BUS INTERFACE

INVENTRA™

THE INTELLIGENT APPROACH TO INTELLECTUAL PROPERTY

MCAN2

CAN 2.0 NETWORK CONTROLLER

User Guide

SOFT CORES

05/06 PU-40107.002-FC

For Customer Support on this product:

- Call up the Inventra Customer Inquiry Service at **http://www.mentor.com/supportnet**
- Email **support_net@mentor.com**
- Phone **1-877-763-8470** (toll-free in US, Mexico and Canada)
  (Customers in other parts of the world can access this service via the AT&T USA Direct service in their country.)

Full details are given in the Customer Support Handbook, provided in Adobe Acrobat format as **custhb.pdf** in the **/databook** directory on the Inventra™ Soft Cores CD. Please note the checklists of actions to take and information to have to hand when contacting Inventra Customer Support that are given in the Customer Support Handbook.

TABLE OF CONTENTS

1. INTRODUCTION

The Inventra™ MCAN2 is a stand-alone controller for the CAN Controller Area Network used in the automotive industry and in a number of other industrial environments. It provides an interface between a microprocessor and a CAN bus which carries out all the actions of data encoding/decoding (including serialisation/deserialisation of data, bit stuffing/unstuffing), message management (acceptance filtering, acknowledgement, error detection and signaling, and re-transmission), bit timing and synchronization involved in transmitting and receiving information over a CAN network.

The MCAN2 controller implements the BOSCH CAN message transfer protocols 2.0A and 2.0B. Specification 2.0A is equivalent to the earlier CAN 1.2 specification and refers to standard message formats (11-bit identifier); Specification 2.0B refers to both standard and extended message formats (both 11-bit or 29-bit identifiers). The MCAN2 can therefore be used in both CAN 2.0 and CAN 1.2 systems.

The MCAN2 is compatible with a Philips SJA1000 working in its PeliCAN mode but with some exceptions (detailed in Section 8 of the MCAN2 Product Specification). In particular, the design has a synchronous PVCI[1]-compatible CPU interface for ease of connection to a range of microprocessor buses.

The MCAN2 soft core is supplied as Verilog and VHDL RTL files, together with functional testbenches, synthesis scripts and scan test scripts. The basic function of the MCAN2 is described in the Product Specification. The purpose of this User Guide is to explain how to implement and adapt the MCAN2 design for use in a target application. A Programmer's Guide is also provided which describes the software interface to the MCAN2 and explains how a typical application should drive the core.

Note: The BOSCH specification uses the convention that, on the CAN bus, Recessive bits are logic '1' while Dominant bits are logic '0'. This convention is also used in the MCAN2 design.

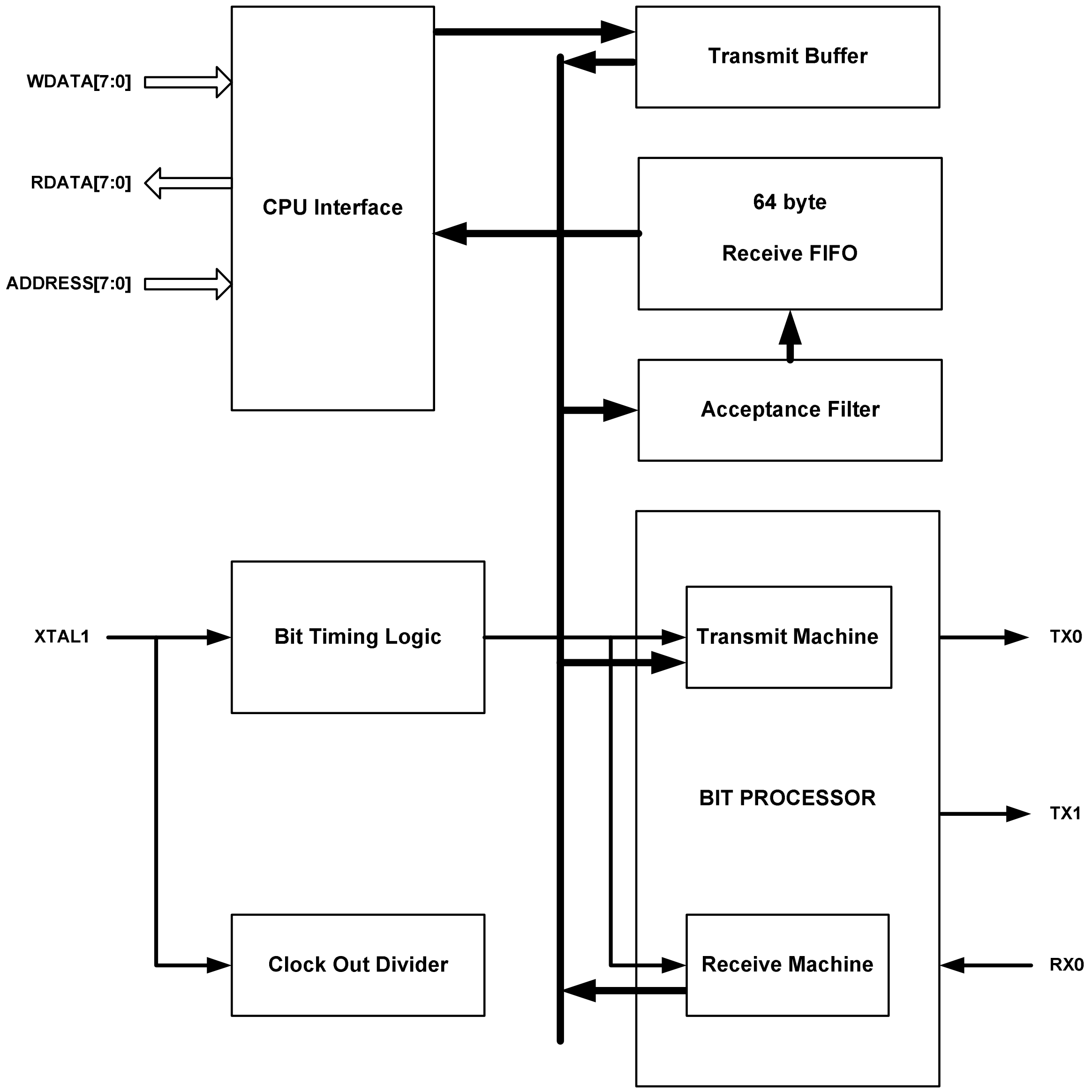
1.1. MCAN2 FEATURES

- Supports full CAN (CAN 2.0 A and CAN 2.0B)
- Supports both 11-bit and 29-bit identifiers
- Supports bit rates from less than 125Kbaud to more than 1Mbaud
- 64 byte Receive FIFO
- Software-driven bit-rate detection (offering hot plug-in support)
- Acceptance filtering
- Single-shot transmission option
- Listen-only mode
- Reception of 'own' messages
- Self Test option
- Error interrupt generated for each CAN bus error
- Arbitration lost interrupt with record of bit position
- Read/write error counters
- Last error register
- Synchronous PVCI[1]-compatible interface for easy connection to a range of microprocessors
- Programmable error limit warning
- Verified against BOSCH CAN2.0 test suite

[1] Peripheral Virtual Component Interface, as defined by the VSI Alliance™ Virtual Component Interface Standard (OCB 2 1.0).

1.2. BLOCK DIAGRAM

The following block diagram shows the main functional blocks of the MCAN2.



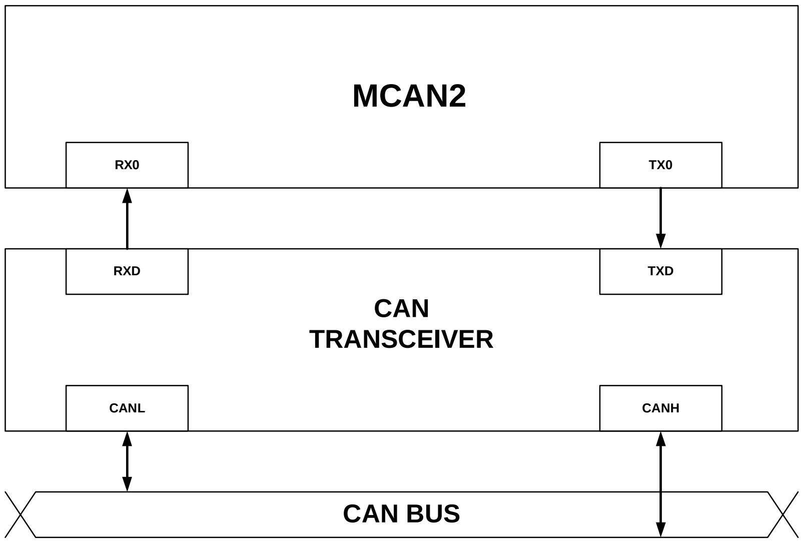
1.3. INTENDED APPLICATION

Figure 2 below shows how the MCAN2 core is intended to be used.

The MCAN2 sits between a Host microprocessor and a standard CAN bus transceiver. The transceiver is responsible for putting logical levels from the MCAN2 onto the CAN bus. The CAN bus is usually a twisted pair which is fed into the differential inputs of the CAN transceiver.

The figure below shows TX0 driving the CAN transceiver. This is the usual arrangement. However, the TX1 output, which can be programmed to output either the inverse of TX0 or the Transmit clock, gives the system designer the option of creating their own interface to the CAN bus instead of using a standard CAN transceiver. Note that the host CPU has not been drawn.

Figure 2. MCAN2 interface to the CAN bus



1.4. SYSTEM DESIGN ISSUES

CAN Bus Bit Rate: CAN systems may be designed to use a wide range of bit rates, not just the 125kBaud – 1MBaud range picked out by the BOSCH specification. The bit rates supported by the MCAN2 depend on the speed at which the MCAN2 is run and on bus timing parameters which together can give a factor of about 200 between the highest bit rate and the lowest bit rate supported by the chosen clock. A clock speed of 50MHz, for example, can support bit rates between 15kBaud and 3MBaud.

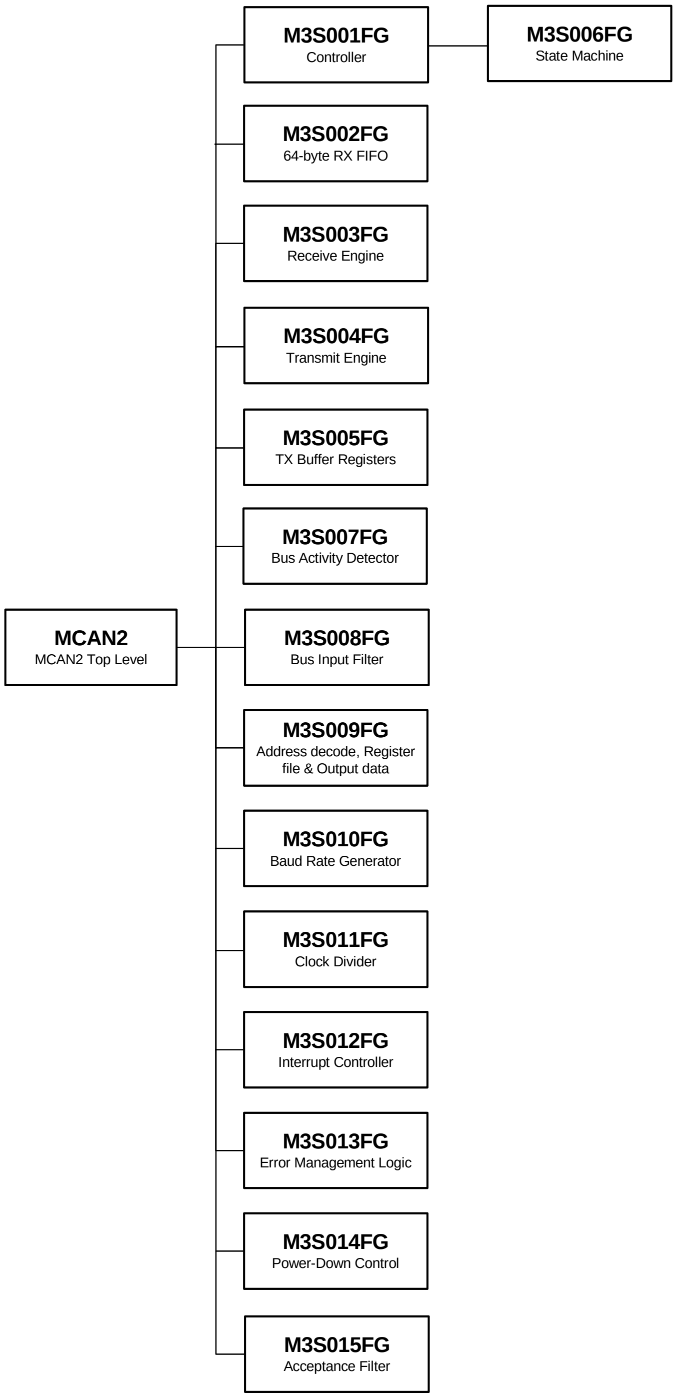
The bit rate that can be used is however affected by the bus length. A bit rate of 1MBaud can only be used where the bus length is less than about 40 metres. A bit rate of 500kBaud, however, can be used on buses up to 100 metres in length while a bit rate of 125kBaud can be used on buses up to 500 metres in length. You should also note that the use of higher bit rates imposes additional requirements on pulse slopes and oscillator stability.

Number of Nodes: A CAN network can in theory support as many nodes as there are separate identifiers of which there are 2048 standard identifiers and 536,870,912 extended identifiers. In practice the number of nodes is limited to between 32 and 64 by the drive capabilities of the individual nodes.

Further information on CAN system design is available from a wide range of web sites including the BOSCH web site at **http://www.can.bosch.com** *and the CAN in Automation (CiA) Users Group web site at* **http://www.can_cia.de**.

1.5. TREE DIAGRAM

The following tree diagram shows the hierarchical structure of the MCAN2 modules included in the **rtl** directory.

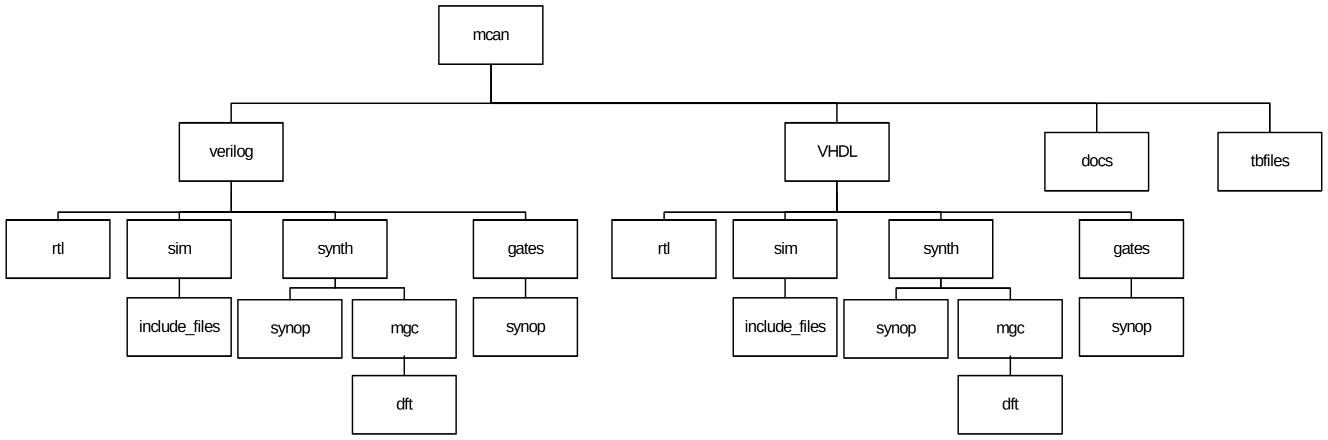


Note: The **rtl** directory also includes a **mcan2_state.v(hd)** file which defines the states of the design's top-level state machine.

2. CORE DELIVERABLES

This section describes the files supplied in support of the MCAN2 design. Not all of the files supplied will be relevant to your design flow. You should use this section as a reference to determine which files and libraries you will need.

The files are provided in a directory named after the design, which is chiefly divided into Verilog and VHDL subdirectories as shown in the diagram below.



- The **rtl** directories contain the source code.
- The **sim** directories contain the testbenches for the design and scripts for simulation using ModelSim or Verilog XL, with the detailed tests provided in the **include_files** subdirectories.
- The **synth/synop** directories contain synthesis scripts and scan insertion scripts for synthesizing the core in the design's reference technology using Synopsys Design Compiler and Test Compiler.
- The **synth/mgc/dft** directories contain scripts for scan insertion and ATPG using Mentor Graphics' DFTAdvisor and FastScan.
- The **gates/synop** directories contain the netlists and standard delay files produced by synthesis in the design's reference technology.
- The **docs** directory contains documentation specific to the core – including a product specification, a programmer's guide and this user guide.
- The **tbfiles** directory contains reference files for the testbench.

Version-specific information about the core is given in a **mcan2.readme** file in the top-level **mcan2** directory.

Note: The files are provided encrypted for reasons of commercial security, so before you can use the MCAN2, the files have to be extracted and decrypted using the MacroLib™ software. For guidance in using the MacroLib™ software, please refer to the 'Easy Start Guide'. Once extracted, all the files will be human readable.

3. DESIGN CONVENTIONS

This section describes the overall features of designs in the Inventra™ Soft Core Library such as the MCAN2.

3.1. COMPATIBILITY

Many of the designs within the Inventra™ Soft Core Library have, as a first priority, been designed to be compatible with the target device on which they have been modeled. This means that they are binary compatible at the register-bit function level and that, in the case of Microcontroller and DSP designs, they are where possible clock-for-clock compatible with the original part. In the case of Peripheral Controllers, the exact timings of the CPU – I/O channel interface may differ from the target, but the difference should not affect system behavior.

As a second priority, the cores have been designed to be technology independent i.e. the cores have been developed to be implemented in any silicon process.

Last but not least, the cores have been designed to be EDA system independent. This means that they do not contain tool specific constructs and this has been verified by passing them through several leading point tools.

3.2. PRODUCT SPECIFICATION

The Product Specification supplied for cores such as the MCAN2 has a standard format that is intended to make it straightforward to find the information you require. It has been written in the style of a semiconductor manufacturer's datasheet since this is something which hardware engineers are familiar with, and typically contains the following information:

3.2.1. DESIGN FEATURES

The main features of the soft core are highlighted as bullet points. These points are expanded with a short discussion and illustrated with a block diagram to enable you to quickly establish whether the function is suitable for your requirements.

3.2.2. DESIGN HIERARCHY

The hierarchy of the design is described using a hierarchical tree diagram such as that shown in Section 1.5 of this User Guide. The top-level module of the design is shown as the left-most box with the lower level blocks fanning out to the right.

Each box in the diagram contains the module name and a brief description of the module's function is given.

3.2.3. SIGNAL DESCRIPTION

The inputs and outputs to the core are illustrated in a top-level diagram. Inputs to the core are typically shown on the left-hand side with the outputs shown on the right-hand side. Buses are denoted by thicker lines.

This top-level diagram could be used as the basis of a schematic symbol.

This is followed by a table describing the function of the signals. This lists the names of the signals, whether they are inputs or outputs, and gives a brief description of their purpose.

3.2.4. DETAILED INFORMATION

Detailed information on the function of the device, giving a description of any software interface at the register-bit level, is given in the main body of the Product Specification.

3.2.5. SPECIAL FEATURES

The detailed functional description is followed by a section on special features, giving information on enhancements to, or differences from, the reference device.

3.2.6. TIMING DIAGRAMS

Timing diagrams are included in a separate section towards the end of the Product Specification – but it is important to note that these diagrams do not contain specific timing information since it is not possible to predict the timing of a design in any particular target technology. What the diagrams do show are the relationships between signals in terms of where a transition on one signal causes another signal to change.

3.3. NAMING COVENTIONS

3.3.1. TOP-LEVEL MODULE NAMES

The top-level module is named after the design. Thus the top-level module of the Verilog version of the RTL code is provided as the file **mcan2.v**, while the top-level module of the VHDL version is provided as the file **mcan2.vhd**.

3.3.2. DESIGN MODULE NAMES

The following convention for module naming has been adopted to avoid name conflicts when compiling using any of the standard EDA tools that are available.

The name for each block of circuitry within the soft core is unique and is (generally) of the form:

m3s*nnnaa*

where *nnn* is the module number and *aa* is a unique alpha code for the design. The MCAN2 has the alpha code **fg**, and thus its modules have names of the form **m3s***nnn***fg**.

These module names are not intended to convey any meaning, but they can be cross-referenced to a functional description via the hierarchy diagram shown in Section 1.5 of this User Guide (and in the Product Specification).

3.3.3. SIGNAL NAMING

Signal names start with alpha characters and only contain alphanumeric tokens.

Signals that are active low are usually prefixed with a ‘N’ – for example NRST.

4. RECOMMENDED DESIGN FLOW

This chapter describes the way in which we suggest you integrate the MCAN2 (or indeed, any soft core) into your design flow.

The philosophy used in generating any Inventra™ design is to provide material that is 'open' so that it can be used with different EDA tools with the minimum of effort. To this end, each Inventra™ design is normally delivered in industry standard VHDL and Verilog formats.

Using an Inventra™ core in an ASIC design should not disrupt your design flow. Large chips are often designed by a team and integrating contributions from each team member is part of the overall design task. Using the Inventra™ core in your ASIC design should be just like using any other separately-designed block. However, it is important to remember that neither the core or the supplied test suites have been designed with any knowledge either of the technology in which you intend to fabricate your ASIC or of the environment in which the core will be used.

This chapter goes through the steps involved in a typical ASIC design flow and discusses how the supplied design material fits in with each part of the process. You will also be pointed to the part of this guide that is relevant to the different points in this flow.

Note: *The first thing to do with any Inventra™ soft core you plan to use is to check the integrity of the files you have extracted from the CD (or other source). The way to do this is first to run the supplied test suite (as described in Section 5). If the supplied test suite does not run correctly, check the list of files you extracted to ensure that all the required files have been extracted.*

4.1. STAGE 1: SYSTEM SPECIFICATION

Functional Information: Functional information about any core's design and interface requirements can be obtained from the core's Product Specification (supplied in its **docs** directory). You may also find it helpful to refer to the documentation that is available for the reference part for the design (which in this case is the Philips SJA1000 in its PeliCAN mode).

Design Partitioning: The area required for the core within your design will be easy to determine where you have already used the selected core in your chosen target technology.

If not, you can obtain an estimate of the area required by reading the netlist for the design (where supplied) into Synopsys Design Compiler along with the appropriate reference technology library, then using a **report_reference** command to generate a reference report (in a file).

The report generated will give a table of cells from which you will be able to deduce the approximate size of the core in your chosen technology by cross-referencing with the cell information given in your target ASIC library.

Design For Test: Before embarking on the overall system ASIC design, you should plan how the final design will be tested. This may involve ensuring that certain signals are controllable from the outside of the chip for scan chain insertion. For further information on this, see the section on Production Test (Section 7).

4.2. STAGE 2: DESIGN CAPTURE

Having specified and partitioned the system, the next step is to implement the design. You are likely to do this using a hierarchical schematic editor or a text editor of some kind.

Instantiating the Core in a schematic: The main task in this case is to create a schematic symbol for the supplied core. This will need to be done using the schematic capture tool's symbol editing package. The top-level diagram in the Product Specification may prove a useful reference here.

Once the symbol has been created, place it on the schematic and connect it up to the rest of the system. Some editors allow you to attach the RTL source for the core to the symbol so that the netlist contains all of the design. Others, however, will produce a netlist that treats the symbol as a standard component, leaving you to complete the description by adding the soft core source as a definition of another level of hierarchy in the design.

Instantiating the Core in a RTL source file: The simplest way to instantiate an Inventra™ core in a RTL source file is to cut and paste the top-level module declaration of the design, change the module definition to a component instantiation and connect the relevant signal definitions to the module. Once the core has been instantiated, compile the system with the soft core source to ensure that there are no unconnected pins or signal type mismatches.

Note: If you use explicit pin definitions, you will get another level of checking from your compiler.

4.3. STAGE 3: SYSTEM VERIFICATION

One of the most common sources of design errors is in the interconnections between blocks. It is therefore very important that you simulate the overall system, and that you run a simulation which tests your ASIC in the way in which you intend to use it in its target application.

The use of the simulation libraries and testbenches provided for the MCAN2 is described in the section on Simulation (Section 5).

Note: The fact that the Inventra™ design has been verified ***doesn't*** mean that you don't need to test it, because there may be a facet of the way in which you are using it in your system that causes problems. This may be as a result of a misunderstanding about exactly what it does, or it may be due to an interface issue.

4.4. STAGE 4: SYNTHESIS

In this stage of the process, the RTL is synthesized into a gate-level netlist for the target technology, according to the timing constraints specified.

This part of the flow is supported through ***example*** synthesis scripts and constraints files, which may be ***adapted*** to suit the target technology and the application requirements. For further information on this topic, please refer to the section on Synthesis (Section 6).

4.5. STAGE 5: GATE-LEVEL VERIFICATION

After synthesis, the gate-level implementation of the system ASIC needs to be verified using the test suite generated during the system verification stage.

4.6. STAGE 6: PRODUCTION TEST VECTOR DESIGN

The final stage in the flow is the development of the production test vectors. More information on this is given in the section on Production Test (Section 7).

5. SIMULATION

This chapter outlines how to simulate the MCAN2 core using the supplied simulation libraries and testbenches.

The purpose of functional simulation is to ensure that the behavior of the design under test conforms to the requirements of the project. Provided with the MCAN2 are a range of script files and testbenches for use in simulating the MCAN2 design. The primary purpose of these files is to allow the user to confirm that the supplied tests run on the supplied source files. It may also be possible to adapt the provided material to provide part of your system verification suite.

What these files offer and the way in which they are intended to be used is explained below.

Note: The supplied files have all been set up to run within the simulation environment used at Inventra, so before running these tests, the script files need to be modified to reflect the user's environment. You should also note they do not test every conceivable mode of operation for a particular design and the user should ensure that the mode(s) of operation required by their system configuration are tested.

For version-specific information on simulating the MCAN2, please refer to the ***sim.readme*** *file in the* ***mcan2/<language>/sim*** *directory.*

5.1. FILES PROVIDED

The testbenches, scripts and associated files that are provided for simulating the MCAN2 are supplied in the **sim** directory within the **mcan2/<language>** directory. The main files provided within this directory are:

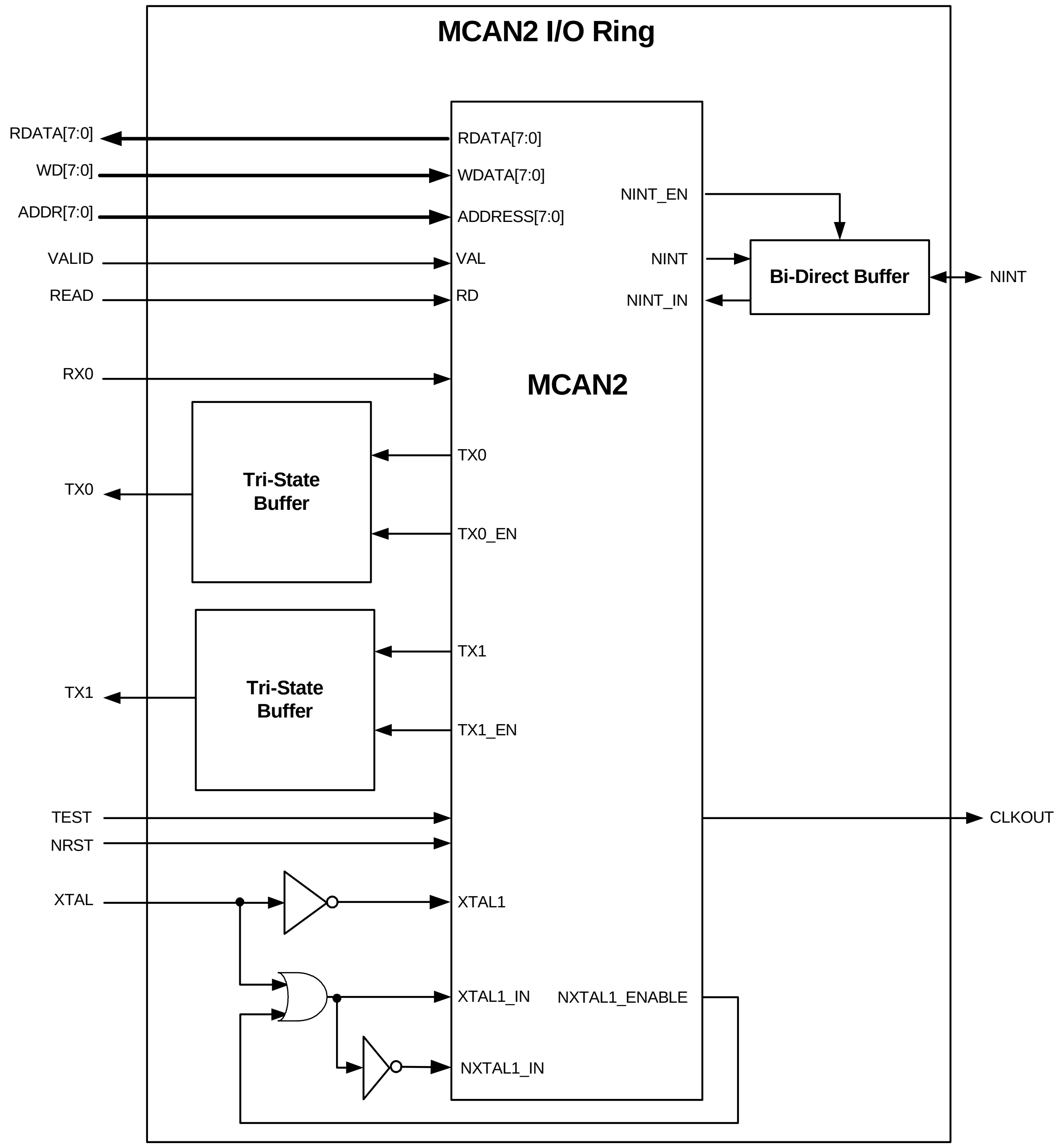
mcan2_io.v(hd)	I/O ring provided for functional verification of MCAN2 core (described in Section 5.2)
mcan2_tb.v(hd)	Main testbench
timescale.v(hd)	File in which the timescale for the simulation is set
msim_rtl.scr	GUI script for running individual RTL tests on the main testbench under ModelSim
msim_rtl_c.scr	Command line version of the above
msim_rtl_run_all.scr	Shell script for running all the RTL tests under ModelSim
msim_gate.scr	GUI script for running individual gate-level tests on the main testbench under ModelSim
msim_gate_c.scr	Command line version of the above
msim_gate_run_all.scr	Shell script for running all the gate-level tests under ModelSim
vxl_rtl_c.scr	Shell script for running individual RTL tests on the main testbench under Verilog XL (Verilog only)
vxl_rtl_run_all.scr	Shell script for running all the RTL tests under Verilog XL (Verilog only)
vxl_gate_c.scr	Shell script for running individual gate-level tests on the main testbench under Verilog XL (Verilog only)
vxl_gate_run_all.scr	Shell script for running all the gate-level tests under Verilog XL (Verilog only)
rtlcompile_tbs.scr	Compile script used for RTL simulation under ModelSim
gatecompile_tbs.scr	Compile script used for gate-level simulation under ModelSim
tb.list	Compile script used for RTL simulation under Verilog XL (Verilog only)
tbg.list	Compile script used for gate-level simulation under Verilog XL (Verilog only)
tb_files.list	List of testbenches used by the 'run_all' versions of the scripts
modelsim.ini	Example modelsim.ini file
wave.do	Waveform window setup file for ModelSim, main testbench

Note: The code for the different tests is supplied in an **include_files** subdirectory. This subdirectory also includes a **can_def.v** file that sets the register addresses and defines the CAN states used by the testbench. This file should only be edited to set the appropriate cycle and strobe values if you wish to run the simulation at a different clock rate to that supplied (currently 50MHz).

5.2. TEST ENVIRONMENT

The following diagram shows the MCAN2 core instantiated within the I/O ring supplied for functional verification of the core.

Note that tri-state buffers are provided for TX0 and TX1 so that these outputs will float at reset.



5.3. RUNNING THE SIMULATION

Scripts called **msim_rtl.scr** / **msim_rtl_c.scr** are provided in the **mcan2**/<language>/**sim** directory for running the **mcan2_tb.v** testbench on the MCAN2 RTL code under ModelSim, together with an **rtlcompile_tbs.scr** which is used to compile the RTL and the testbench. Similar **msim_gate** scripts supplied are for use together with a **gatecompile_tbs.scr** in running the supplied testbench on the gate-level code, while equivalent **vxl_rtl_c.scr** and **vxl_gate_c.scr** scripts are provided together with **tb.list** and **tbg.list** files for running RTL and gate-level simulations under Verilog XL.

A wide range of tests are provided (listed in the **sim.readme** file). These tests can be run individually using commands of the form:

```
<script_name>  <test_name>
```

Alternatively the full range of tests can be run using the **msim_rtl_run_all.scr** / **msim_gate_run_all.scr** scripts (or the equivalent Verilog XL scripts, **vxl_rtl_run_all.scr** and **vxl_gate_run_all.scr**). *Note:* These scripts all call on the file **tb_files.list** to provide the list of tests to be run. It is therefore possible to run a more selective list of tests simply by editing the **tb_files.list** file.

Simulation using any of these scripts generates a ***.lis** file containing the signal listing. These listing files are compared against reference files stored in the **tbfiles** directory using the UNIX **diff** command. Any differences are saved to a **./dif** (**./gdif** in gate-level case) directory. Any files in the **./dif** and **./gdif** directories should be of zero length.

Further information about the tests is given in the **sim.readme** file in the **mcan2/<language>/sim** directory.

Important: The **modelsim.ini** file included in the **sim** directory shows the simulation environment used within Mentor Graphics and is provided solely as an *example*. Do not use this file without first modifying it for your simulation environment.

5.4. BACK-ANNOTATION OF TIMING INFORMATION

Most libraries include an intrinsic delay for each macrocell in the library, for use in the absence of back-annotation. These give the macrocells fixed delays, which assume that the cell outputs have no loads. In order to get a more realistic idea of timing, you need to use back-annotation.

Timing back-annotation is supported by reading a Standard Delay Format (SDF) file into the simulation database at compile time. (Most simulators have a switch through which SDF data may be loaded.)

SDF files are generated by various synthesis and layout tools. The SDF files supplied with Inventra's pin-compatible cores have been generated by the Synopsys Design Compiler **write_timing** or **write_sdf** command (with a little post-processing). A suitable SDF file can be output from Design Compiler for any Inventra™ design when it is synthesized using the supplied technology library.

Again, further information is given in the **sim.readme** file in the **mcan2/<language>/sim** directory.

Note: SDF V2.1 files are recommended for back-annotation in order to ensure SDF data is matched to model parameters.

6. SYNTHESIS

Each pin-compatible core in the Inventra™ Soft Core Library includes compilation scripts and constraints files for use with Synopsys Design Compiler for synthesizing RTL source in both Verilog and VHDL.

This section discusses the process of synthesizing the MCAN2 from the deliverable material using the supplied synthesis scripts. The supported synthesis tool is Synopsys Design Compiler. The script **synth.scr** is a Synopsys dc_shell script that will synthesize the entire design.

\For version-specific information on synthesizing the MCAN2 design, please refer to the ***synth.readme*** *file in the* ***mcan2/<language>/ synth/synop*** *directory.*

6.1. THE FILES PROVIDED

The following files are provided in the **mcan2/<language>/synth/synop** directory for use with Synopsys Design Compiler for synthesizing supplied RTL source.

Synthesis principally makes use of the following scripts:

synth.scr	Main synthesis script for the design (Unix C shell script)
hier_setup.scr	Timing and process constraint file (Synopsys dc_shell script)
hier_compile.scr	Main compilation script (Synopsys dc_shell script)

Note: The supplied scripts and constraints files are all designed for synthesizing the MCAN2 design in its reference technology. You will need to develop your own files for synthesizing the core in your chosen technology, either from scratch or by modifying the supplied files (see Section 6.2 below). The supplied files should be treated as no more than an illustration of the sequence of steps and the sorts of constraints that might be needed.

6.2. CONFIGURING THE FILES TO THE TECHNOLOGY USED

The supplied scripts have been written to reference one particular technology library. Synthesis of the design in some other technology will therefore require a different set-up file either written from scratch or produced by modifying the supplied scripts to suit the technology you are using and the constraints imposed by your design requirements.

There are a number of fields in this file that should be changed to make best use of the target technology and you should seek guidance on the most appropriate entries from your ASIC vendor's technical data or application engineering staff. You will at least need to change the **library** variables to call up appropriate ASIC vendor technology files.

Even if you are intending to use the technology the script is currently set up for, you will need to edit the supplied **.synopsys_dc.setup** file to make various variables reflect your system environment.

6.2.1. SET-UP FILES

Synopsys Design Compiler uses a hierarchy of set-up files in order to set the many switches used in compilation. This usually includes a system set-up file called **.synopsys_dc.setup** which is used to ensure that variables are set consistently on a global basis, but which can be overridden by a **.synopsys_dc.setup** file included in the directory from which you invoke Synopsys.

The **.synopsys_dc.setup** file supplied with any Inventra™ core is configured for synthesizing the design in the core's reference technology. Synthesis of the design in some other technology will therefore require a different set-up file either written from scratch or produced by modifying the supplied set-up file, at least to change the **link_library** and **target_library** variables to call up appropriate ASIC vendor technology files.

Other variables concern Verilog and VHDL file input and output. Again, you may have other requirements according to your system environment. You will probably also want to change the designer and company variables to your own preference.

Before using the supplied set-up file, you will in any case need to edit it to make various variables reflect your system environment.

6.2.2. CONSTRAINTS FILES

The main constraints used in synthesizing the MCAN2 design are contained in the **hier_setup.scr** file supplied with the design.

A number of synthesis parameters may need be set or changed in the **hier_setup.scr** file in order to make best use of your chosen target technology. Further information about some of these parameters is given below.

You should seek guidance on the most appropriate entries from your ASIC vendor's technical data or application engineering staff.

6.2.2.1. SET_OPERATING_CONDITIONS

This variable sets the process/temperature/voltage corner at which all delays in the design are timed. The way the process corner is specified can vary and is likely to be different for each vendor.

Note: In some cases, this variable is set in the technology library itself and so determines the library file that is used.

6.2.2.2. SET_WIRE_LOAD

This variable selects the wire load model used when calculating the wire delays during static timing analysis. This is usually a function of the area of the design and each ASIC technology library typically offers a choice of wire load models from which to select the one most appropriate to the design you are synthesizing.

6.2.2.3. SET_MAX_TRANSITION

This variable imposes the constraint upon the design that no output signals should have a longer transition time than the stated time. This is in effect constraining the maximum propagation delay through every macrocell in the design.

Note: In some cases, this variable is set in the technology library itself and so determines the library file that is used.

6.2.2.4. SET_DRIVE

This variable sets up the default drive strength of the signals driving into the design and is used by the static timing analyzer. In the reference technology libraries used for Inventra™ designs, it is set by default to be the equivalent of the drive of the standard inverter cell.

Note that this variable can be overridden for particular ports, if required.

6.2.2.5. SET_LOAD

This variable sets the default load for all gate outputs in the design, and is used in static timing analysis. In the reference technology libraries used for Inventra™ designs, this has been set to be the equivalent of ten times the input load of the standard inverter.

6.2.2.6. OTHER CONSTRAINTS

The other constraints in the **hier_setup.scr** file relate to signal interface timing and are design specific. They are usually based on timings from a target device datasheet and are intended to be used as a reference.

In particular, clock attributes are defined on signals that are used to clock sequences of flip-flops. These can include read and write strobes, in which case the clock is defined as being the highest speed of read/write access that is supported.

All data inputs to Inventra™ cores are specified as being stable for several nanoseconds prior to a clock edge and all outputs from the design are required to have stabilized several nanoseconds before the next clock edge. These are not timing constraints as such, but are time-budgeting parameters to allow the static timing analysis to take account of the delay environment surrounding the core. In your design, you may wish to replace the supplied constraints with your own derived timing, or use the **characterize** command at a higher level of your design hierarchy.

6.3. RUNNING THE SYNTHESIS SCRIPT

The synthesis of the MCAN2 is driven from the **synth.scr** script provided for the design. This invokes Synopsys dc-shell, and calls up **hier_compile.scr** as an include file. **hier_compile.scr** in turn calls up **hier_setup.scr** which contains the constraints applied.

The **hier_compile.scr** script causes Synopsys dc_shell to read in the RTL source file, analyze and elaborate it. The timing and load constraints outlined in the **hier_setup.scr** file are then applied and the design compiled according to the directions given in the **hier_compile.scr** file.

The output from the synthesis is placed in the **gates/synop** directory and comprises:

mcan2.v(hd)	-	Netlist in the target technology
mcan2.sdf	-	SDF back-annotation timing file

The **hier_compile.scr** script can be edited to change the names of these files if required.

Further information about the supplied scripts is given in the **synth.readme** file in the **mcan2/<language>/synth/synop** directory.

Note: The files supplied are intended only as an example that gives a useful starting point. For synthesis in the desired technology, the user should modify the contents of these files as described in Section 6.2 above.

6.4. REPORT FILES

The **hier_compile.scr** script specifies the creation of the following Design Compiler report files in the **synth/synop** directory:

constraints.rep	-	Verbose account of the constraints applied.
mcan2_cnt.rep	-	Area/Gate count report.
timing.rep	-	Basic timing paths.
violation_all.rep	-	Constraint violations report.
violation_verb.rep	-	Verbose version of the above report.

Other reports can be generated by modifying the supplied script.

7. PRODUCTION TEST

The MCAN2 is designed to be scan test ready and scripts are provided for scan insertion using either Synopsys Test Compiler or Mentor Graphics DFTAdvisor, with additional scripts provided for ATPG using FastScan.

The files for use with Synopsys' Test Compiler are provided in the **mcan2/<language>/synth/synop** directory and comprise the Unix C shell script **scan.scr** and the dc_shell script **hier_scan.scr. scan.scr** invokes Synopsys Test Compiler and runs the script **hier_scan.scr** which inserts scan test into the design. Note that, before using these scripts, you will need to set the target library in the **.synopsys_dc.setup** file and to modify **hier_scan.scr** to adapt it to your directory structure.

The files for use with Mentor Graphics DFTAdvisor and FastScan are provided in the **mcan2/<language>/synth/mgc/dft** directory and comprise:

- **dft_ad.scr** which invokes DFTAdvisor and runs the example Mentor scan insertion command script **dft_ad.do** which inserts scan test into the design
- **fscan.scr** which invokes FastScan and runs the example test pattern generation script **fscan.do**.

(Again, before using these scripts, you will need to modify these files to reflect your chosen target library and your DFT and FastScan directory structures.)

Note: After scan chains have been inserted, the design should be re-compiled incrementally to ensure that it conforms to the design constraints and a functional simulation should be run to verify that no circuit errors have been introduced as a result of the scan insertion process.

8. REVISION HISTORY

8.1. ISSUE 1

16th October 2001. First issue of document.

8.2. ISSUE 2

7th May 2006. Addition made for NXTAL1_IN port (Section 5.2).

MCAN2 USER GUIDE

INVENTRA™

http://www.mentor.com/inventra

Corporate Headquarters
Mentor Graphics Corporation
8005 S.W. Boeckman Road
Wilsonville, OR 97070 USA
Phone: 503-685-7000
North American Support Center
Phone: 800-547-4303
Fax: 800-684-1795

Silicon Valley Headquarters
Mentor Graphics Corporation
1001 Ridder Park Drive
San Jose, California 95131 USA
Phone: 408-436-1500
Fax: 408-436-1501

Europe Headquarters
Mentor Graphics Corporation
Immeuble le Pasteur
13/15, rue Jeanne Braconnier
92360 Meudon La Foret
France
Phone: 33-1-40-94-74-74
Fax: 33-1-46-01-91-73

Pacific Rim Headquarters
Mentor Graphics (Taiwan)
Room 1603, 16F,
International Trade Building
No.333, Section 1, Keelung Road
Taipei, Taiwan, ROC
Phone: 886-2-87252000
Fax: 886-2-27576027

Japan Headquarters
Mentor Graphics Japan Co., Ltd.
Gotenyama Hills
7-35, Kita-Shinagawa 4-chome
Shinagawa-Ku, Tokyo 140
Japan
Phone: 81-3-5488-3033
Fax: 81-3-5488-3021

05/06 PU-40107.002-FC