HurriCANe v5.2.4

HurriCANe v5.2.4 User's Manual

Prepared by:	W. Errico	Date:	03/11/2011
Verified by:	W. Errico		03/11/2011
Approved by:	F. Bigongiari (Project Manager)		03/11/2011

ESA DEVELOPMENT CONTRACT REPORT

ESA CONTRACT No	SUBJECT		CONTRACTOR
19943/06/NL/CO	HurriCANe v5.2.4 User's Manual		Aurelia Microelettronica S.p.A. Via Vetraia, 11 55049 Viareggio (LU) ITALY Tel.: +39 0584 388398 Fax: +39 0584 388959
* ESA CR ( ) No	* STAR CODE	No of volumes 1 This is volume No 1	CONTRACTOR'S REFERENCE AUR/HURRICANE/UMD/4

ABSTRACT

This document describes the content of the HurriCANe CAN distribution.

Names of authors

Fabrizio Bertuccelli, Walter Errico

**NAME OF ESA STUDY MANAGER	** ESA BUDGET HEADING
Francisco Tortosa Lopez DIV: DIRECTORATE:	

Distribution List

Document title: HurriCANe v5.2.4 User's Manual		
Distribution	**Type**	**Copies**
Francisco Tortosa Lopez (ESTEC)	E	1*
Agustin Fernandez Leon	E	1*
SITAEL AEROSPACE Document Center	I	1*+1
I = Internal E = External * = Electronic Copy		

Change Document Record

Document title: HurriCANe v5.2.4 User's Manual				
Issue	**Date**	**Total pages**	**Modified pages**	**Notes**
1	22/11/2006	13	None	First Issue
2	10/04/2007	22	13-22 (Added)	Second Issue
3	07/09/2007	24	All	Third Issue
4	03/11/2011	25	1	Changed name in prepared by field
			8	Updated paragraph §1.2
			9	Updated Figure 2
			10	Updated paragraph §1.3 and §1.4
			11	Added paragraph §1.5.1
			12	Upgraded table 2
			12	Added paragraph §1.5.3
			17 - 19	Updated paragraph §2.3 (removed all test = '0' sentences)
			20	Modified figure 7 (removed can_signal)
			21 - 22	Updated appendix A
			23	Updated appendix B

List of acronyms and abbreviations

AMBA	Advanced Microcontroller Bus Architecture
CAN	Controller Area Network
VHDL	Very High Speed Integrated Circuit Hardware Description Language
FPGA	Field Programmable Gate Array
IP	Intellectual Property
LFSR	Linear Feedback Shift Register
RTR	Remote Transmission Request
EOF	End Of Frame

Table of Contents

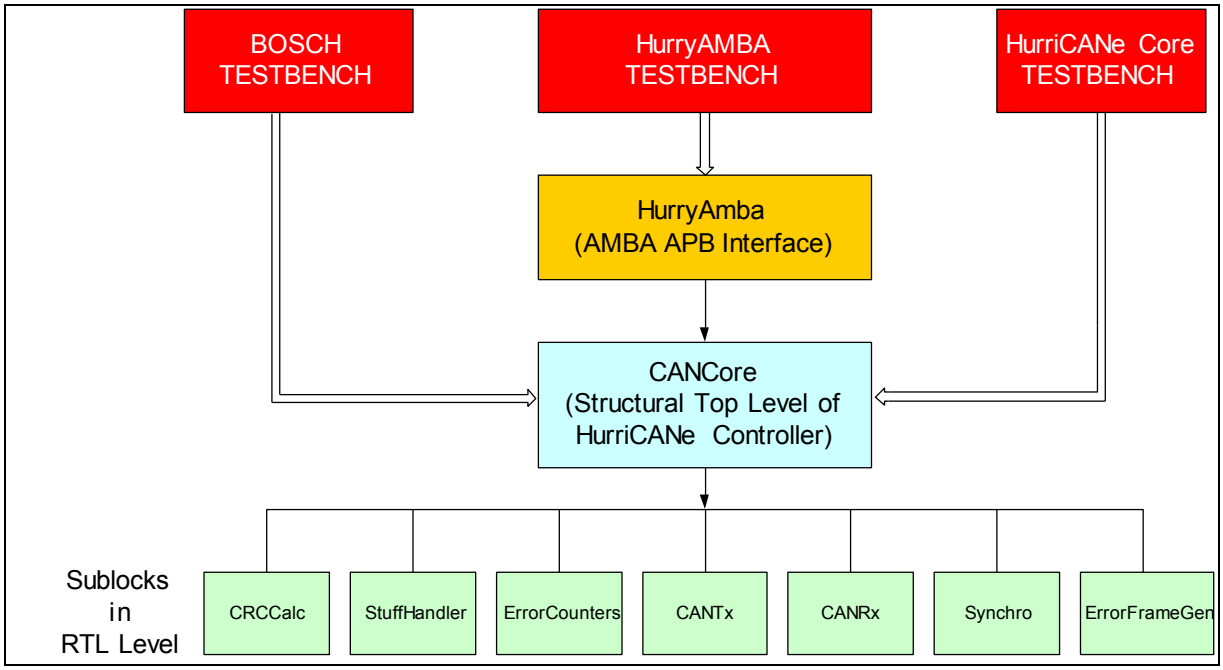
1. DISTRIBUTION DESCRIPTION

1.1. Introduction

The HurriCANe distribution is intended for designers who want to build their own implementation of a CAN controller starting from a validated CAN core, or to use a full implementation of a CAN controller inside their own projects.

This core implements a CAN Bus controller node according to the ISO 11898-1:2003 standard and to the BOSCH CAN Specification 2.0B.

This core has been verified against the BOSCH "VHDL Reference CAN Model" testbench, demonstrating full compliance to the BOSCH CAN Specification.



The distribution package includes test benches supporting the following features:

- CAN Protocol Version 2.0 Part A, B
- Flexible test-bench environment
- Simulate entire CAN bus system (number of nodes defined by user)
- Test program set can be extended by user

The following files are provided to assist the user in working with this distribution and in understanding its functionality:

- User Manual (this document)
- Functional Specification
- Verification Specification
- Well documented CAN source code

- Test benches
- Simulation examples
- Synthesis examples

Note: The user of this distribution is expected to be familiar with the CAN Specification Revision 2.0 Part A and Part B.

1.2. Installation

To install the HurriCANe distribution please proceed in the following way:

1. *Create a directory where you want to install the distribution by typing*:
   mkdir <desired path>/HurriCANe
   Example: **mkdir** /projects/HurriCANe

2. *Copy the HurriCANe.tar file to this directory.*
   cp HurriCANe_5.2.4.zip <desired path>/HurriCANe
   Example:**cp** HurriCANe._5.2.4.zip /projects/HurriCANe

3. *Change your working directory to the distribution directory:*
   cd <desired path>/HurriCANe
   Example:**cd** /projects/HurriCANe

4. *Untar the distribution:*
   unzip HurriCANe_5.2.4.zip

Please check also that your VHDL simulator is set up correctly before proceeding.
In the following paragraph you can find a list of the files and directories together with a short description.
Simulations were carried out using the following simulators:

- CADENCE ncsim simulator, version 05.10-s010.
- Mentor Graphics ModelSim version 6.3d

1.3. Directory Description

Figure 1 indicates how the directory tree should look like after the **unzip** command performed at point 4 of the previous paragraph.

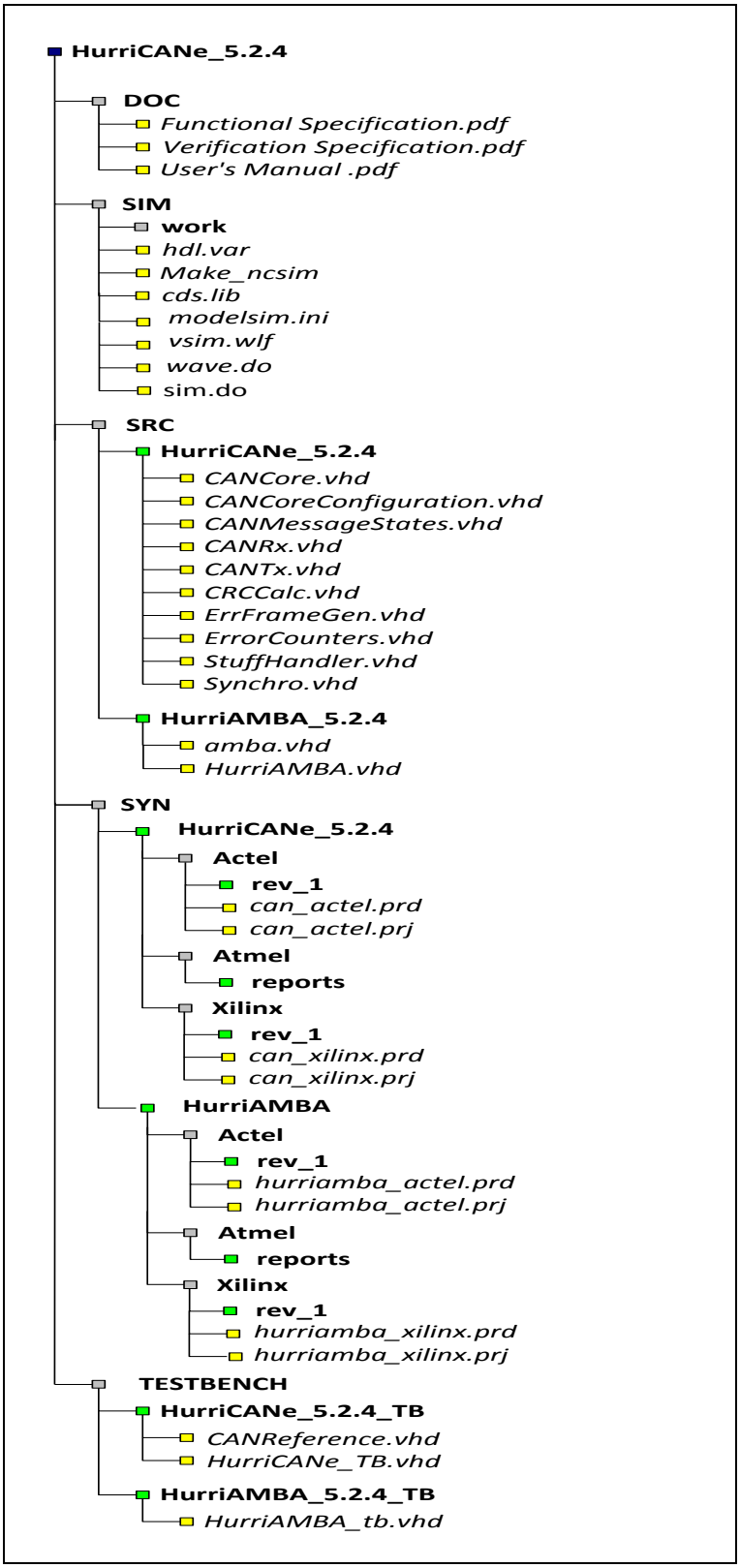


Figure 2: HurriCANe Directories and files

The directory names are represented with **bold** letters, files with *italics*.

All the source code is located under the **SRC** directory:
- HurriCANe_5.2.4 subdirectory
  VHDL source files for the HurriCANe CAN core v.5.2.4
- HurriAMBA_5.2.4 subdirectory
  VHDL source files for a complete CAN controller with AMBA interface
  The HurriAMBA.vhd file contains the HurriCANe core entity together with an implementation of the AMBA interface used to control the core.

The VHDL files for the validation of the CAN core and of the complete CAN Controller are located under the TESTBENCH directory: the HurryAMBA_TB.vhd can be used to check the behavior of the HurryAMBA CAN Controller while the compliance of the core to the BOSCH specification can be evaluated running the HurryCANe_TB.vhd file.

Finally the **DOC** directory contains all the documents that are released with this distribution:
- **Functional Specification.pdf**
  Describe the CAN core architecture and its fundamental blocks as well as the complete CAN controller with the AMBA interface.
- **Verification Specification.pdf**
  Analyze the structure of the HurriCANe Core and of the HurryAMBA interface testbenches and provides information to let the user add additional checks to the predefined set of test.
- **User's Manual.pdf**
  The current document.

1.4. Compilation and Simulation

If the NCSIM Simulator is installed on your machine, you can proceed with the following commands:

5. **cd** <desired path>/HurriCANe/SIM
   Example: **cd** /projects/HurriCANe/SIM

The make files in the SIM directory (*Makefile* and *make_ncsim* - see Appendix A for more details) allow the user to simulate the HurriCANe core and the HurryAMBA controller in a fast and easy way, using either the Mentor Graphics ModelSim or the Cadence NCSim simulators.

- ***Using the Mentor Graphics ModelSim simulator:***

  To compile, elaborate and simulate the HurriCANe core using ModelSim, type:

  6. **make** libs
  7. **make** HurriCANe

  To compile, elaborate and simulate the HurryAMBA CAN controller using ModelSim, type:

  8. **make** HurriAMBA

  The compiled files of the desired module can be found in the **work** directory.

  Finally to clear the simulation environment and to restart simulating from scratch, type:

  9. **make** clean

The content of the **work** directory will be deleted and the initial configuration restored.

- ***Using the Cadence NCSim simulator:***

  To compile, elaborate and simulate the HurriCANe core using NCSim, type:

  6. **make** –f make_ncsim HurriCANe

  To compile, elaborate and simulate the HurryAMBA CAN controller using NCSim, type:

  7. **make** –f make_ncsim HurriAMBA

  The compiled files of the desired module can be found in the **work** directory.

  Finally to clear the simulation environment and to restart simulating from scratch, type:

  8. **make** clean

  The content of the **work** directory will be deleted and the initial configuration restored.

1.5. Synthesis

Examples of various synthesis runs of the HurriCANe core, targeting FPGA (Xilinx and Actel) and ASIC (Atmel) technologies, can be found in the SYN directory. The sythesis runs targeting the FPGAs were carried out using Synplicity Synplify Pro v9.0.1, while the synthesis for the Atmel ASIC process was performed using Synopsys Design Compiler v2007.03-SP5.

1.5.1. *HurriCANe synthesis for Xilinx FPGA*

A project setup file for synthesizing the HurriCANe for a Xilinx Virtex-2 FPGA, using Synplicity Synplify Pro, can be found in the SYN/xilinx directory. To start the synthesis run, open the **can_xilinx.prj** file from within Synplify, and simply press the RUN button in the Synplify GUI.

The synthesis results for the HurriCANe core, targeting a Xilinx XC2V250FG256-6 FPGA (without I/O insertion), are given in Table 1 below.

Table 1 : Resource usage report for Xilinx XC2V250FG256-6

Cell's Type	Occupacy
LUTs	1047 (34%)
Registers (FFs)	715 (23%)
Clock Buffers	1
Max clock frequency	88 MHz

1.5.2. *HurriCANe synthesis for Actel FPGA*

A project setup file for synthesizing the HurriCANe for an Actel RTSX FPGA, using Synplicity Synplify Pro, can be found in the SYN/actel directory. To start the synthesis run, open the **can_actel.prj** file from within Synplify, and simply press the RUN button in the Synplify GUI.

The synthesis results for the HurriCANe core, targeting an Actel RT54SX72S FPGA, are given in Table 2 below.

Table 2 : Resource usage report for ACTEL A54SX72A

Cell's Type	Occupacy
Combinational Cells	1217 of 4024 (30%)
Sequential Cells	530 of 2012 (26%)
Total Cells	1747 of 6036 (31%)
Clock Buffers	2
IO Cells	247
Max clock frequency	10.4 MHz

1.5.3. *HurriCANe synthesis for Atmel ASIC*

Indicative synthesis results for the ATMEL ATC18RHA 180nm rad-hard ASIC process are given in Table 3 below. The results were obtained using Synopsys Design Compiler v2007.03-SP5. Detailed reports from this synthesis run can be found in the SYN/atmel/reports directory.

Table 3 : Resource usage report for ATMEL ATC18RHA

Cell's Type	Occupacy
Combinational area	2810 gates (NAND2)
Non-combinational area	9680 gates (NAND2)
Total area	12490 gates (NAND2)
Max clock frequency	50 MHz

(*Note*: Area of NAND2 gate = 12,544 μm^2)

2. HURRICANE CORE OPERATIONAL DESCRIPTION

2.1. Introduction

The aim of this chapter is to provide the user with the necessary information to setup the HurriCANe module in order to perform the most common CAN operations (DATA transmission, DATA reception, REMOTE FRAME transmission, REMOTE FRAME reception, etc.).

2.2. HurriCANe signal interface description

A list of all the HurriCANe inputs and outputs is reported in Table 4.

Signal	Direction	Polarity	Function	
clock	IN	-	System Clock	
reset	IN	AH	System Reset	
ps1[3:0]	IN	-	Phase Segment 1 value	These inputs are used to control the TX/RX data baud rate. Refer to section "Baud rate settings" for details on how to program the baud rate.
ps2[3:0]	IN	-	Phase segment 2 value	
bpr[1:0]	IN	-	Baud Rate Prescaler	
rsj[2:0]	IN	-	Resynchronization jump width	
filter_remote	IN	AH	Message Filtering result of Remote Frame transmission 1: Remote Frame received match the Rx identifier 0: Remote Frame Received doesn't match the Rx identifier	
remote_frm	IN	AH	Reply to Remote Frame Transmission	
tx_msg[0:101]	IN	-	Message Frame to be transmitted	
start	IN	AH	Start Transmission	
bit_stream	IN	-	Input stream from CAN Bus	
syncbit	OUT	AH	Synchronization Segment	
sampled_bit	OUT	-	Received bit	
rx_msg[0:101]	OUT	-	Received Message Frame	
rx_completed	OUT	AH	Reception Completed Flag When reception is completed a 1 *nominal bit time* active pulse is generated on this pin	
tx_bit	OUT	-	Output Stream to CAN Bus	
tx_busy	OUT	AH	Transmission in Progress 1: Data Transmission is on 0: Data Transmission is off	
tx_completed	OUT	AH	Transmission Completed Flag When transmission is completed a 1 *nominal bit time* active pulse is generated on this pin	
rx_err_cnt[0:7]	OUT	-	Rx Error Counter Value	
tx_err_cnt[0:7]	OUT	-	Tx Error Counter Value	
Err_passive	OUT	AH	Error Passive Flag 1: CAN node is error passive condition 0: CAN node is not in error passive condition	
bus_off	OUT	AH	Bus Off Flag 1: CAN node is in bus off condition 0: CAN node is not in bus off condition	

Table 4 : HurriCANe Input-Output pin list.

▪ *Baud Rate Settings*

The **ps1[3:0]**, **ps2[3:0]**, **rsj[2:0]** and **bpr[1:0]** inputs control data transmission and data reception baud rate. In particular **bpr[1:0]** acts like a baud rate prescaler which divides the input system clock according to the following relationship:

$$F_{int\,clock} = \frac{F_{clock}}{2^{bpr}}$$

The **ps1[3:0]** input is the sum of the PROP_SEG plus the PHASE_SEG1 time segments and it can assume any value between 2 and 15, **ps2[3:0]** is related to the PHASE_SEG2 segment and it can be programmed in the range [1,8],
As a consequence of the above statements the bit rate of messages transmitted on the bus will be:

$$BitRate = \frac{F_{clock}}{2^{bpr} \cdot ((ps1+1)+(ps2+1))}$$

Table 1 gives some examples of how to program **ps1[3:0]**, **ps2[3:0]** and **bpr[1:0]** registers in order to obtain the most common bit rate from the most typical system clock (F_{clock}).

Table 5 : Example of ps1[3:0], ps2[3:0] and bpr[1:0] setting

Bit Rate (KHz)	F_{clock} (MHz)	ps1[3:0]	ps2[3:0]	bpr[1 :0]
1000	16	8	6	0
1000	16	9	5	0
1000	16	10	4	0
1000	20	11	7	0
1000	20	12	6	0
1000	40	13	5	1
500	16	8	6	1
250	16	8	6	2

Finally **rsj[2:0]** is used for the implementation of the resynchronization jump mechanism.

▪ *Data Transmission*

Data to be transmitted by HurriCANe must first be fed into the **tx_msg[0:101]** array beginning with the ARBITRATION field, and up to the DATA field, as shown in Figure 3. STANDARD or EXTENDED data frames are transmitted by the HurriCANe core depending on the bit value stored in the IDE field (IDEbit = 0 STANDARD FRAME selected; IDEbit = 1 EXTENDED FRAME selected).
The number of DATA bytes to be transmitted (contained in the DATA field) must be inserted into the DLC field at the appropriate position (bit 14-17 or bit 34-37 of **tx_msg[0:101]** depending on the type of message selected).

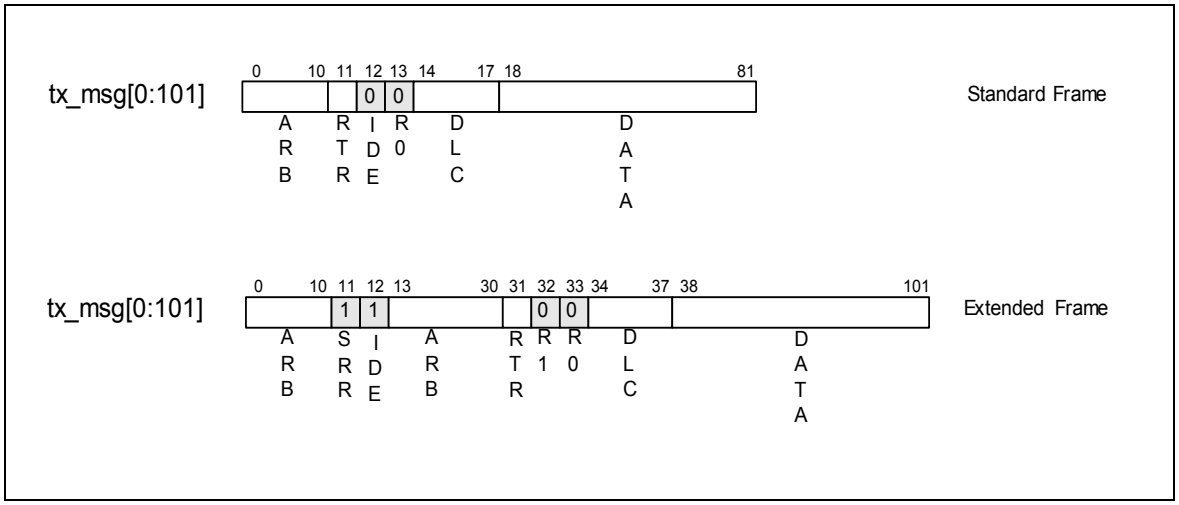


Figure 3: **`tx_msg[0:101]`** structure in the case of Standard and Extended frame transmission.

A pulse on the **`start`** pin initiates data transmission on the **`tx_bit`** output line. Transmission completion is signalled by a 1/BitRate width pulse generated on the **`tx_completed`** output pin.

`Start` must be held high until the **`tx_completed`** pulse has been detected.

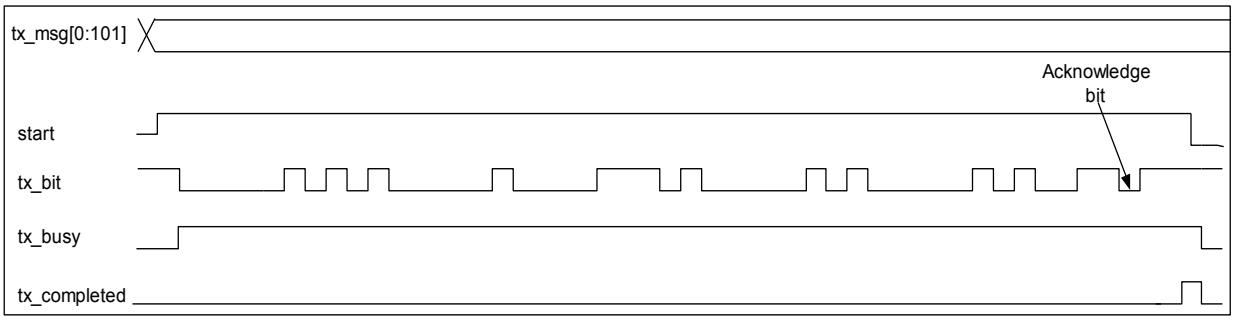


Figure 4: Transmission waveform diagram.

- *Data Reception*

CAN bus data are received on the **`bit_stream`** input. A 1-bitrate-long pulse is generated on the **`rx_completed`** pin to indicate a successful RX completion. All the data bits received are then output to the **`rx_msg[0:101]`** array (see Figure 5).

Data format representation in the **`rx_msg[0:101]`** follows the same structure as the **`tx_msg[0:101]`** array (see Figure 3).

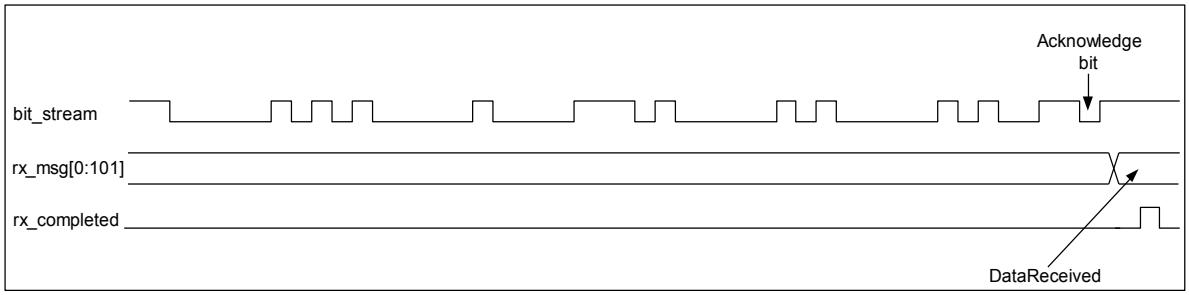


Figure 5: Reception waveform diagram.

- *Remote Frame Transmission*
  REMOTE FRAME transmission is achieved by setting the RTR bit to 1 (bit 11 of **tx_msg[0:101]** in case of STANDARD FRAME and bit 31 of the same array in case of EXTENDED FRAME) before toggling the **start** signal.

- *Reply to a Remote Frame Reception*
  DATA FRAME reply to a REMOTE FRAME reception is managed by using of the **remote_frm** and **filter_remote** signals.
  When the **remote_frm** and **tx_start** signals are both active the CAN core waits until a succesfull REMOTE FRAME reception has been completed (indicated by **rx_completed="1")** and the **filter_remote** pin has been set high before sending the DATA FRAME reply.

- *Error Management*
  Error management is achieved by using the **tx_err_cnt[0:7]**, **rx_err_cnt[0:7]**, **err_passive** and **bus_off** signals.
  The tx_err_cnt[0:7] and **rx_err_cnt[0:7]** counter registers represent the TRANSMIT ERROR COUNT and the RECEIVE ERROR COUNT registers used for CAN BUS fault confinement.
  Depending on the value of these registers, the CAN controller core can be in one of the following states (see Figure 6):

  - ERROR ACTIVE
  - ERROR PASSIVE
  - BUS OFF

  In the ERROR ACTIVE state the **err_passive** and **bus_off** flags are both held low.

  The ERROR PASSIVE state is entered when (a) either the TX error counter (**tx_err_cnt[0:7])** or the RX error counter (**rx_err_cnt[0:7]**) exceed the value of 127, and (b) the **err_passive** flag is set.

  The BUS OFF state is entered when **tx_err_cnt[0:7]** becomes greater than 255; when this condition is reached the **bus_off** signal is set high.

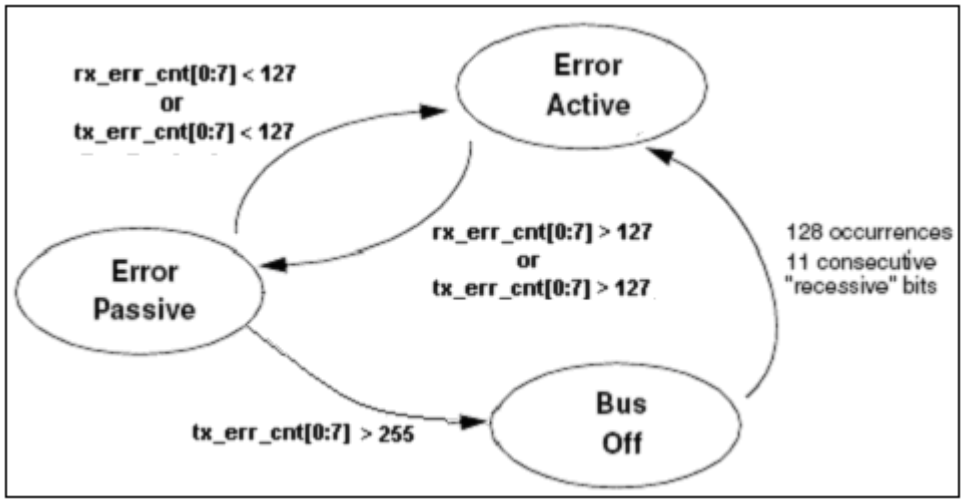


Figure 6: Error State flow diagram.

2.3. Configuration Examples

This section presents some examples for configuring the HurriCANe CAN controller to perform the most common CAN operations.

1. ***STANDARD DATA FRAME Transmission***

Parameters:
- F_{clock} : 16 MHz
- Bit Rate : 1 MHz
- Arbitration : 0x7AE
- Data length : 0x06 bytes
- Data : 0x10 0x11 0x12 0x13 0x14 0x15

The inputs to the CAN core must be set up as follows:

- **`ps1[3:0]`** : "1000"
- **`ps2[3:0]`** : "0110"
- **`bpr[1 :0]`** : "00"
- **`rsj[2 :0]`** : "010"
- **`filter_remote`** : '0'
- **`remote_frm`** : '0'
- **`tx_msg[0:101]`** : "11110101110" & '0' & '0' & '0' & "0110" & "00010000" & "00010001" & "00010010" & "00010011" & "00010100" & "00010101"

To perform data transmission the **`start`** signal must be held high until a pulse is detected on the **`tx_completed`** pin.

2. ***EXTENDED DATA FRAME Transmission***

Parameters:
- F_{clock} : 16 MHz

- Bit Rate : 1 MHz
- Arbitration : 0x31236AEF
- Data length : 0x08 bytes
- Data : 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17

The inputs to the CAN core must be set up as follows:

- **ps1[3:0]** : "1000"
- **ps2[3:0]** : "0110"
- **bpr[1 :0]** : "00"
- **rsj[2 :0]** : "010"
- **filter_remote** : '0'
- **remote_frm** : '0'
- **tx_msg[0:101]** : "11000100100" & '1' & '1' & "01101101011101111" & '0' & '0' & '0' & "1000" & "00010000" & "00010001" & "00010010" & "00010011" & "00010100" & "00010101" & "00010110" & "00010111"

To perform data transmission the **start** signal must be held high until a pulse is detected on the **tx_completed** pin.

3. *STANDARD REMOTE FRAME Transmission*

Parameters:
- F_{clock} : 16 MHz
- Bit Rate : 1 MHz
- Arbitration : 0x7AE
- Data length : 0x06 bytes

The inputs to the CAN core must be set up as follows:

- **ps1[3:0]** : "1000"
- **ps2[3:0]** : "0110"
- **bpr[1 :0]** : "00"
- **rsj[2 :0]** : "010"
- **filter_remote** : '0'
- **remote_frm** : '0'
- **tx_msg[0:101]** : "11110101110" & '**1**' & '0' & '0' & "0110" & "00010000" & "00010001" & "00010010" & "00010011" & "00010100" & "00010101"

Note the bit at 1 in the RTR position (bit 11 of **tx_msg[0:101]**); to perform REMOTE FRAME transmission the **start** signal must be held high until a pulse is detected on the **tx_completed** pin.

4. *EXTENDED REMOTE FRAME Transmission*

Parameters:
- F_{clock} : 16 MHz
- Bit Rate : 1 MHz
- Arbitration : 0x31236AEF
- Data length : 0x08 bytes

The inputs to the CAN core must be set up as follows:

- **ps1[3:0]** : "1000"
- **ps2[3:0]** : "0110"
- **bpr[1 :0]** : "00"
- **rsj[2 :0]** : "010"
- **filter_remote** : '0'
- **remote_frm** : '0'
- **tx_msg[0:101]** : "11000100100" & '1' & '1' & "01101101010111011111" & '**1**' & '0' & '0' & "1000" & "00010000" & "00010001" & "00010010" & "00010011" & "00010100" & "00010101" & "00010110" & "00010111"

Note the bit at 1 in the RTR position (bit 31 of **tx_msg[0:101]**); to perform REMOTE FRAME transmission the **start** signal must be held high until a pulse is detected on the **tx_completed** pin.

5. DATA FRAME Transmission as a reply to a REMOTE FRAME Request

Parameters:
- F_{clock} : 16 MHz
- Bit Rate : 1 MHz
- Arbitration : 0x31236AEF
- Data length : 0x07 bytes
- Data : 0x10 0x11 0x12 0x13 0x14 0x15 0x16

The inputs to the CAN core must be set up as follows:

- **ps1[3:0]** : "1000"
- **ps2[3:0]** : "0110"
- **bpr[1 :0]** : "00"
- **rsj[2 :0]** : "010"
- **filter_remote** : '0'
- **remote_frm** : '1'
- **tx_msg[0:101]** : "11000100100" & '1' & '1' & "01101101010111011111" & '0' & '0' & '0' & "0111" & "00010000" & "00010001" & "00010010" & "00010011" & "00010100" & "00010101" & "00010110"

To perform data transmission as a reply to a REMOTE FRAME reception, set the **start** signal high. When a REMOTE FRAME has been received that matches the Arbitration code, the **filter_remote** must be set high. Both the **start** and **filter_remote** signals must then be held high until a pulse is detected on **tx_completed** pin.

3. RESET AND CLOCK SIGNALS DISTRIBUTION

The distribution of the **`clock`** and **`reset`** nets inside the HurriCANe core is shown in Figure 7.

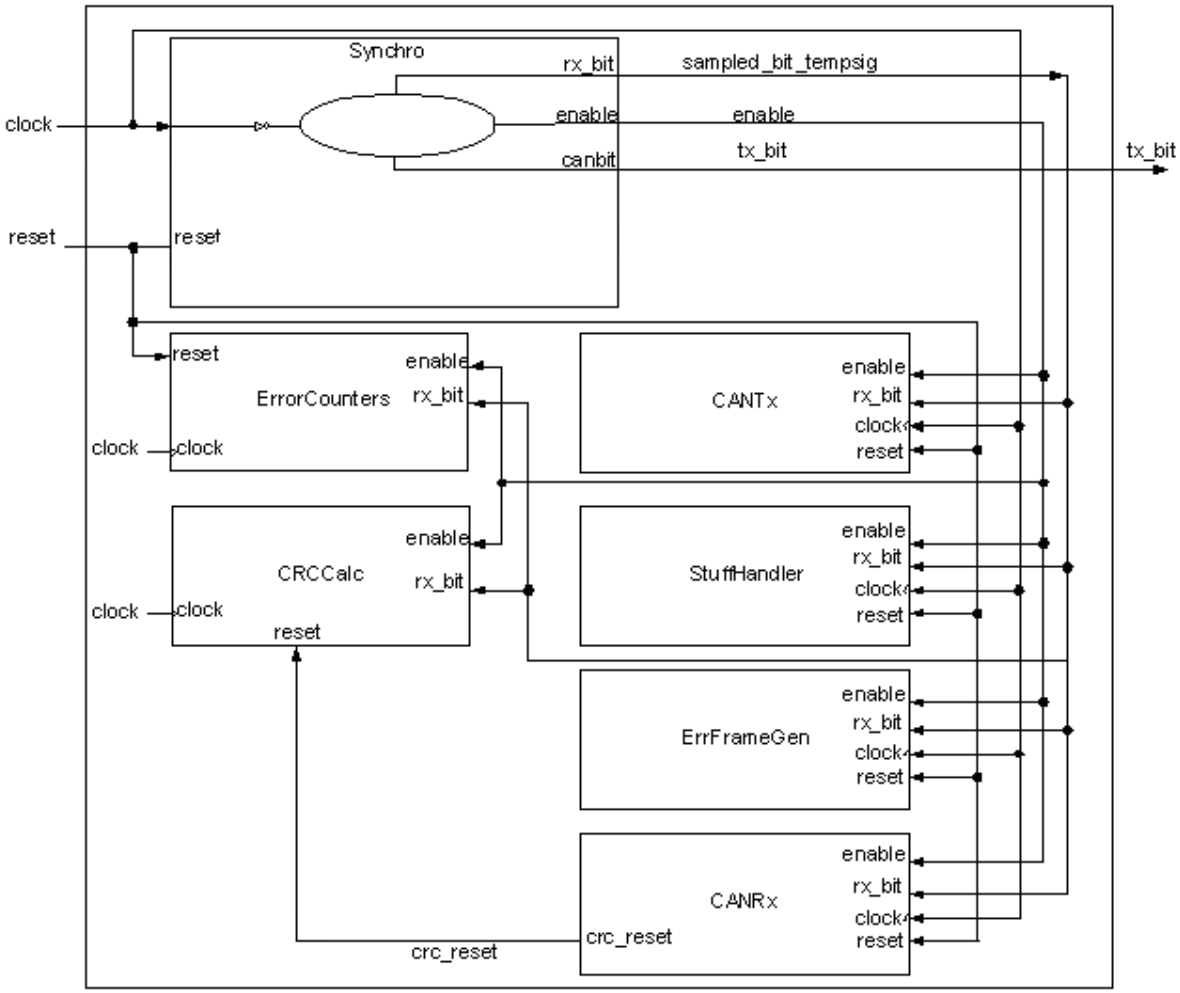


Figure 7: Reset and clock net distribution.

The **`clock`** net acts as the main system clock and it is distributed to all the different sub-system blocks of the HurriCANe core (Synchro, ErrorCounters, CRCCalc, CANTx, StuffHandler, ErrFrameGen, CAN Rx). Inside the Synchro block an invertion is performed on the **`clock`** signal and the inverted clock is used to generate the **`rx_bit`**, **`enable`** and **`canbit`** nets. The **`reset`** input is used to reset the Synchro and ErrorCounters blocks only. The CRCCalc block uses the **`crc_reset`** pin coming out from the CANRx module.

4. APPENDIX A : MAKEFILE

```
CORE_DIR       = ../SRC/HurriCANe_5.2.4/
CORETB_DIR     = ../TESTBENCH/HurriCANe_5.2.4_TB
INTRFACE_DIR   = ../SRC/HurriAMBA_5.2.4
INTRFACETB_DIR = ../TESTBENCH/HurriAMBA_5.2.4_TB
CORE_SRC       = CANMessageStates.vhdl\
            CANCoreConfiguration.vhdl\
            CANCore.vhdl\
                 CRCCalc.vhdl\
            StuffHandler.vhdl\
                 CANRx.vhdl\
            ErrFrameGen.vhdl\
            Synchro.vhdl\
                 CANTx.vhdl\
            ErrorCounters.vhdl
Hurry_SRC      = amba.vhdl\
            HurriAMBA.vhdl
CORETB_SRC     = CANReference.vhdl\
                 HurriCANe_TB.vhdl
INTRFACETB_SRC = HurriAMBA_tb.vhdl
VPATH          = $(CORE_DIR):$(CORETB_DIR):$(INTRFACE_DIR):$(INTRFACETB_DIR)

%.vhdl : %.vhd
   vcom -quiet -93 -work work $<

HurriCANe : $(CORE_SRC) $(CORETB_SRC)
   vsim -c -l vsim.log work.hurricane_tb\(tb_architecture\) -do "run -all"  |
   grep -e "--->"
   quit

HurriAMBA : $(CORE_SRC) $(Hurry_SRC) $(INTRFACETB_SRC)
   vsim -c -l vsim.log work.HurryAMBA_TB\(hurryAMBATB\) -do "run -all" |
   grep -e "# **"
   quit

libs:
   vlib work
   vmap work work

clean:
   \rm  -rf work/*
```

MAKE_NCSIM

```
CORE_DIR       = ../SRC/HurriCANe_5.2.4/
CORETB_DIR     = ../TESTBENCH/HurriCANe_5.2.4_TB
INTRFACE_DIR   = ../SRC/HurriAMBA_5.2.4
INTRFACETB_DIR = ../TESTBENCH/HurriAMBA_5.2.4_TB
CORE_SRC       = CANMessageStates.vhdl\
             CANCoreConfiguration.vhdl\
             CANCore.vhdl\
                  CRCCalc.vhdl\
             StuffHandler.vhdl\
                  CANRx.vhdl\
             ErrFrameGen.vhdl\
             Synchro.vhdl\
                  CANTx.vhdl\
             ErrorCounters.vhdl
Hurry_SRC      = amba.vhdl\
             HurriAMBA.vhdl
CORETB_SRC     = CANReference.vhdl\
               HurriCANe_TB.vhdl
INTRFACETB_SRC = HurriAMBA_tb.vhdl
VPATH          = $(CORE_DIR):$(CORETB_DIR):$(INTRFACE_DIR):$(INTRFACETB_DIR)

%.vhdl : %.vhd
     ncvhdl -V93 -WORK work $<


%.elab : %
    ncelab -ACCESS +rw -work work -cdslib ./cds.lib -logfile ncelab.log -
    errormax 15 -messages -status -v93 work.$<:tb_architecture


HurriCANe : $(CORE_SRC) $(CORETB_SRC)
    ncelab -ACCESS +rw -work work -cdslib ./cds.lib -logfile ncelab.log -
    errormax 15 -messages -status -v93 work.HurriCANe_TB:tb_architecture
    ncsim -cdslib ./cds.lib -logfile ncsim.log -errormax 15 -messages -status
    WORK.HURRICANE_TB:TB_ARCHITECTURE | grep  -e "--->"

HurriAMBA : $(CORE_SRC) $(Hurry_SRC) $(INTRFACETB_SRC)
    ncelab -ACCESS +rw -work work -cdslib ./cds.lib -logfile ncelab.log -
    errormax 15 -messages -status -v93 work.HurryAMBA_TB:hurryAMBATB
    ncsim -cdslib ./cds.lib -logfile ncsim.log -errormax 15 -messages -status
    WORK.HurryAMBA_TB:hurryAMBATB

clean:
    \rm  work/*
```

5. APPENDIX B : COMPILATION AND ELABORATION RESULTS

```
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/canmessagestates.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/cancore_config.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/CANCore.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/CRCCalc.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/StuffHandler.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/CANRx.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/ErrFrameGen.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.

ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/Synchro.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/CANTx.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../SRC/HurriCANe_5.2.4/ErrorCounters.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../TESTBENCH/HurriCANe_5.2.4_TB/CANReference.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncvhdl -V93 -WORK work ../TESTBENCH/HurriCANe_5.2.4_TB/HurriCANe_TB.vhd
ncvhdl: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
ncelab -ACCESS +rw -work work -cdslib ./cds.lib -logfile ncelab.log -errormax 15 -messages -
status -v93 work.HurriCANe_TB:tb_architecture
ncelab: 05.10-s010: (c) Copyright 1995-2004 Cadence Design Systems, Inc.
        Elaborating the design hierarchy:
        Building instance specific data structures.
        Design hierarchy summary:
                              Instances  Unique
                Components:          99      10
                Default bindings:    97      88
                Processes:         1114     100
                Signals:           1558     161
        Writing initial simulation snapshot: WORK.HURRICANE_TB:TB_ARCHITECTURE
ncelab: Memory Usage - 9.3M program + 11.7M data = 21.0M total
ncelab: CPU Usage - 0.1s system + 0.8s user = 0.9s total (1.4s, 67.1% cpu)
```

6. APPENDIX C : SIMULATION RESULTS

```
ncsim -cdslib ./cds.lib -logfile ncsim.log -errormax 15 -messages -status
WORK.HURRICANE_TB:TB_ARCHITECTURE
---> STARTING CRC TEST <---
--->        PASSED !!!        <---
---> STARTING CRC DELIMITER TEST(1) <---
--->        PASSED !!!        <---
---> STARTING CRC DELIMITER TEST(2) <---
--->        PASSED !!!        <---
---> STARTING ACK SLOT TEST (1) <---
--->        PASSED !!!        <---
---> STARTING ACK SLOT TEST (2) <---
--->        PASSED !!!        <---
---> STARTING ACK SLOT TEST (3) <---
--->        PASSED !!!        <---
---> STARTING ACK DELIMITER TEST (1) <---
--->        PASSED !!!        <---
---> STARTING ACK DELIMITER TEST (2) <---
--->        PASSED !!!        <---
---> STARTING END OF FRAME TEST <---
--->        PASSED !!!        <---
---> STARTING OVERLOAD TEST(1) <---
--->        PASSED !!!        <---
---> STARTING OVERLOAD TEST(2) <---
--->        PASSED !!!        <---
---> STARTING OVERLOAD TEST(3) <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 0 <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 1 <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 2 <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 3 <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 4 <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 5 <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 6 <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 7 <---
--->        PASSED !!!        <---
---> STARTING RECEPTION TEST : DLC = 8 <---
--->        PASSED !!!        <---
---> STARTING STUFFED BIT TEST (1) <---
--->        PASSED !!!        <---
---> STARTING STUFFED BIT TEST : SOF STUFFING <---
--->        PASSED !!!        <---
---> STARTING STUFFED BIT TEST : IDENTIFIER STUFFING (DOMINANT) <---
--->        PASSED !!!        <---
```

```
---> STARTING STUFFED BIT TEST : IDENTIFIER STUFFING (RECESSIVE) <---
--->        PASSED !!!       <---
---> STARTING STUFFED BIT TEST : SRR-IDE STUFFING <---
--->        PASSED !!!       <---
---> STARTING STUFFED BIT TEST : EXTENDED IDENTIFIER STUFFING (DOMINANT) <---
--->        PASSED !!!       <---
---> STARTING STUFFED BIT TEST : EXTENDED IDENTIFIER STUFFING (RECESSIVE) <---
--->        PASSED !!!       <---
---> STARTING STUFFED BIT TEST :RTR-R1-R0 STUFFING <---
--->        PASSED !!!       <---
---> STARTING STUFFED BIT TEST : DLC STUFFING (DOMINANT) <---
--->        PASSED !!!       <---
---> STARTING STUFFED BIT TEST : DLC STUFFING (RECESSIVE)<---
--->        PASSED !!!       <---
---> STARTING STUFFED BIT TEST : DATA STUFFING (RECESSIVE)<---
--->        PASSED !!!       <---
---> STARTING STUFFED BIT TEST : DATA STUFFING (DOMINANT)<---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 0 <---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 1 <---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 2 <---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 3 <---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 4 <---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 5 <---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 6 <---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 7 <---
--->        PASSED !!!       <---
---> STARTING TRANSMISSION TEST : DLC = 8 <---
--->        PASSED !!!       <---
---> STARTING ERROR COUNTERS TEST (1) <---
--->        PASSED !!!       <---
---> STARTING ARBITRATION TEST (1) <---
--->        PASSED !!!       <---
---> REMOTE FRAME TEST(1) :  <---
--->        PASSED !!!       <---
---> REMOTE FRAME TEST(2) : REPLY   <---
--->        PASSED !!!       <---
---> REMOTE FRAME TEST(3) : WRONG REMOTE FRAME <---
--->        PASSED !!!       <---
---> REMOTE FRAME TEST(4) : GOOD REMOTE FRAME <---
--->        PASSED !!!       <---
---> REMOTE FRAME TEST(5) : DATA FRAME <---
--->        PASSED !!!       <---
```