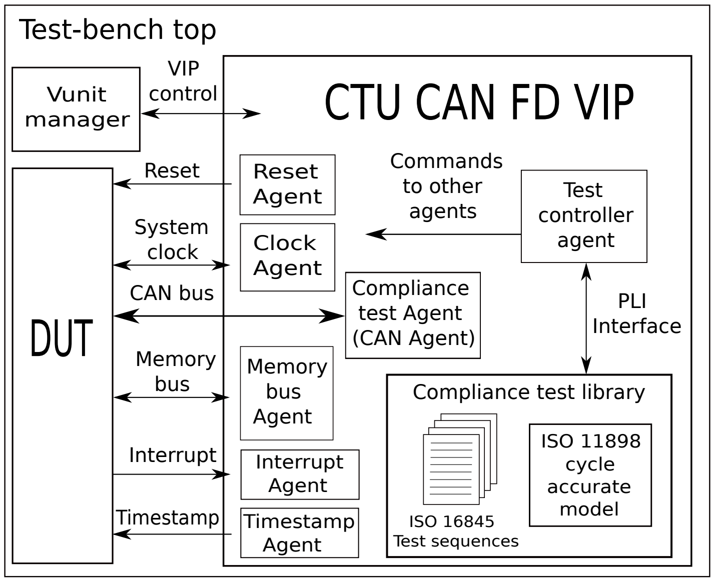

Figure 2.4: Compliance test

- Full support of ISO 11898-1 2015 (all three variants: CAN FD enabled, CAN FD tolerant, Classical CAN)
- Cycle accurate representation of CAN frame. Allows lenghtening/shortening 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 documentation in commercial delivery of this library.

PLI Interface

As PLI interface, VIP supports VPI (GHDL specific) and VHPI interface (IEEE 1076 standardized). PLI interface itself consists of set of signals over which communication is performed. These signals are listed in Table 2.2. Compliance test library, acts as master on this interface. It pushes transactions into shared memory location (inside Compliance test library itself), and simulator side of this interface "picks-up" these request with VPI/VHPI callback on ***pli_clk***, and drives them to PLI signals in VIP. Test controller agent then interprets these signals, and sends commands to target agent via standard communication channel. VIP therefore acta as a slave on PLI interface. This approach guarantees that internal signal handles 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 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 via CAN bus with another instance of CTU CAN FD located inside Feature test agent (Test node), allowing it to invoke various situations inside of 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 overflow occurs upon reception of frame which should fill RX buffer memory (not before), therefore veryfing that overflow occurs properly.

Feature tests use bit rate on CAN bus set by VIPs generics (see 3.2). This bit rate is used for both DUT and Test node.

2.8.3 Reference tests

Reference tests apply bit-sequence (via CAN agent) to **can_rx** of DUT which was recorded from reference controller implementation of CAN bus 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.

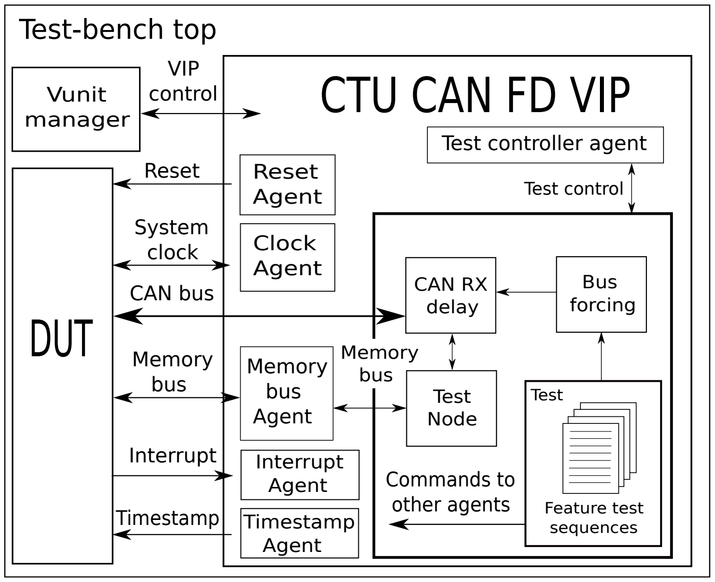

Figure 2.5: Feature test

3. VIP Integration guide

This chapter walks you through steps required for integration of VIP for CTU CAN FD into custom digital TB in standalone mode (2.2). The guide assumes that DUT itself, has already been integrated into the design which is being verified.

3.1 Connection of VIP signals

VIP signals shall be connected as described in Table 3.1.

Signal	Description
clk_sys	Mirror value of ***clk_sys*** DUT pin to this signal. This pin is "inout" of VIP, however, it shall be driven as input.
res_n	Implement force of ***res_n*** of DUT. When ***res_n*** of VIP is 0, ***res_n*** of DUT shall be also 0. (When VIP asserts reset, DUT shall be reset).
int	Mirror value of ***int*** DUT pin to this signal.
can_tx	Connect to ***can_tx*** signal of DUT. This signal shall at any time contain what DUT is transmitting on ***can_tx***.
can_rx	Implement force of ***can_rx*** signal of DUT. Alternatively, implement wired-AND solution (if there are multiple nodes connected internally on a bus). At any time, if VIP is transmitting 0 on ***can_rx***, it shall be present on ***can_rx*** of DUT. If VIP is transmitting 1, it shall be present on DUTs ***can_rx*** unless over-written by 0 of another CAN node.
test_probe	Mirror value of ***test_probe*** of DUT to this signal. This is verification signal only, therefore it can be left unconnected in design.
timestamp	Reffer to 3.6.
write_data	Transactions on this memory interface shall be translated to memory transactions on DUTs memory interface. A potentially best approach is to force related signals on DUT inputs when ***scs*** signal of VIP is 1. If ***scs*** = 0, then DUT memory interface can be driven by a master that it is connected to in a design. It is not recommended to translate transactions on this interface to memory transactions in higher levels of memory bus hierarchy within a system (even if they will eventually end in DUT). Such "indirect" translation would impose delay upon access to DUT, and could affect test results.
read_data	
address	
scs	
srd	
swr	
sbe	
test_start	Shall be driven by test-bench test control mechanism. Reffer to 3.4 for further description.
test_done	
test_success	

Table 3.1: CTU CAN FD VIP interface connection

3.2 Configuration of VIP generics

Generic	Description
test_name	Put name of test to this string. Reffer to [4] for list of tests available in the VIP.
test_type	Put "compliance" for compliance tests, "reference" for reference tests and "feature" for feature tests.
stand_alone_vip_mode	Set to false.
cfg_sys_clk_period	Set to value which corresponds to clock period of DUT on ***sys_clk*** input.
seed	Set to seed which is used for randomization in your TB.
reference_iterations	Leave default (1000).
cfg_brp	Baud rate prescaler - Nominal bit rate (BTR[BRP] register of DUT).
cfg_prop	Propagation segment of bit - Nominal bit rate (BTR[PROP register of DUT).
cfg_ph_1	Phase 1 segment of bit - Nominal bit rate (BTR[PH1] register of DUT)
cfg_ph_2	Phase 2 segment of bit - Nominal bit rate (BTR[PH2] register of DUT)
cfg_sjw	Synchronization jump width - Nominal bit rate (BTR[SJW] register of DUT)
cfg_brp_fd	Baud rate prescaler - Data bit rate (BTR_FD[BRP] register of DUT).
cfg_prop_fd	Propagation segment of bit - Data bit rate (BTR_FD[PROP register of DUT).
cfg_ph_1_fd	Phase 1 segment of bit - Data bit rate (BTR_FD[PH1] register of DUT)
cfg_ph_2_fd	Phase 2 segment of bit - Data bit rate (BTR_FD[PH2] register of DUT)
cfg_sjw_fd	Synchronization jump width - Data bit rate (BTR_FD[SJW] register of DUT)

Table 3.2: CTU CAN FD VIP interface connection

3.3 Linking compliance test library

Compliance test library needs to be first configured and built. Reffer to [3] for build instructions of compliance test library. After compliance test library was built, there are following options how to link compliance test library to simulation:

- VPI interface - GHDL specific, as it is not standardized for VHDL. Can be linked wih "–vpi-lib" option of GHDL.
- VHPI interface - Interface standardized by IEEE 1076. Can be linked to any simulator supporting this interface.

Note that compliance test library is mostly C++ library, however, critical parts for VPI/VHPI interfacing are written in C, therefore guaranteeing that linking mechanisms can find proper C names for VPI/VHPI start-up routines. If your simulator does not support VHPI, reffer to [3] for instructions how to implement connection of compliance test library to simulation.

3.4 Control of test execution

Type and name of test to be executed are selected at compile-time by VIP generics (***test_name***, ***test_type***). Reffer to [4] for list of available test names for each test type. It is up to scripting system calling digital simulator to set these generics.

Start of test execution within a test-bench is done by setting ***test_start*** signal of VIP to 1. Test-bench shall then wait until ***test_done*** = 1. At the time when ***test_done*** = 1, ***test_succes´*** = 1 if the test passed. Otherwise, ***test_success*** = 0. If test fails, reason of failure can be found from logs with error severity in simulator log.

3.5 Selection of CAN bus bit rate

Bit rate on CAN bus can be selected by VIP generics, it is therfore selected at compile time (see 3.2). It is up to scripting system calling digital simulator to set these generics (or hard-code them in VIP instance in TB). bit rate configured in VIP is used by VIP to configure DUT (via Memory bus). Note that this bit rate is used only for compliance tests and feature tests. Reference tests ignore these settings, and always use 2Mbit/500 Kbit with 80% sample point.

3.6 Control of DUTs time flow

DUTs sees "time" flowing in a system in which it is integrated via ***timestamp*** input (reffer to [1, 2]). In an SoC design, this input is probably driven by upcounting unsigned counter measuring flow of time within this SoC. If there is no such capability, ***timestamp*** input of DUT is tied high. VIP contains Timestamp agent which shall drive ***timestamp*** input of DUT. This is typically done in stand-alone mode of VIP operation, since there is no part of the design which drives ***timestamp*** input. However, several feature tests can not function correctly without Timestamp agent generating ***timestamp*** for DUT. For these tests, DUTs ***timestamp*** input shall be forced to value of VIPs ***timestamp*** output. For other tests (compliance tests, reference tests and remaining feature tests), timestamp input can remain driven by design which integrates DUT. Tests which require force of timestamp input are following:

- rx_settings_tsop - Verifies RX buffer timestamp option (timestamp in SOF or EOF).
- timestamp_low_high - Verifies functionality of TIMESTAMP_LOW and TIMESTAMP_HIGH registers.
- tx_arb_time_tran - Verifies time triggered transmission.

3.7 Test specific limitations

Several tests have following limitations when it comes to configuration options:

Compliance tests

BTR[BRP]=BTR_FD[BRP_FD] Limitation states that prescaler for nominal bit rate must be equal to prescaler for data bit rate. This condition must be met for following tests: **iso_7_8_3_1** and **iso_7_8_4_1**.

BTR[BRP]>2 Prescaler for nominal bit rate must be higher than 2. This condition must be met for following tests: **iso_8_8_1_2**, **iso_7_7_11**, **iso_8_7_1**, **iso_8_7_2**, **iso_8_7_4**, **iso_8_7_5**, **iso_8_7_6.**

BTR[BRP]>1 Prescaler for nominal bit rate must be higher than 1. This condition must be met for following tests: **iso_8_7_1**

BTR_FD[BRP_FD]>2 Prescaler for data bit rate must be higher than 2. This condition must be met for following tests: **iso_8_8_1_3**, **iso_8_8_1_4**, **iso_8_8_2_3**, **iso_8_8_2_4**, **iso_8_8_3_1**, **iso_8_8_3_2**, **iso_8_8_4_1**, **iso_8_8_4_2**.

These limitations occur due to lack of input delay compensation in compliance test library implementation.

Bibliography

[1] CTU CAN FD - System architecture

[2] CTU CAN FD - Datasheet

[3] ISO 16845 Compliance test library - Manual

[4] test_list.txt - Test list in CTU CAN FD VIP delivery package