# CAN-CTRL

## CAN Controller Core

### For CAN 2.0B and CAN FD

# Integration Manual

**February 2019**

**IP Product Version**
**7x06N00S00**

**Document Signature**
**CAN-CTRL–INM–7x06N00S00–109**

**CAST, Inc.**

**CONFIDENTIAL**

This core Engineered by:
Fraunhofer IPMS

# Document Version

This document with its associated Release Notes and the Design Specification document applies to the version(s) of the core specified on the cover. See the Release Notes for any updates and additional information not included here.

**Table 1-1 Document Version History**

| Version | Date | Person | Changes from Previous Version |
|---|---|---|---|
| 100 | 2017/02/02 | R.H. | First release (Users Guide split into Design Specification and Integration Manual) |
| 101 | 2017/04/28 | R.H. | Typing corrections |
| 102 | 2017/10/17 | R.H. | Contents of the release package updated<br>Note for netlist release packages added |
| 103 | 2017/12/19 | M.Z. | Linux driver |
| 104 | 2018/01/08 | R.H. | CDC in a subcomponent |
| 105 | 2018/02/12 | R.H. | File <tbench_package.vip.v> removed from release package<br>Hints added to chap. 6.3. |
| 106 | 2018/06/11 | R.H. | Xilinx Vivado GUI project replaced by TCL script |
| 107 | 2018/11/27 | R.H. | Reset synchronizer added to release package<br>Comments about oscillator tolerance added |
| 108 | 2018/12/05 | R.H. | NCSim script for IP compilation added |
| 109 | 2019/02/06 | R.H. | “Hello World” test added<br>Reset synchronizer only available in source code package.<br>Files <src/btl.gif> and <src/btl.fig> removed from release. |

# Table of Contents

**1. Introduction ..... 4**
1.1 Release Packages ..... 4
1.2 Quick start ..... 4
1.3 CAN Bit Timing Calculation ..... 4
1.4 The Source Code IP Core ..... 4
1.5 The Netlist IP Core ..... 5
**2. Testbenches ..... 6**
2.1 Three-node Environment ..... 6
2.1.1 Tbench ..... 6
2.2 CAN Multicore Testbench ..... 7
**3. Synthesis Hints ..... 8**
3.1 General Hints ..... 8
3.2 Xilinx ISE ..... 8
3.3 Too many I/Os ..... 9
3.3.1 Altera Quartus ..... 9
3.3.2 Xilinx Vivado ..... 9
3.3.3 Xilinx ISE ..... 9
3.4 CDC in a Subcomponent ..... 9
**4. Oscillator Tolerance ..... 10**
**5. Contents of the Release Packages ..... 11**
**6. How to use the CAN-CTRL Core ..... 13**
6.1 How to set up the HDL environment ..... 13
6.2 How to use the CAN-CTRL core as microcontroller peripheral ..... 13
6.3 How to transmit or receive a frame ..... 13
6.4 How to learn about the testbench usage ..... 13
6.5 How to do a “Hello World” Test ..... 14
6.5.1 General ..... 14
6.5.2 One Node “Hello World” Test ..... 14
6.5.3 Two Node “Hello World” Test ..... 14
**7. Linux Driver ..... 16**
7.1 Feature Overview ..... 16
7.2 System Configuration ..... 16
7.3 Usage ..... 16
**8. Support ..... 17**

# List of Figures

Figure 2-1 Concept of /tb/tbench.v(hd) ..... 7
Figure 2-2 Testbench for the CAN Multicore ..... 7

# List of Tables

Table 1-1 Document Version History ..... 2
Table 1-1 Structure of the Release Packages ..... 4
Table 3-1 Multi-Cycle Paths inside the CAN Protocol Machine ..... 8
Table 5-1 Files ..... 11

# 1. Introduction

This Integration Manual for the IP Core CAN-CTRL shall be used in addition to the Design Specification. The Design Specification includes all definitions that are used here inside the Integration Manual. The purpose of the Integration Manual is the description of the release package and to give a guide how to use the contents.

## 1.1 Release Packages

The content of the release packages is defined in chap. 4. There are two types of release packages: one package type including source code and one package type including a netlist for a selected FPGA type. All packages are available as VHDL or Verilog packages.

Table 1-1 gives an overview about the release package contents.

**Table 1-1 Structure of the Release Packages**

| Folder | Type | Description |
|---|---|---|
| /doc | Documentation | Main Documentation of the IP core. Please note that the source code also includes inline source code comments for additional documentation. This additional source code comments give additional hints but the main documentation is sufficient for the usage of the IP core. |
| /scripts | Scripts | Simulation scripts for HDL simulators for behavioral (‘RTL) and netlist simulation. These simulation scripts are a good point to start for getting an overview where files are located and which files are needed. It is recommended to look inside the scripts for Cadence Incisive (NCSim), because these are the most advanced scripts. |
| /software | Software | Software example code. This folder includes basic software examples for usage of the core. |
| /src | Sources | HDL source code of the IP core. This includes the IP core as well as the additional host interfaces. |
| /syn | Synthesis | Synthesis scripts, projects and netlists. For source code packages this folder includes example scripts and projects for various synthesis tools. For netlist packages this includes the netlist as well as a simulation model for post synthesis or post place & route simulation. |
| /tb | Testbench | Testbench HDL code. The Verification Specification document gives further details about the type tests, that are included in the release packages. The testbench can be used as example for the integration and simulation of the IP core. |

## 1.2 Quick start

Chap. 6 includes a description of a quick start. This summarizes all the more detailed explanations given in this document and the Design Specification document.

## 1.3 CAN Bit Timing Calculation

A CAN and a CAN FD bus can be configured to virtually any data rate. This flexibility makes it not always simple to chose the right data rate. Furthermore beside the data rate the position of the sample point inside a CAN bit is important. This topic is covered in detail in the Design Specification.

To make it more easy to find a proper CAN bit timing configuration additional to the contents of the release packages the CAN Bit Timing Calculator (an executable program for MS Windows) is delivered. This tool and an Excel calculation sheet, that can be found at folder <doc/>, can help a lot to select a good configuration.

## 1.4 The Source Code IP Core

The best way to find all relevant files and to know how to use them is to read the script <scripts/nc_tbench_vhd.sh> (VHDL) or <scripts/nc_tbench_ver.sh> (Verilog). This shell script is not too difficult to understand and gives a good guide even for users that do not use NCSim.

## 1.5 The Netlist IP Core

The best way to find all relevant files and to know how to use them is to read the script <scripts/nc_tbench_netlist_vhd.sh> (VHDL) or <scripts/nc_tbench_netlist_ver.sh> (Verilog). This shell script is not too difficult to understand and gives a good guide even for users that do not use NCSim.

Please note that this script covers simulation of the synthesis and place & route result including SDF back-annotation. The netlist for using the IP core inside a project is located at an appropriate subfolder of <syn/>.

To ease the integration of the netlist a netlist release package contains a description of the I/O signals of the IP core as a VHDL entity or Verilog stub in the <src/> directory. A Verilog stub is similar to a VHDL entity: it contains only the I/O definitions but no declarations of internal signal, no always blocks and no assignments.

# 2. Testbenches

The testbenches included in the release packages can be used to test the functionality of the core during pre- and post-synthesis simulation. The testbench file list can be found in the folder <tb/> in Table 5-1.

If a testbench finishes without an error then “Simulation successful!” is written to the simulator console.

The concept of the testbenches is a chain of trust: The behavior of the IP core has been tested during development. After embedding the IP core inside a customer’s system a simple “hello world” test developed by the customer is sufficient. After synthesis and place & route Logic Equivalence Check (LEC) and Static Timing Analysis (STA) verify the correctness of the synthesis results. Finally a netlist simulation only has to do a simple “hello world” test to verify that all memories are included and all connections are correct. If this IP core is synthesized separately then the testbenches included in the release packages can be used to do this “hello world” verification.

## 2.1 Three-node Environment

### 2.1.1 Tbench

The “tbench” test examines the host controller interface and the most important features offered by the core to a host. It should be noted that the complete, exhaustive testing of the core has been done for verification purposes, but it is not fully included in the testbench delivered.

The “tbench” test presents an example CAN communication including source comments. Figure 2-1 depicts the concept of the testbench. Each node has its own clock generator and error injector. A master (tb) controls the slave testbench processes (tbslave) for each node. There are two test vectors included:

- example CAN communication, for start-up. This test is enabled when the constant (parameter) “testvec_example” is equal 1. The “testvec_example” can be found in the header of the tbench.v(vhd) file.
- detailed set of vectors to test the most important features and to check the components configured by the pre-synthesis settings (NR_OF_RBUFS, etc.) For more detailed test vector description refer to the testbench source: “tbench.v(hd)”. This test is executed when the constant (parameter) “testvec_postsyn” is equal 1. The “testvec_postsyn” can be found in the header of the tbench.v(vhd) file.

Tests can be turned on/off and comments are included at the beginning of the testbench source code. Search in `tbench.v(hd)` for the string “START HERE” to get an entry point into the test bench.

The delivery package includes only a small, but sufficient set of tests in the three-node testbench. The CAN-CTRL core has been exhaustively tested with the three-node testbench, but these testcases are not included in the delivery package.

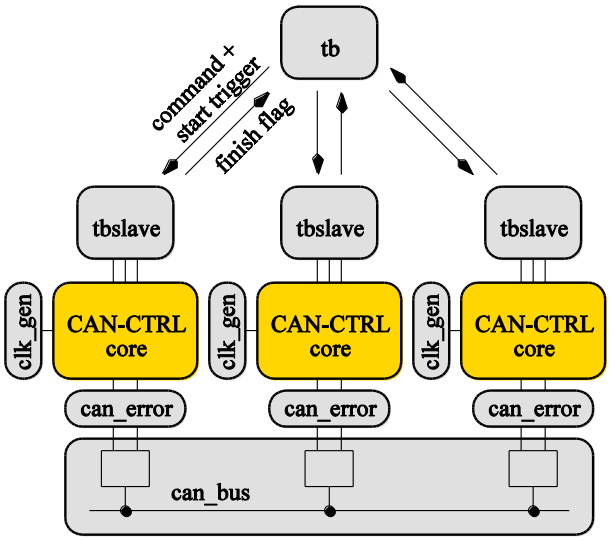


**Figure 2-1 Concept of /tb/tbench.v(hd)**

## 2.2 CAN Multicore Testbench

For the CAN multicore container (see the Design Specification, chapter “CAN Multicore”) a simple testbench is included in the delivery package. It connects 3 CAN multicore containers to one single CAN bus. Each container includes 2 CAN-CTRL cores (Figure 2-2).

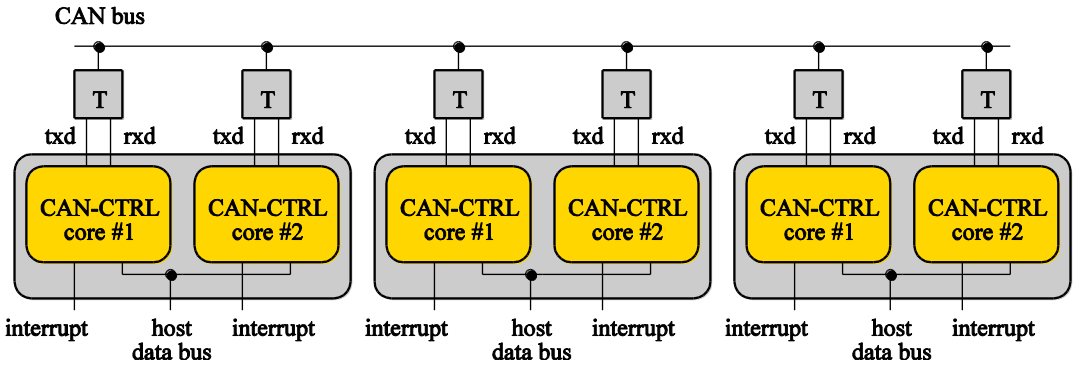


**Figure 2-2 Testbench for the CAN Multicore**

The testbench for the CAN multicore verifies the system only with a very basic communication test. This is sufficient because the CAN-CTRL core has been exhaustively tested.

# 3. Synthesis Hints

## 3.1 General Hints

The core is a fully synchronous design having asynchronous resets. Only the top level signals <host_rst_b>, <can_rst_b> and <timer_rst_b> are used as asynchronous resets. There is no reset synchronizer inside the IP core. If these signals are driven by a true asynchronous source then it is recommended to add reset synchronizers outside of the IP core.

Three clock domains exist in the core: one for the host clock (input <host_clk>), one for the CAN clock (input <can_clk>) and one for the CiA 603 timer (input <timer_clk>). The top level I/O signals have the prefix “host_” if they are in the host clock domain and “timer_” if they are in the timer clock domain. Appropriate clock domain crossing is included in the core. Therefore all signals between the clock domains should be declared as false paths during synthesis and the clocks should be treated asynchronously. The clock frequency for the CAN domain should be chosen as recommended in the Design Specification, chapter “CAN Bit Time”. The host clock frequency is the frequency of the host interface. If necessary it is possible to use only one clock source for all of the three clocks which makes them all synchronous.

Example synthesis scripts and projects are included in the release package. Readme files are included as well as comments in the scripts. The synthesis examples also include constraint files. The synthesis projects can be used as examples if synthesis needs to be done with a different tool.

For FPGA synthesis tools it is handy to copy the sources into a subdirectory of the synthesis project. For Verilog sources this is necessary as some files are referenced by a Verilog ‘include directive (this is the Verilog method for doing something similar to VHDL packages). Because of this the sources are needed to be copied to the synthesis subdirectory. The release packages include Unix shell scripts for this work. These scripts are easy to read even if there is no Unix background and can be used as example.

The release package includes source code for a synchronous memory. FPGA tools will synthesize this to BlockRAM. For ASIC synthesis this would result in a big array of flipflops which would be only acceptable for small amounts of memory. Otherwise this source needs to be replaced by a RAM core.

Parts of the CAN protocol machine inside the core only take actions at the sample point of a CAN bit or at the start of a CAN bit. (The top level outputs <tst_sample> and <tst_clock> signal these events and the signals <can_core_nobuf_i/btl_i/sample_i> and <can_core_nobuf_i/btl_i/clock_i> are the source of these signals.) Therefore a lot of paths are multi-cycle paths. Table 3-1 gives the number of clock cycles for these multi-cycle paths. Using this as constraint will relax the synthesis.

**Table 3-1 Multi-Cycle Paths inside the CAN Protocol Machine**

| From | To | CAN clock cycles |
|---|---|---|
| Sample Point | Sample Point | 4 |
| Sample Point | CAN Bit Start | 2 |
| CAN Bit Start | Sample Point | 2 |
| CAN Bit Start | CAN Bit Start | 4 |

## 3.2 Xilinx ISE

Xilinx ISE is an obsolete tool, but it is required to use it for FPGAs that are not supported by Xilinx Vivado. The latest version of ISE (14.7) includes a bug when synthesizing a memory, that is bigger than one BRAM instance. ISE reports the synthesis as “completed” and timing as met, but post synthesis simulation fails. The bug occurs for true dual-port memories during a write access to port B. This write access is not executed in the simulation model.

CAN-CTRL includes a synthesizable memory model (<src/memory.v(hd)>) that includes a workaround for this bug of Xilinx ISE. The workaround cuts a memory into parts if its size exceeds the size of a BRAM. The size of one BRAM depends on the used FPGA and can be set using the constant <ISE_BLOCK_SIZE> in the source code of the memory model.

## 3.3 Too many I/Os

Most real applications use CAN-CTRL as a subcomponent and therefore only the two signals to the CAN transceiver are relevant for I/O pin planning. But for evaluation synthesis to a given FPGA an IP core is often synthesized as top component which results in the fact that all I/Os need to be mapped to pins. Such synthesis projects are also included in the source code release package and these projects are also used to generate the netlists, that are included in FPGA netlist release packages of CAN-CTRL.

CAN-CTRL includes so many I/O signals, that a standalone FPGA synthesis run with place & route stops because of not enough I/O pins for a given FPGA device. Workarounds for this problem for different FPGA tools are described in the following subchapters.

### 3.3.1 Altera Quartus

As a workaround to avoid too many I/Os the example synthesis project for Quartus uses an additional component as a hull around CAN-CTRL. This hull is similar to a boundary scan-chain and just maps a lot of the inputs of CAN-CTRL to shift registers. This hull (file <src/fpga_boundary.v(hd)>) has no behavioral meaning and should be ignored. The IP core CAN-CTRL itself is defined as a design partition. This design partition is used to generate the <qxp> netlist, that is included in the Altera netlist delivery package.

For post place & route simulation a Verilog netlist is generated, that contains both the outer hull as well as the inner IP core. Only the inner IP core is of interest and used by the simulation.

### 3.3.2 Xilinx Vivado

Vivado offers the Out Of Context (OOC) synthesis feature. Using OOC synthesis the I/Os of an IP core are not mapped to I/O pins but place & route is still possible. The example synthesis scripts for Vivado are configured to OOC synthesis.

### 3.3.3 Xilinx ISE

ISE offers to generate a post translate netlist without I/Os, which is used as netlist in the FPGA netlist delivery package and a post place & route netlist with I/Os. To avoid too many I/Os for place & route the example synthesis project for ISE uses an additional component as a hull around CAN-CTRL. This hull is similar to a boundary scan-chain and just maps a lot of the inputs of CAN-CTRL to shift registers. This hull (file <src/fpga_boundary.v(hd)>) has no behavioral meaning and should be ignored.

For ISE it is necessary to set CAN-CTRL as the top component for the translate process without I/Os and to set the outer hull as top component for the place & route process with I/Os.

It is necessary to keep the hierarchy if after place & route a Verilog netlist is required for post place & route simulation to separate the hull from CAN-CTRL.

## 3.4 CDC in a Subcomponent

Clock Domain Crossing (CDC) is done using 2 flipflops for synchronization for single data bits which need to be transferred into a different clock domain. If several bits need to be transferred into a different clock domain in parallel then single bit CDC signals are used for handshaking.

Single bit CDC using 2 flipflops is done in a separate subcomponent located at <src/cdc.v(hd)>. This gives the opportunity to replace these flipflops with specific ones from the target library as well as to set synthesis constraints for only this subcomponent.

# 4. Oscillator Tolerance

Oscillator tolerance is a complex topic and there is no simple answer for the question how good the oscillator should be, that drives <can_clk>.

The old CAN 2.0B standard (“classic CAN”) defines a fixed tolerance value in chap. 9 - "Oscillator Tolerance" of 1.58%.

For CAN FD it is much more complex. The CAN FD specification (ISO 11898-1:2015) defines in chap. 11.3.2.5 - "Tolerance range of the oscillator frequencies" a set of 5 equations to calculate the clock tolerance with respect to the desired data rate.

CAN in Automation (CiA) has gone into more detail about this and the result is the document CiA 603-1 "Node and system design" <https://www.can-cia.org/can-knowledge/can/cia601/>. This document explains this in greater detail and gives hints about how to improve the system for higher data rates. Connected to the CiA 601-3 PDF document there is a complex MS Excel calculation sheet, that takes into account the desired CAN bit timing configuration, the used transceiver, the bus topology, the PCB and the oscillator tolerance and calculates the margin for the desired data rate.

The mentioned documents are protected by copyright and therefore no information from them can be extracted. It is recommended to follow the detailed explanations and recommendations of CiA 601-3.

# 5. Contents of the Release Packages

The source code of the core is available in VHDL (*.vhd) or Verilog (*.v). Therefore the files are named using the operator “v(hd)” in Table 5-1.

Files included in all release packages are marked without a color highlight in Table 5-1, files only included in source code packages are marked red and files only included in netlist packages are marked orange. Files related to the optional CAN multicore are only included in the release package if purchased and this is marked with blue color.

**Table 5-1 Files**

| Folder | File(s) | Description |
|---|---|---|
| / | LOAD_EDA | Dummy file for tool setups (used by scripts) |
| /doc | can-ctrl-des.pdf | Design Specification |
| | can-ctrl-inm.pdf | Integration Manual (this document) |
| | can-ctrl-ves.pdf | Verification documentation |
| | can-ctrl_rel.pdf | Release notes |
| | can-ctrl_tests.xlsx | List of testbench test for verification document |
| | can-ctrl-memory.xlsx | Calculation of memory size |
| | CBC.xlsx | CAN Bit timing Calculation sheet |
| | readme_netlist.txt | Readme for netlist packages |
| /scripts | cds.lib | NC-Sim library definition |
| | hdl.var | NC-Sim file |
| | ms_init.tcl | ModelSim simulation: initialization file used by the other scripts |
| | ms_tbench_multicore_v*.tcl | ModelSim simulation: tb/tbench_multicore.v(hd) |
| | ms_tbench_netlist_v*.tcl | ModelSim simulation: tb/tbench.v(hd) (netlist) |
| | ms_tbench_v*.tcl | ModelSim simulation: tb/tbench.v(hd) |
| | nc_prepare_ip.sh | NC-Sim script for IP compilation |
| | nc_sdf*.cmd | NC-Sim SDF command files (see scripts/nc_tbench_netlist_v(hd).sh) |
| | nc_tbench_ahb_v*.sh | NC-Sim simulation: tb/tbench_ahb.v(hd) |
| | nc_tbench_apb_v*.sh | NC-Sim simulation: tb/tbench_apb.v(hd) |
| | nc_tbench_multicore_v*.sh | NC-Sim simulation: tb/tbench_multicore.v(hd) |
| | nc_tbench_netlist_v*.sh | NC-Sim simulation: tb/tbench.v(hd) (netlist) |
| | nc_tbench_v*.sh | NC-Sim simulation: tb/tbench.v(hd) |
| | nc_trace*.cmd | NCSim command files (see scripts/nc_tbench_*.sh) |
| | readme.txt | Readme for scripts |
| /software | The software example is specific for the compiler “IAR embedded workbench” for a MSP430-compatible microcontroller. It can be easily adapted for different host controllers. | |
| | Msp430/can.h | Definition header file for host controller |
| | msp430/isr_compat.h | Definition header file for host controller |
| | msp430/mcu.h | Definition header file for host controller |
| | msp430/example/* | Example application for host controller (2 CAN nodes) |
| | Linux driver | |
| | linux_driver/device_tree/* | Device-Tree Template and Example for using Linux Driver |
| | linux_driver/driver_module/*.c | Linux Driver Module ipms_canfd_driver.c |
| /src | interface/can_amba_ahb.v* | AMBA AHB interface |
| | interface/can_amba_apb.v* | AMBA APB interface |
| | interface/can_sync_08.v* | Synchronous 8 bit host interface |
| | interface/can_wishbone_32.v* | Wishbone 32 bit interface |
| | acf_ctrl.v(hd) | acceptance filters |

| Folder | File(s) | Description |
|---|---|---|
| | arithmetic_package.v(hd) | Tools for HDL source calculations |
| | btl.v(hd) | Bit timing logic |
| | can_core_nobuf.v(hd) | CAN-CTRL core without buffers (without memories) |
| | can_ctrl.v(hd) | CAN-CTRL top level of the IP core |
| | can_ctrl_entity.v(hd) | CAN-CTRL top level of the IP core entity for netlist handling |
| | can_multicore.v(hd) | CAN multicore |
| | can_package.v(hd) | Common settings and declarations |
| | can_package_synparam.v(hd) | Configurable synthesis parameters (actual selection) |
| | can_package_synparam_vip_*.v(hd) | Synthesis parameters chosen for Verification IP |
| | cdc.(hd) | Clock domain crossing plus optional additional buffer |
| | crc_check.v(hd) | CRC checker |
| | eml.v(hd) | Error management logic |
| | fpga_boundary.v(hd) | Hull component to ease FPGA example synthesis projects |
| | host_interface.v(hd) | Host interface |
| | iml.v(hd) | Interface management logic |
| | memory.v(hd) | Memory model (BlockRAM, DistributedRAM) |
| | ram_ctrl.v(hd) | Controller for internal buffer RAM |
| | rbuf.v(hd) | Receive buffer |
| | reset_sync.v(hd) | Reset synchronizer |
| | status_buffer.v(hd) | Receive buffer: status buffer (frame header) |
| | tbuf.v(hd) | Transmit buffer control (PTB and STB) |
| | tcl.v(hd) | Transmit control logic (CAN protocol machine) |
| | timer_ttcan.v(hd) | TTCAN timer |
| | tstamp_ctrl.v(hd) | Time-stamping controller for CiA 603 |
| /syn | * | Source package: Example synthesis projects / scripts<br>Netlist package: Specific synthesis netlist |
| /tb | can_bosch_tb_package.vhd | VHDL Package for extended test features for Bosch testbench (only for VHDL release) |
| | can_bus.v(hd) | CAN bus model for /tb/tbench.v(hd) |
| | can_error.v(hd) | CAN bus error stimulation for /tb/tbench.v(hd) |
| | can_tran.v(hd) | CAN transceiver model for /tb/tbench.v(hd) |
| | clk_gen.v(hd) | Clock generator for /tb/tbench.v(hd) |
| | tbench.v(vhd) | Behavioral testbench |
| | tbench_ahb.v(hd) | Testbench for AMBAP AHB wrapper |
| | tbench_apb.v(hd) | Testbench for AMBAP APB wrapper |
| | tbench_multicore.v(hd) | Behavioral testbench for multicore container |
| | tbench_multicore_package.v(hd) | Behavioral testbench for multicore container (package) |
| | tbench_package.v(hd) | Behavioral testbench (package) |
| can-ctrl_cbc.tar.gz | | CAN Bit timing Calculator (zipped Windows software tool) |

# 6. How to use the CAN-CTRL Core

The following subchapters include a brief description on how to use the CAN-CTRL core. These subchapters do not replace the detailed descriptions, but are dedicated to give a quick look at the possible core's usage.

## 6.1 How to set up the HDL environment

- Compile the sources to your working library.
- Make an instance of the IP core top module `can_core` in your system (file `src/can_core.v(hd)`).
- If simulation is required it is very important to use a CAN bus model. It is suggested to use the model from `tb/can_bus.v(hd)` as a basis for your own modifications.
  It should be noted that the bus model `tb/can_bus.v(hd)` is used in the testbench, therefore it is strongly recommended to make a copy of that file before modifying.

## 6.2 How to use the CAN-CTRL core as microcontroller peripheral

- Make an instance of the CAN-CTRL core as a peripheral component.
- Have a look at `can.h` and `example.c` in the directory `software/example`.
- Refer to chapter 6.3 for further hints.

## 6.3 How to transmit or receive a frame

- Set up the CAN bit timings (see the Design Specification, chapter "CAN Bit Time").
  After a hardware reset the CAN-CTRL core is automatically configured to $f_{BUS} = 1\,\mathrm{Mbaud}$ with a system clock of $16\,\mathrm{MHz}$.
- Disable RESET.
- If necessary: Enable / Disable required interrupts.
  - For transmission: TPIE or TSIE.
  - For reception: RIE and if necessary RAFIE, RFIE, ROIE
- For transmission:
  - Write a frame to a transmit buffer (PTB or STB). (Take care of TBSEL and TSNEXT.)
  - Enable the transmission: use TPE or TSONE or TSALL.
  - If necessary wait for the selected interrupt.
- For reception:
  - Wait for the selected interrupt.
  - Read the received frame from the RB and use register RCTRL to determine the type of frame.
- Additional features:
  - Acceptance filter (see the Design Specification, subchapter "Acceptance Filters").
  - Buffer control: TSSTAT and RSTAT
  - Error handling (bits EIE, EIF, bits EWL register ERRINT, RECNT, TECNT, EALCAP)

## 6.4 How to learn about the testbench usage

- The 3-node testbench is documented in the chapter 2.1.1 .
- Search in `tbench.vhd` for the string "START HERE" to get an entry point into the test bench.

- Follow the instructions in the comments and use the string search to jump to the pointed locations.

## 6.5 How to do a "Hello World" Test

### 6.5.1 General

This chapter is a guideline to do a first "Hello World" test with the developed system where CAN-CTRL is included. In contrast to other chapters only little knowledge about the behavior and features of CAN-CTRL is required. In other words: this chapter is dedicated to hardware system developers, who may not have detailed knowledge about the application.

The best way to a "Hello World" test with a communication system is to use two instances and connect it to each other. This is called a "two node test". If the effort to do such a test is considered too high, then CAN-CTRL can be tested with a "one node test" too.

### 6.5.2 One Node "Hello World" Test

The basic idea behind this test is to use the internal loop-back mode (LBMI).

- Set RESET=0.
- Write a frame into the transmit buffer TBUF:
  - Address of TBUF: 0x12345678 ID
  - Address of TBUF+4: 0x00000080 IDE=1, DLC=0
- Set LBMI=1, TPE=1. (This starts a transmission.)
- Wait until interrupt (output signal <host_irq>). Several interrupt flags may be set.
  - Check if RIF=1. (Receive interrupt flag has been set. Ignore other interrupt flags.)
- Read the received frame from the reception buffer RBUF:
  - Address of RBUF: 0x12345678 ID
  - Address of RBUF+4: 0x00001080 IDE=1, DLC=0, TX=1
- Set RESET=1. (This clears pending interrupts, and marks all RBUF slots as empty.)

Running this test verifies the connection of the host interface between the host controller and CAN-CTRL as well as the interrupt signaling. It further verifies the connection of <host_clk> and <can_clk>. It does not check the connection between CAN-CTRL and the top-level I/O to the CAN transceiver <txd> and <rxd>.

### 6.5.3 Two Node "Hello World" Test

This test requires to set up a CAN bus:

- A real world application requires a transceiver for each node and a 120Ω termination resistor at each end of the bus. (This two node network will most likely also work without termination resistors.)
- In simulation it is required to use a CAN bus and a transceiver simulation model, which are included in the release package: <tb/can_bus.v(hd)> and <tb/can_tran.v(hd)>. Please note, that the bus simulation model is designed for 3 CAN nodes. For a two node test the third <txd> signal should be set to logic '1'. An example of how to use these simulation models can be found in <tb/tbench.v(hd)>.

The following steps are similar to chap. 6.5.2 , but not equal. It is assumed, that both connected nodes are using the same CAN bit timing configuration. If both nodes are identical systems including CAN-CTRL, then this requirement is met.

- Set RESET=0 for both nodes.
- Write a frame into the transmit buffer TBUF of node #1:
  - Address of TBUF: 0x12345678 ID

  - Address of TBUF+4: 0x00000080 IDE=1, DLC=0
- Set TPE=1 at node #1. (This starts a transmission.)
- At node #2 wait until interrupt (output signal <host_irq>). Several interrupt flags may be set.
  - Check if RIF=1. (Receive interrupt flag has been set. Ignore other interrupt flags.)
- At node #2 read the received frame from the reception buffer RBUF:
  - Address of RBUF: 0x12345678 ID
  - Address of RBUF+4: 0x00000080 IDE=1, DLC=0
- Set RESET=1 for both nodes. (This clears pending interrupts, and marks all RBUF slots as empty.)

# 7. Linux Driver

## 7.1 Feature Overview

Feature list:

- Platform Driver[1]
- Device Tree based configuration of system[2]
- Based on SocketCAN networking stack[3]
- CAN and CAN FD supported
- Usage of primary transmit buffer (PTB)

A linux platform driver is provided for CAN controller core. The platform description is outside the scope of the driver module. The user is responsible for describing the system using a Device Tree based description or any other way of defining the platform devices. The driver is based on the SocketCAN networking stack. This makes the applications independent of the used CAN controller.

## 7.2 System Configuration

The description of the system can be done using the Device Tree. Users are encouraged to read available documentation about Device Tree based configuration. For using the driver two additional files are provided, a template file and an example file.

The field *compatible* is used for matching linux driver module and platform device. It can be either “ipms,can” for the CAN core without CAN FD support or “ipms,canfd”.

The field *reg* describes memory base address and length. The system designer is responsible for giving the correct address.

The fields *interrupt* and *interrupt-parent* describe the used interrupt number and interrupt type of the CAN core. Hint for Zynq-based system: subtract Zynq-Interrupt ID by 32 as the interrupt number of the device tree is related to the ID of the interrupt controller “*intc*”.

The field *clock* is used to describe the used clock for the can_clk of the core, so the driver knows the frequency of the CAN clock. The correct frequency is needed for bittiming calculation. The given example “clocks = <&clkc 15>;” describe clock FCLK_CLK0 of a Zynq-based system.

## 7.3 Usage

Make sure, the used Linux Kernel supports SocketCAN. Also enable “iproute2” in the kernel configuration. The driver module has to be loaded either automatically by the system or manually by the user (command insmod). Then there will be one or more devices called can0, can1 and so on.

The timing of the CAN devices must be configured using the “ip” command. Following example configures device can0 using 125kbit/s arbitration rate and 2 Mbit/s FD data rate: “ip link set can0 type can bitrate 125000 dtq 50 dprop-seg 8 dphase-seg1 8 dphase-seg2 2 dsjw 2 fd on”

The devices can be enabled and disabled, also using the “ip” command. Use “ip link set can0 up” for enabling the device 0. Please also check the SocketCAN documentation for the usage of the “ip” command.

There are already applications available for testing CAN communication using SocketCAN[4]. Especially *cansend* and *candump* can be used for testing the system.

[1] https://www.kernel.org/doc/Documentation/driver-model/platform.txt
[2] https://elinux.org/Device_Tree_Usage
[3] https://www.kernel.org/doc/Documentation/networking/can.txt
[4] https://github.com/linux-can/can-utils

# 8. Support

Every effort has been made to ensure that this core functions correctly. If a problem is encountered please contact CAST:

CAST, Inc.
11 Stonewall Court
Woodcliff Lake, New Jersey 07677 USA

Technical Support Hotline: +1-201-391-8300 ext. 2

Fax: +1 (845) 818-3767
E-mail: support@cast-inc.com
URL: www.cast-inc.com