

CCIPC - Data Sheet

Prepared by: P. Tosi



Date: 10/02/2015

Verified by: W. Errico



10/02/2015

Approved by: F. Bigongiari (Project Manager)



10/02/2015

Change Document Record

Document title: CCIPC - Data Sheet				
Issue	Date	Total pages	Modified pages	Notes
1	25/05/2010	21	All	First issue
2	16/07/2010	26	11	Added system frequency parameters in Tab.5-1
			15	Added Generic I/O interface description
			22	Added AMBA interface description (Tab.6-2)
			21	Modified CCIPC IRQ and error handling paragraph (§9.2)
			40	Added description of system clock configuration in §5.1
2.1	19/11/2010	57	10	In §4.1 added and update: Modified Tab.4-1 Added Tab.4-2 and Tab.4-3
			10	In §4.1 added Tab.4-5
			13	Modified §4.5
			16	Modified §5.4
			17	In §7.2 : - Modified Generic I/O interface description (Tab. 7-2); - Added Fig. 7-2, Fig. 6-2 and Fig. 7-4
			32	In §10: Modified Tab.9-3, Tab.9-4, Tab.9-5, Tab.9-6, Tab.9-7, Tab.9-8, Tab.9-9, Tab.9-10 and Tab.9-11 Added Tab.9-12
3	20/12/2010	60	41	Added §10
			9	Added §4.1.1. about CCIPC SW requirement
			14	Modified Tab.5-1 and §5.1, §5.2 and §5.3
			18	Modified introduction on §6
			20	Changed Generic I/O periphery description in Tab.6-1
			21	Inverted Q and D signals in Fig.6-3, Fig.6-4, Fig.6-5, Fig.6-6, Fig.6-7
			26	Modified Simulation environment description in §7
			30	Added a new command to execute CCIPC test procedures in §7.3
			37	Change default value of "HurriCANE Configuration register" in Tab.9-3 and of BPR value in Tab.9-6
4	25/02/2011	64	44	Added compilation commad for edac_rom.c file in §10
			44	Modified all CCIPC GUI screenshots in §10
			19	Modified Synchronous Window length and Consumer Heartbeat time description in Tab. 6-1.
			20	Added §6.1.1 - Node-ID Acquisition and Modification
			21	Modified §6.1.2 and §6.1.3
			22	Modified §6.2.2. Added information on Inhibit time and event time for asynchronous TPDO.
			27	Added §7.1.1- IRQ & External Trigger.

Document title: CCIPC - Data Sheet

Issue	Date	Total pages	Modified pages	Notes
			28	Modified Fig. 7-5
			28	Added §7.1.2 to explain CAN bus selection interface.
			30	Modified Fig. 7-6
			32	Modified Fig. 7-8
			32	in §7.4 added additional AMBA Bus information.
			39	In §8.2 added information on CAN MST interface for simulation environment.
			42	In §9 added information on constraint file used during synthesis process.
			48	In §10.2 added information on SDO abort code supported by CCIPC
			53	In §10.2 added information on how to manage IRQ lines.
5	24/03/2011	68	27	In §7.1.1 the relationship between IRQ[] peripheral signals and IRQ condition has been specified.
			28	In §7.1.2 added information on BusSel signal functionality
			28	Modified Fig. 7-5 of Generic I/O interface.
			29	Added Tab. 7-3 information on HOST interface signals needed to connect arbiter module
			30	Modified Fig. 7-6 of Direct Interface.
			32	Modified Fig. 7-8 of AMBA interface.
			37	In §8.1 added information on simulation bit-rate and system frequency.
			49	Added IRQ reference column in
			52	Added information on how to enable or disable CCIPC EDAC protection.
			58	Added information on CCIPC configuration steps in §11.
			61	In §11.3 added naming convention information.
65	Replaced Fig. 11-6 with correct definition of COB-ID Synch Object.			
6	22/04/2011	70	15	Modified Tab. 5-1 with new Source file
			19	Added "SDO Type" item in Tab. 6-1
			20	Added information on "Node count" and "SDO type" items.
			27	In §7.1.1 (Generic Interface) : Added more information on IRQ[0] line . Removed description of Host trigger interface.
			28	Modified Fig. 7-5. Removed Host trigger interface.
			29	In §7.3 (Direct Interface) added description of Host trigger interface
			32	Modified Fig. 7-8. Removed Host trigger interface
			35	In §8 added simulation instruction to point correct CCIPC FSM file
40	Modified Tab. 8-3 according to new Configuration area addressing			

Document title: CCIPC - Data Sheet

Issue	Date	Total pages	Modified pages	Notes
			45	In Tab. 10-2 removed TASI SCET entry
			47	In §10.2 modified information on how to manage IRQ lines according to new IRQ interface
			47	Update Tab. 10-3 according to new Configuration Area definition
			47	Update Tab. 10-4 according to new Configuration Area definition
			49	Update Tab. 10-10 and Tab. 10-12 according to new IRQ register definition
			51	Added explanation of new RPDO and TPDO IRQ management. Added Tab. 10-14, Tab. 10-15, Tab. 10-16, Tab. 10-17, Tab. 10-18, Tab. 10-19, Tab. 10-20 and Tab. 10-21.
			53	Added explanation of new TPDO external trigger register interface. Added Tab. 10-23 and Tab. 10-24
			59	In §11.2 modified VHDL constants list. Added: GENERICH_TECH, SDO_FUNCTION, NODE_COUNT, PDO_IRQCNT, PDOIRQ_IDX and MAX_SRPDO values.
			61	Modified Fig. 11-1 according to new CCIPC GUI interface. Added information on SDO type selection.
7	23/09/2011	83	12	Added §4 - CANOPEN OVERVIEW
			15	Modified SRC file list in Tab. 5 1
			20	In §6.1.1 modified statement on resetting core procedure.
			21	Modified Fig. 6-1 – Acquisition of Node-Id
			24	Modified §7 - CCIPC interface
			27	Modified §7.1.1. Bus switch event assigned to IRQ line
			27	In §7 modified Fig. 7-4 - Read-Modify.
			28	In §7.2 Changed “D” with “Q” in “At the next clock cycle the RD, WR and A and Q(during write cycle) are stable at the interface” sentence.
			28	In §7.1.2 changed “implements” with “includes” in “CCIPC includes a single CAN controller capable to handle two physical busses” sentence.
			32	Modified Tab. 7-6
			37	In §8.1 modified “Stimuli” file description
			39	In Tab. 8-1 modified BLKIDL, BLKDL, BIKIUL and BIKUL definitions.
			40	Modified ID field in Tab. 8-2
			40	Added R_CF_ST_ADDR0 constant definition in Tab. 8 3
			41	Modified Test procedures in §8.3
			48	Changed HOSI Access type of SDO multiplexor field in Tab. 10-3
			48	In §10.2 added information on: MsgIn Queue and Synch Fifo full conditions and T/RPDO generic error

Document title: CCIPC - Data Sheet				
Issue	Date	Total pages	Modified pages	Notes
				conditions.
			49	Changed "Bus selection" nomenclature to "Current Active Bus" in Tab. 10-7
			49	Changed BPR default value from 0 to 1 in Tab. 10-6
			50	Added bus switch flag signal in IRQ status register (Tab. 10-10) and IrQ mask-clear register (Tab. 10-12)
			50	Added indication on RPDO and TPDO generic error in Tab. 10-10
			53	Modified TRX_IRQ and SYNCH IRQ pseudo codes in §10.2
			54	Added SDO download and Upload flag indication on Multiplexor register (Tab. 10-25)
			76	Added §12 - CCIPC RESOURCE O
			78	Added §13 - CCIPC TIMING CHARACTERISTICS
			82	Added §14 – CCIPC Portability
8	28/11/2011	83	15	Modified file list in Tab. 5-1
			15	Modified file list in Tab. 5-2
			16	Modified file list in Tab. 5-3
			47	In Tab. 10-3 modified Cfg Address for Device Type and Error register items.
			52	Added explanation of Hamming code used for EDAC
9	20/07/2012	87	29	Corrected Tab. 7 3 – Generic I/O signals
			30	Corrected Tab. 7 4 – Direct interface signals
			33	Corrected Tab. 7 6 – AHB signals
			39	In Tab. 8-1 modified BLKDL instruction definition. Added \$cflag option.
			47	Modified SDO and HOSI access type in Tab. 10-3
			52	In §10.2 modified EDAC algorithm.
			55	In §10.2 added information on MsgIn Queue full (incoming message queue full) condition management
			63	Updated Fig. 11-3
			75	Update §12 with new resource occupation information
			77	In §13 updated timing information and added §13.1 with more details on TPDO service.
10	03/08/2012	87	15	Adjusted file name in Tab. 5-1. Adjusted file name in Tab. 5-2
			21	Added Tab. 6-3 to explain the association between CCIPC "Mask" input port and "ID Mask" register value.
			45	In §10 added information on CCIPC Initialization Area.
			47	In Tab. 10 3 modified access type of Ctoggle entry to Read and clean. A write access allows cleaning register value.
			50	Added Tab. 10-11 reporting Synch FIFO size information
			77	In §13, modified explanation of possible Heartbeat Consumer timing mismatch.
			84	Corrected timing values on Tab. 13-5

Document title: CCIPC - Data Sheet				
Issue	Date	Total pages	Modified pages	Notes
			85	Corrected timing values on Tab. 13-6
11	31/10/2012	140	All	In all document replaced Actel reference with Microsemi.
			13	In §4 explained with more detail the CANopen features supported by CCIPC.
			21	In §5.1 highlighted that SW requirement refers to the minimum tested.
			23	In §5.2 added Fig. 5-2 adding more details of SRC files.
			24	Added §5.2.1 with a detailed description of Libraries directory. A description of RAM model characteristics has been included.
			25	Incorporate previous TESTBENCH directory section in §5.2.1.
			25	Renamed §5.2.2. Added also HurriCANE file list.
			26	Modified §5.4 about Synthesis script directory.
			27	Modified §5.5 about Fitting script directory
			30	Added §6.1.4.1. – “SDO Implementation: CCIPC FSMs” reporting additional information about CCIPC FSMs implemented.
			32	Modified §7: - Included Memory interface description in §7.1.1, - Added Configuration Area Description in §7.1.2, - Included in a §7.2 description of all 3 Host interfaces; - Added §7.1.6 with a simple description of Reset distribution inside CCIPC.
			36	Moved previous §6.1.1 “Node-ID Acquisition and Modification” to §7.1.4
			41	In §7.2.3 added information on supported AMBA features for both CCIPC Master and Slave interfaces.
			44	In §8 modified instructions for CCIPC simulations and test execution.
			49	Added §8.3 describing simulation Log file.
			50	Split §8.4 in two paragraphs: 8.4.1 - CCIPC Default Tests and 8.4.2 - User Defined Tests.
			54	Added §9.2 with instructions to synthesize and fit CCIPC for Xilinx.
56	§10 has been completely reviewed.			
95	In §13 added some information on CCIPC tasks scheduling.			
95	In §13 replaced “Segment Reply” with “Block Reply” in SDO Block Download table.			
104	In §14.3 inserted additional information on EDAC feature implemented in CCIPC.			
12	04/12/2012	109	15	In §4.1 added that CCIPC is a SYNC consumer
			17	In §4.5.3 substituted Ctoggle with Ntoggle.
			18	Revisited §4.6.1.1.1.

Document title: CCIPC - Data Sheet

Issue	Date	Total pages	Modified pages	Notes
			18	Revisited §4.6.1.2.1
			22	Modified Tab. 4-11
			22	Modified Tab. 4-12
			23	Modified Tab. 4-13
			33	In §6.1.4.1 removed warning on FSM implementation.
			37	In §7.1.1 changed CCIPC endianness from big to little
			39	In Fig. 7-6 changed Q[n] with D[n]
			39	In §7.1.4 corrected reference to Tab. 7-5 and Fig. 7-7
			56	In §9.1.1 removed warning on FSM implementation.
			58	In §9.2.1 removed warning on FSM implementation.
60	In §10.1 changed host access type of: - Error Register; - msgin_full bit of HurriCANE & CCIPC status register; - IRQ Mask-clear register; - SDO multiplexor			
103	In §13 modified parameters name of SDO Download and Upload Block to match the standard CANopen nomenclature.			
13	21/03/2013	109	18	Modified Synchronous Window Length error in §4.6.1.1.1
			18	Replaced TPDO with RPDO in §4.6.1.2.1
			22	In §4.6.2.2 included additional information on SDO segmented transfer.
			22	In §4.6.2.3 included additional information on SDO block transfer.
			40	In §7.1.6 added information about CCIPC reset.
			66	Replaced correct address of Ctoggle register.
			66	Replaced correct address of ID Base register.
			67	Replaced correct address of ID Mask register.
			75	Modified access type of "index" field in SDO multiplexor register.
			89	Modified Fig. 11-3 - Node Id setting
101	Added Tab. 12-1 with information of CCIPC internal RAM modules occupation.			
14	20/06/2013	109	18	In §4.6.1.2.1 modified explanation of RPDO synchronous error detection.
			22	In §4.6.2.3 removed any references of SDO Segmented and modified note on Tab. 4-12.
			28	In §5.3 removed reference about ADDON_TB tests.
			54	In §8.4.1 removed reference about ADDON_TB tests.
15	23/07/2013	109	18	In §4.6.1.2.1 modified explanation of RPDO synchronous error detection.
16	08/05/2014	110	70	In §10.1 added information on IRQ masking.
			74	In §10.1 added explanation of <i>serr_mask</i> and <i>derr_mask</i> field of EDAC error register.
			77	In §10.1 added Tab. 10-7.
17	10/02/2015	113	35	In §7 modified figures Fig. 7-2, Fig. 7-3, Fig. 7-4, Fig.

Document title: CCIPC - Data Sheet

Issue	Date	Total pages	Modified pages	Notes
				7-5, Fig. 7-6, Fig. 7-7 and Fig. 7-10.
			81	In §10.3.1 added Tab. 10-10 which shows the Shared Memory occupation of Mapping Parameter structure.
			82	In §10.3.2 added Tab. 10-11 and Fig. 10-1 to illustrate the mapping of Application Objects in the Shared memory.

Table of Contents

1 INTRODUCTION..... 12

2 DOCUMENTS..... 13

2.1 *Reference Documents..... 13*

3 LIST OF ACRONYMS AND ABBREVIATIONS..... 14

4 CANOPEN OVERVIEW..... 15

4.1 *Communication Model..... 15*

4.2 *Object Dictionary..... 15*

4.3 *Data Type..... 16*

4.4 *Object Type..... 17*

4.5 *Network Management Objects..... 17*

4.5.1 *Module Control Services..... 17*

4.5.2 *Bootup Service..... 17*

4.5.3 *Error Control Service and Bus Redundancy..... 17*

4.6 *Communication Objects..... 17*

4.6.1 *Process Data Object (PDO)..... 17*

4.6.1.1 *Transmit PDO..... 18*

4.6.1.1.1 *Synchronous..... 18*

4.6.1.1.2 *Asynchronous..... 18*

4.6.1.2 *Receive PDO..... 18*

4.6.1.2.1 *Synchronous..... 18*

4.6.1.2.2 *Asynchronous..... 19*

4.6.2 *Service Data Object (SDO)..... 19*

4.6.2.1 *CCIPC – Expedited Configuration..... 21*

4.6.2.2 *CCIPC – Segmented Configuration..... 22*

4.6.2.3 *CCIPC – Block Configuration..... 22*

5 IP CORE DATABASE..... 24

5.1 *Software Requirement..... 24*

5.2 *SRC Directory..... 25*

5.2.1 *LIBRARIES Directory..... 27*

5.2.1 *TESTBENCH Directory..... 27*

5.2.2 *CAN_CORE Directory..... 28*

5.3 *SIM Script Directory..... 28*

5.4 *SYN Script Directory..... 29*

5.5 *FIT Script Directory..... 30*

5.6 *CONFIG File Directory..... 30*

5.7 *CONFIG Tool Directory..... 30*

6 CCIPC CONFIGURATION..... 31

6.1 *CCIPC Set-up and Configuration..... 31*

6.1.1	Synchronous Window Length	32
6.1.2	Consumer Heartbeat Time.....	32
6.1.3	Producer Heartbeat Time.....	32
6.1.4	SDO Server Configuration	32
6.1.4.1	SDO Implementation: CCIPC FSMs	33
6.1.5	Bus Manager Parameters	33
6.2	Communication Objects Configuration.....	33
6.2.1	Receive PDO	34
6.2.2	Transmit PDO	34
6.3	Application Object Configuration.....	34
7	INTERFACE.....	35
7.1	CCIPC Periphery Signals	36
7.1.1	Memory Interface	37
7.1.2	Configuration Area Interface.....	38
7.1.3	IRQ & External Trigger.....	39
7.1.4	Node-ID Acquisition and Modification	39
7.1.5	CAN Bus Selection	40
7.1.6	Reset Distribution.....	40
7.2	Host Interfaces	41
7.2.1	Generic Interface	41
7.2.2	Direct I/O Interface.....	42
7.2.3	AMBA Bus Interface.....	44
8	SIMULATION ENVIRONMENT	47
8.1	CCIPC Core Test-Bench	49
8.2	Input Stimuli Format	50
8.3	Output Test Log File	52
8.4	Test Procedures Simulation.....	54
8.4.1	CCIPC Default Tests.....	54
8.4.2	User Defined Tests	55
9	SYNTHESIS AND FITTING SCRIPTS	56
9.1	Microsemi (RT)AX FPGA.....	56
9.1.1	Synthesis.....	56
9.1.2	Fitting	57
9.2	Xilinx FPGA	58
9.2.1	Synthesis.....	58
9.2.2	Fitting	59
10	CCIPC MEMORY MAPPING	60
10.1	Configuration & Status Area	60
10.1.1	IRQ Handling	77
10.2	On-board Area.....	79
10.2.1	PDO Communication Entries.....	79
10.2.2	AOs Addressing Pointers	80
10.3	Shared Memory Area	80
10.3.1	PDO Mapping Parameters	81

10.3.2	Application Objects Area	82
11	CONFIGURATION TOOL	84
11.1	ROM IMAGE FILE	84
11.2	VHDL Configuration File	85
11.3	CCIPC GUI	86
11.4	CCIPC Set Up Window	88
11.5	Configuration Object Window	90
11.6	Application Object	92
11.7	Communication Object	95
12	CCIPC RESOURCE OCCUPATION	101
13	CCIPC TIMING CHARACTERISTICS	103
13.1	TPDO Numerical Example	106
13.1.1	CCIPC with 32 Node	107
13.1.2	CCIPC with 16 Node	108
13.1.3	CCIPC with 8 Node	108
13.1.4	CCIPC with 4 Node	109
13.1.5	CCIPC with 2 Node	110
13.1.6	CCIPC with 1 Node	110
14	CCIPC PORTABILITY	112
14.1	CAN Bus Controller	112
14.2	FPGA Technology	112
14.3	Rad-Hard Technique	112

1 INTRODUCTION

This document acts as CAN Controller IP Core data sheet and toolset user manual. The Core features, its development environment and the foreseen configuration options are here described.

A quick start approach to CCIPC should pass through the following steps:

- 1) Set-up the CCIPC database inserting HurriCANE core (see §5.2.2). The IP Core Database is presented in §4 with a detailed description its directories structure and main files
- 2) Run standard configuration tests. The simulation environment set-up is presented in §8.3, with explicit information about the files involved in the simulation process.
- 3) Generate custom configuration defining interface, main parameters and Object Dictionary (OD). The parameters functions are described in §6. In §7 the different options for the core host interface are presented. The CCIPC graphical configuration tool is described in §11 where the user is guided step-by-step in the object dictionary definition.
- 4) Simulate application specific core instance the explanation how to customise the simulation environment and the compiler tool are provided in §8.
- 5) Synthetize and Fit application specific configuration. Section §9 provide users with basic instructions for synthetizing and fitting of the CCIPC core instance in the Microsemi Axcelerator technology.

CCIPC area occupation and operating frequency for different CCIPC configuration and FPGA technologies are reported in §12.

In §13 an estimation of elaboration time of the main CANopen features is reported.

A brief description of CCIPC portability is illustrated §14

2 DOCUMENTS

2.1 Reference Documents

[RD 1] CiA Draft Standard 301 Version 4.02

[RD 2] CiA Draft Standard 306 Version 1.3

3 LIST OF ACRONYMS AND ABBREVIATIONS

Abbreviation	Meaning
AHB	Advanced High-performance Bus
AMBA	Advanced Microcontroller Bus Architecture
AO	Application Object
CAN	Controller Area Network
CAN MST	CAN Master
CCIPC	CANOPEN Controller IP core
CiA	CAN In Automation
COB-ID	Communication Object Identifier
DCF	Device Configuration File
EDAC	Error Detection And Correction
EDS	Electronic Data Sheet
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GUI	Graphical User Interface
HB	Heartbeat
HW	Hardware
IRQ	Interrupt Request
NMT	Network Management
OD	Object Dictionary
PDO	Process Data Object
RAM	Random Access Memory
ROM	Read Only Memory
RPDO	Receive PDO
SDO	Service Data Object
SW	Software
SYNC	Synchronization Object
TPDO	Transmit PDO
U8	Unsigned 8
U16	Unsigned 16
U32	Unsigned 32
UUT	Unit Under Test
VHDL	VHSIC (Very High Speed Integrated Circuit) HW Description Language

4 CANOPEN OVERVIEW

In this paragraph, an overview of CANopen services supported by the CANopen Controller IP Core (CCIPC) core is provided.

The CAN in Automation (CiA) Standard “CANopen Application Layer and Communication Profile” ([RD 1]) is used as reference document for the CANopen standard.

4.1 Communication Model

The CCIPC is designed to act as:

- **SLAVE** node in a CANopen network, responding to a master request;
- **SERVER** node for SDO Download and Upload services;
- **PRODUCER** and **CONSUMER** node for PDO and Heartbeat services
- **CONSUMER** of SYNC message;

4.2 Object Dictionary

The CCIPC supports the Object Dictionary implementation. A limited set of Object Dictionary entries are supported as shown in next table.

The CCIPC Object Dictionary foresees a dedicated area (entries starting from index 2000h) that defines CCIPC specific parameters and two macro areas (Read-Write and Read-Only) that allows user to define its specific Application Objects.

Index	Sub-Index	Entry Name
1000h	0h	Device Type
1001h	0h	Error register
1005h	0h	COB-ID synch
1007h	0h	Synchronous Window length
1016h	0h	Consumer Heartbeat timer
	1h	Master & Consumer Heartbeat time
1017h	0h	Producer Heartbeat time
1018h	0h	Identity object
	1h	Vendor-ID
	2h	Product code
	3h	Revision number
	4h	Serial Number
SDO Server Parameter		
1200h	0h	Server SDO Parameter
	1h	COB-ID client-server
	2h	COB-ID server-client
RPDO Parameter		
1400h 15FFh	0h	RPDO Communication
	1h	COB-ID
	2h	Transmission Type
1600h 17FFh	0h	RPDO Mapping
	1h – 8h	Mapping Parameter
TPDO Parameter		
1800h 19FFh	0h	RPDO Communication
	1h	COB-ID

Index	Sub-Index	Entry Name
	2h	Transmission Type
	3h	Inhibit Time
	4h	Reserved
	5h	Inhibit Time
1A00h	0h	TPDO Mapping
1BFFh	1h – 8h	Mapping Parameter
CCIPC Parameter		
	0h	
	1h	Bdefault
2000h	2h	Ttoggle
	3h	Ntoggle
	4h	Ctoggle
2001	0h	Filler entry
	0h	
2002h	1h	ID base
	2h	ID mask
	0h	
	1h	HurriCANE configuration
	2h	HurriCANE & CCIPC status
	3h	CCIPC-SDO error
	4h	IRQ status
	5h	IRQ mask-clear
	6h	RPDO IRQ0
	7h	RPDO IRQ1
2003h	8h	RPDO IRQ2
	9h	RPDO IRQ3
	Ah	TPDO IRQ0
	Bh	TPDO IRQ1
	Ch	TPDO IRQ2
	Dh	TPDO IRQ3
	Eh	EDAC error
	Fh	TPDO Trig
	10h	SDO multiplexor
Read-Write Area		
6000h	0h – 254h	User Defined Application Objects
6FFFh		
Read-Only Area		
7000h	0h – 254h	User Defined Application Objects
7FFFh		

Tab. 4-1: CCIPC Object Dictionary Layout.

4.3 Data Type

The CCIPC supports the following types to define the data type of each Application Object:

- Unsigned 8 (U8);
- Unsigned 16 (U16);
- Unsigned 32 (U32).

4.4 Object Type

The CCIPC supports the following Object types to define the type of the Application Objects entry:

- **Variable** (VAR): single value;
- **Record**: multiple data field object composed by a combination of simple variables;
- **Array**: multiple data field object composed by a combination of simple variables of the same data type.

Each variable can be defined using only the U8, U16 or U32 data types.

4.5 Network Management Objects

The CCIPC works as slave node in Network Manager service. Through this service a master is in charge of managing the status of each slave in the CAN net.

4.5.1 Module Control Services

The CCIPC supports the following services:

- Start Remote Node: CCIPC enters in OPERATIONAL state;
- Stop Remote Node: CCIPC enters in STOP state;
- Enter Pre-Operational: CCIPC enters in PRE-OPERATIONAL state;
- Reset Node: CCIPC enters RESET APPLICATION state;
- Reset Communication: CCIPC enters RESET COMMUNICATION state.

CCIPC supports a special feature that allows CCIPC to enter directly in OPERATIONAL state at the end of initialization phase when a dedicated flag is active.

4.5.2 Bootup Service

CCIPC supports the Boot-up service signaling to the Master that the initialization phase ended.

4.5.3 Error Control Service and Bus Redundancy

The CCIPC supports the Heartbeat protocol to detect failures in a CAN network. CCIPC is in charge of receiving and transmitting Heartbeat messages.

When CCIPC does not receive the Heartbeat message from the Master node it generates an "Heartbeat event".

This event is used by CCIPC to implement the "Bus Redundancy Management" protocol. Through this protocol CCIPC is able to control two CAN buses (nominal and redundant). CCIPC periphery is furnished with a bus selection flag that, in a multiplexing way, allows defining which is the currently CAN active bus.

Two parameters are available to control the redundancy protocol:

- **Ttoggle** counter
- **Ntoggle** counter

The **Ttoggle** counter defines the maximum number of Heartbeat events causing a bus switch.

The **Ntoggle** counter defines the maximum number of bus toggling before CCIPC stops the redundancy process.

4.6 Communication Objects

4.6.1 Process Data Object (PDO)

The real time data transfer is performed by the "PDO" service. Two kinds of PDO are supported:

- Transmit PDO (TPDO) – CCIPC transmits data;
- Receive PDO (RPDO) – CCIPC receives data.

4.6.1.1 Transmit PDO

The CCIPC supports the **Synchronous** and **Asynchronous** transmission modes for TPDO service.

4.6.1.1.1 Synchronous

The transmission of this type of TPDO is associated to the following trigger:

- *Event trigger*: expiration of the specified transmission period associated to the reception of SYNCH message;

The Synchronous TPDO service is controlled by an additional parameter, the Synchronous Window Length that defines a time window within which the synchronous TPDOs have to be sent. If one synchronous TPDO cannot be sent before the synchronous window length expiration, an error is signaled by CCIPC.

4.6.1.1.2 Asynchronous

The transmission of this type of TPDO is associated to the following triggers:

- *Event trigger*: external host device issues a TPDO transmission request;
- *Timer Driven*: expiration of the transmission period associated to internal timers defined as multiple of milliseconds.

Two-timer parameters can be associated to an Asynchronous TPDO:

- *Event Time*: its expiration is considered an event that causes the transmission of the TPDO;
- *Inhibit Time*: it defines the minimum interval for TPDO transmission.

The following table shows all the possible TPDO configurations supported by CCIPC. Note that for the Asynchronous TPDO the Event trigger feature is always enabled and it is in charge of the host device to use it or not.

Type	SWL	Event trigger	Event Time	Inhibit Time	Description
S	yes	no	no	no	The TPDO is sent at expiration of synch period and the check on SWL is enabled.
	no	no	no	no	The TPDO is sent at expiration of synch period
AS	no	yes	no	no	The TPDO is sent only when external trigger is issued
	no	yes	no	yes	The TPDO is sent only when external trigger is issued and the inhibit period is expired.
	no	yes	yes	no	The TPDO is sent both when external trigger is issued and when even timer expires.
	no	yes	yes	yes	The TPDO is sent both when external trigger is issued or when even timer expires and the inhibit period is expired. The event time is reloaded to default value when TPDO transmission is caused by external trigger.

Tab. 4-2: CCIPC – possible TPDO configurations. S-synchronous, AS-Asynchronous, SWL-Synchronous Window Length.

4.6.1.2 Receive PDO

The CCIPC supports the **Synchronous** and **Asynchronous** elaboration modes for RPDO service. In fact a RPDO message is received in asynchronous way but the CCIPC elaborates the content immediately or not according to the RPDO type.

4.6.1.2.1 Synchronous

The elaboration of this type of RPDO is associated to the reception of the SYNC message.

When the RPDO is received, CCIPC stores it in a dedicated internal queue and only at reception of the SYNC message it starts the elaboration of the message.

As for synchronous TPDO, also the Synchronous RPDO service is controlled by the Synchronous Window Length parameter that defines a time window within which all the synchronous RPDOs received have to be elaborate. If one of the synchronous RPDOs is not elaborated within the synchronous window length expiration, an error is signaled by CCIPC.

4.6.1.2.2 Asynchronous

The CCIPC starts the elaboration of this type of RPDO immediately after its reception.

4.6.2 Service Data Object (SDO)

The SDO service allows a client to access the Object Dictionary of a server node to read (SDO Upload) or write (SDO Download) specific Object Dictionary entries.

The SDO foresees 3 different type of services:

- *Expedited*: it is used to transfer up to 4 bytes;
- *Segmented*: it is used to transfer more than 4 bytes as a sequence of segments;
- *Block*: it is used to transfer more than 4 bytes as a sequence of blocks without CRC feature;

The difference between Segmented and Block services is that Block service allows an higher bus utilization. In fact, segment service foresees that each segment of 7 bytes is transferred in an acknowledged way, while Block service allows transferring up to 127 segments of 7 bytes consecutively. CCIPC supports the SDO Abort service. It is in charge to notify errors if a SDO protocol error is detected and stopping SDO transfer when an SDO abort is received.

The CCIPC is in charge to detect and transmit the following abort codes:

Abort Code (Hex)	Description
05030000	Toggle bit not alternated
05040001	Client Server command specifier not valid or unknown
06010002	Attempt to write a read only object
06070010	Data type does not match, length of service parameter does not match
06090011	Subindex does not exist
05040002	Invalid Block size
05040003	Invalid Sequence number
08000000	General Error

Tab. 4-3: CCIPC Abort condition and test configurations.

The following tables report the SDO access type supported by CCIPC for each Object Dictionary entry:

Index	Sub-Index	Entry Name	Obj Type	SDO access	SDO type
1000h	0h	Device Type	VAR_U32	Ro	Expedited
1001h	0h	Error register	VAR_U8	Ro	Expedited
1005h	0h	COB-ID synch	VAR_U32	Ro	Expedited
1007h	0h	Synchronous Window length	VAR_U32	Ro	Expedited
1016h	0h	Consumer Heartbeat timer	U8	Ro	Expedited
	1h	Master & Consumer Heartbeat time	U32	Rw	Expedited
1017h	0h	Producer Heartbeat time	U16	Rw	Expedited
1018h	0h	Identity object	U8	Ro	Expedited
	1h	Vendor-ID	U32	Ro	Expedited
	2h	Product code	U32	Ro	Expedited
	3h	Revision number	U32	Ro	Expedited
	4h	Serial Number	U32	Ro	Expedited

Tab. 4-4: CCIPC - SDO Access: Communication Area.

Index	Sub-Index	Entry Name	Obj Type	SDO access	SDO type
1200h	0h	Server SDO Parameter	U8	Ro	Expedited
	1h	COB-ID client-server	U32	Ro	Expedited
	2h	COB-ID server-client	U32	Ro	Expedited

Tab. 4-5: CCIPC - SDO Access : SDO Server Parameter.

Index	Sub-Index	Entry Name	Obj Type	SDO access	SDO type
1400h 15FFh	0h	RPDO Communication	U8	Ro	Expedited
	1h	COB-ID	U32	Ro	Expedited
	2h	Transmission Type	U8	Ro	Expedited
1600h 17FFh	0h	RPDO Mapping	U8	Ro	Expedited
	1h – 8h	Mapping Parameter	U32	Ro	Expedited

Tab. 4-6: CCIPC - SDO Access : RPDO parameters.

Index	Sub-Index	Entry Name	Obj Type	SDO access	SDO type
1800h 19FFh	0h	RPDO Communication	U8	Ro	Expedited
	1h	COB-ID	U32	Ro	Expedited
	2h	Transmission Type	U8	Ro	Expedited
	3h	Inhibit Time	U16	Ro	Expedited
	4h	Reserved	U8	Ro	Expedited
	5h	Inhibit Time	U16	Ro	Expedited
1A00h 1BFFh	0h	TPDO Mapping	U8	Ro	Expedited
	1h – 8h	Mapping Parameter	U32	Ro	Expedited

Tab. 4-7: CCIPC - SDO Access : TPDO parameters.

Index	Sub-Index	Entry Name	Obj Type	SDO access	SDO type
2000h	0h		U8	Ro	Expedited
	1h	Bdefault	U8	Ro	Expedited
	2h	Ttoggle	U8	Rw	Expedited
	3h	Ntoggle	U8	Rw	Expedited
	4h	Ctoggle	U8	Rc	Expedited
2001	0h	Filler entry	U8	--	--
2002h	0h		U8	Ro	Expedited
	1h	ID base	U8	Ro	Expedited
	2h	ID mask	U8	Ro	Expedited
2003h	0h		U8	Ro	Expedited
	1h	HurriCANE configuration	U32	Ro	Expedited
	2h	HurriCANE & CCIPC status	U32	Ro	Expedited
	3h	CCIPC-SDO error	U32	Ro	Expedited
	4h	IRQ status	U32	Ro	Expedited
	5h	IRQ mask-clear	U32	Ro	Expedited
	6h	RPDO IRQ0	U32	Ro	Expedited
	7h	RPDO IRQ1	U32	Ro	Expedited
	8h	RPDO IRQ2	U32	Ro	Expedited
9h	RPDO IRQ3	U32	Ro	Expedited	

Index	Sub-Index	Entry Name	Obj Type	SDO access	SDO type
	Ah	TPDO IRQ0	U32	Ro	Expedited
	Bh	TPDO IRQ1	U32	Ro	Expedited
	Ch	TPDO IRQ2	U32	Ro	Expedited
	Dh	TPDO IRQ3	U32	Ro	Expedited
	Eh	EDAC error	U32	Ro	Expedited
	Fh	TPDO Trig	U32	Ro	Expedited
	10h	SDO multiplexor	U32	Ro	Expedited

Tab. 4-8: CCIPC - SDO Access : CCIPC parameters.

Index	Sub-Index	Entry Name	Obj Type	SDO access	SDO type
6000h 6FFFh	0h	Read-Write Area	U8	Ro	Expedited
	1h – 254h		U8 U16 U32	Rw	Expedited Segmented Block

Tab. 4-9: CCIPC - SDO Access : Application Objects Read-Write Area.

Index	Sub-Index	Entry Name	Obj Type	SDO access	SDO type
7000h 7FFFh	0h	Read-Write Area	U8	Ro	Expedited
	1h – 254h		U8 U16 U32	Ro	Expedited Segmented Block

Tab. 4-10: CCIPC - SDO Access : Application Objects Read-Only Area.

The CCIPC can be configured in 3 different ways according to supported SDO service:

- **Expedited:** CCIPC supports only SDO Expedited service;
- **Segmented:** CCIPC supports SDO Expedited and Segmented services;
- **Block:** CCIPC supports SDO Expedited and Block service.

As illustrated in Tab. 4-9 and Tab. 4-10 the SDO Segmented or Block service can be used only to access the Read-Write or Read-Only areas.

In these areas the user has the possibility to add its specific entries defining them as Array, Variable or Record. Next paragraphs describe how CCIPC replies to an incoming SDO request according to Type of entry addressed and CCIPC configuration selected.

4.6.2.1 CCIPC – Expedited Configuration

Tab. 4-11 illustrates how CCIPC, in its Expedited configuration, replies to a SDO request that addresses an entry defined in the Read-Write or Read-Only area.

SDO request		Variable	Record	Array
Expedited	Download	Accepted	Depends on addressed sub-Index: <ul style="list-style-type: none"> • 0 – Abort • Other – Accepted 	Depends on addressed sub-Index: <ul style="list-style-type: none"> • 0 – Abort • Other – Accepted
	Upload	Accepted	Accepted	Depends on addressed sub-Index: <ul style="list-style-type: none"> • 0 – Accepted • Other – Abort

Segmented	Download	Abort	Abort	Abort
	Upload	(*)	(*)	(*)
Block	Download	Abort	Abort	Abort
	Upload	Abort	Abort	Abort

(*) Expedited and Segmented Upload requests have the same structure. It's in charge of CCIPC selecting the correct reply. In this case the same principles of Expedited are applied to Segmented Upload.

Tab. 4-11: CCIPC Expedited : SDO reply behaviour.

4.6.2.2 CCIPC – Segmented Configuration

Tab. 4-12 illustrates how CCIPC, in its Segmented configuration, replies to a SDO request that addresses an entry defined in the Read-Write or Read-Only area. For SDO Segmented, CCIPC is able to transfer data from the pointed sub-index (different from 0) to the end of the array.

SDO request		Variable	Record	Array
Expedited	Download	Accepted	Depends on addressed sub-Index: <ul style="list-style-type: none"> 0 – Abort Other – Accepted 	Depends on addressed sub-Index: <ul style="list-style-type: none"> 0 – Abort Other – Accepted
	Upload	Accepted.	Accepted	Accepted. if addressed Sub-index <ul style="list-style-type: none"> 0 – Expedited Other – Segmented
Segmented	Download	Abort	Abort	Depends on addressed sub-Index: <ul style="list-style-type: none"> 0 – Abort Other – Accepted
	Upload	(*)	(*)	(*)
Block	Download	Abort	Abort	Abort
	Upload	Abort	Abort	Abort

(*) Expedited and Segmented Upload requests have the same structure. It's in charge of CCIPC selecting the correct reply. In this case the same principles of Expedited are applied to Segmented Upload.

Tab. 4-12: CCIPC Segmented: SDO reply behaviour.

4.6.2.3 CCIPC – Block Configuration

Tab. 4-13 illustrates how CCIPC, in its Block configuration, replies to a SDO request that addresses an entry defined in the Read-Write or Read-Only area. For SDO Block, CCIPC is able to transfer data from the pointed sub-index (different from 0) to the end of the array.

SDO request		Variable	Record	Array
Expedited	Download	Accepted	Depends on addressed sub-Index: <ul style="list-style-type: none"> 0 – Abort Other – Accepted 	Depends on addressed sub-Index: <ul style="list-style-type: none"> 0 – Abort Other – Accepted

SDO request		Variable	Record	Array
	Upload	Accepted.	Accepted	Accepted. if addressed Sub-index <ul style="list-style-type: none"> • 0 – Expedited • Other – Abort
Segmented	Download	Abort	Abort	Abort
	Upload	(*)	(*)	(*)
Block	Download	Abort	Abort	Depends on addressed sub-Index: <ul style="list-style-type: none"> • 0 – Abort • Other – Accepted
	Upload	Abort	Abort	Depends on addressed sub-Index: <ul style="list-style-type: none"> • 0 – Abort • Other – Accepted

(*) Expedited and Segmented Upload requests have the same structure. Since SDO Segmented is not supported in Block configuration, the CCIPC replies with the same rules of SDO Expedited.

Tab. 4-13: CCIPC Block: SDO reply behaviour.

5 IP CORE DATABASE

The CCIPC database tree is described. Fig. 5-1:

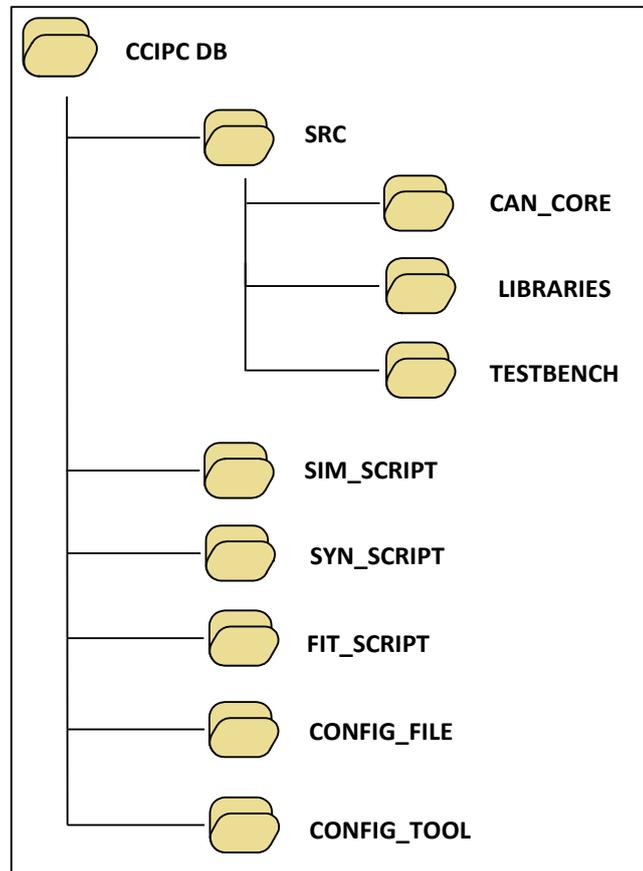


Fig. 5-1: CCIPC database tree.

It is composed of the following directories:

- **SRC** – It includes all the VHDL files that constitutes the CCIPC architecture
- **SIM SCRIPT** – It includes the scripts and file used to compile the CCIPC core
- **SYN SCRIPT** – It includes files needed to perform the CCIPC synthesis process
- **FIT SCRIPT** – It includes files needed to perform the CCIPC fitting process
- **CONFIG FILE** – it includes the configuration files needed by the CCIPC configuration tool
- **CONFIG TOOL** – it includes the executable CCIPC configuration tool

5.1 Software Requirement

The minimum software requirement for Linux kernel and GNU tools versions that has been tested is the following:

- Linux Kernel 2.6.27-11-generic
- GNU make 3.81
- gcc version 4.3.2

- Perl V5.10.0
- perl-Tk V804.028

All the CCIPC SW features has been tested using Ubuntu 8.10 (Linux Kernel 2.6.27-11-generic) and Ubuntu 9.10 (Linux Kernel 2.6.31-22-generic).

5.2 SRC Directory

The **SRC** directory contains the source files that fully describe the CCIPC core in behavioural synthesizable VHDL format, and (into separated subdirectories) the additional files needed to map CCIPC in specific technologies, to compose its simulation environment, and to house the low level CAN controller core.

The CCIPC proper VHDL source files are listed in Tab. 5-1:

Name	Description
Common Files	
CCIPC_conf.vhd	CCIPC configuration file
CANopen_pkg.vhd	CANopen constant parameters
mem_tech.vhd	Definition of memory blocks using specific technology
EDAC_pack.vhd	Definition of EDAC package
CCIPC.vhd	CCIPC top level
CCIPC_interface.vhd	CCIPC CAN interface controller
ccipc_can.vhd	CCIPC can module. It embodies both CCIPC_interface and CAN controller entities.
MsgInController.vhd	Queue controller of message received
MsgOutController.vhd	Queue controller of message to be transmitted
NetworkManager.vhd	Network Manager
event_handler.vhd	Event handler
OD_handler.vhd	CANopen communication protocol handler
MemRF.vhd	OD_handler service registers mapped on on-chip memory
CCIPC_FSM_EXP.vhd	FSM implementation of the CANopen protocol services: T/RPDOs, SDO expedited
CCIPC_FSM_SEG.vhd	FSM implementation of the CANopen protocol services. T/RPDOs, SDO expedited and segmented
CCIPC_FSM_BLK.vhd	FSM implementation of the CANopen protocol services. T/RPDOs, SDO expedited and block
CCIPC_FSM_DIRECT.vhd	FSM implementation of the CANopen protocol services for Direct interface: T/RPDOs, SDO expedited
MemController.vhd	Memory Controller
Generic I/O interface Files	
CCIPC_GENIO.vhd	Top entity for Generic I/O interface
Arbiter.vhd	Generic I/O interface memory arbiter
AMBA interface files	
CCIPC_AHB.vhd	Top entity for AMBA Bus interface
ahb_arbiter_4m.vhd	AMBA BUS arbiter. It supports up to 4 AHB masters
ahb_mst.vhd	AMBA BUS master module
ahb_sslv.vhd	AMBA BUS slave module
ahb_pkg.vhd	AMBA BUS parameters file
regfifo.vhd	AMBA master and slave service FIFO

Name	Description
Direct interface files	
CCIPC_DIRECT.vhd	Top entity for Direct interface
IDX_ROM.vhd	Static Index Table when DIRECT interface is selected
COM_ROM.vhd	Static Communication Parameters table when DIRECT interface is selected
MAP_AO_ITF.vhd	Static Mapping Parameters and Application Objects table when DIRECT interface is selected
Makefile	VHDL Makefile
Makefile.mem	Technology Makefile

Tab. 5-1: CCIPC SRC files.

The following figure illustrates the dependencies between the modules constituting the possible CCIPC configurations.

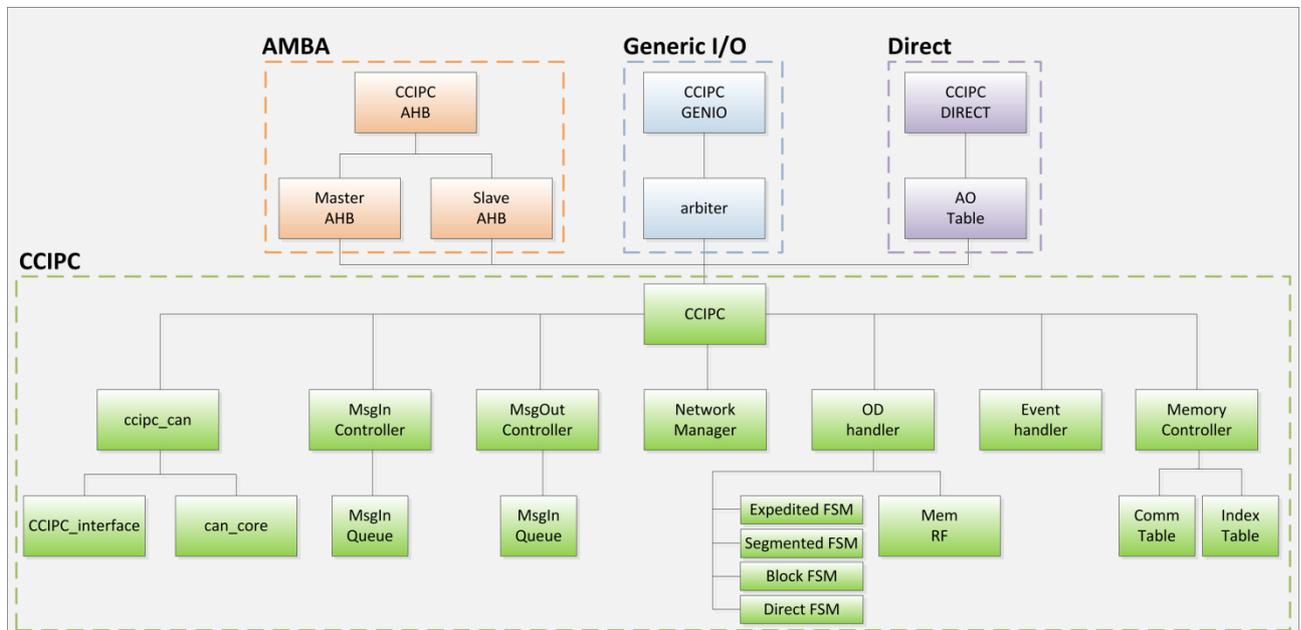


Fig. 5-2: CCIPC SRC directory tree.

The green modules define the CCIPC architecture composed of:

- **CAN interface:** composed by the CAN controller (*can_core*) and its interface manager (*CCIPC_interface*);
- **Messages queues:** they store message received (*MsgIn Controller*) and message to be transmitted (*MsgOut Controller*). The queues are implemented using memory devices (*MsgIn Queue, MsgOut Queue*);
- **Network Manager:** it is the manager of the Network Management Services, controlling the NMT state and performing the Bus Redundancy management (*Network Manager*);
- **Object Dictionary Handler:** composed by an event scheduler (*event_handler*) that controls the engine of the CCIPC (*OD handler*) in order to elaborate the Communication services (RPDO, TPDO, SDO)
- **Memory controller:** it is the module that allows accessing the whole Object Dictionary.

The CCIPC can be configured using three different interfaces to communicate with the Host device:

- **Generic I/O (CCIPC GENIO):** it is a generic memory-like interface in which an *arbiter* module manages the communication between Host and CCIPC;
- **AMBA (CCIPC AHB):** the CCIPC is connected to the AMBA Bus with a Master (*Master AHB*) and Slave interface (*Slave AHB*);
- **Direct (CCIPC DIRECT):** the CCIPC periphery is directly connected to an external device.

5.2.1 LIBRARIES Directory

The **LIBRARIES** subdirectory contains the files used to adapt the CCIPC to different FPGA families.

Directory	File Name	Description
ACTEL_AX	ax_ram512x8.vhd (not furnished)	RAM 512x8 bit RAM model
ACTEL_PA3	pa3_ram512x8.vhd (not furnished)	RAM 512x8 bit RAM model
XILINX_VIRTEX	virtex_ram_512x8.vhd (not furnished)	RAM 512x8 bit RAM model
TECH_GEN	tech_generic.vhd	RAM 512x8 bit Generic RAM model

Tab. 5-2: CCIPC – FPGA technology files.

Currently the CCIPC supports only the Microsemi Accelerator and ProAsic3E families and Xilinx Virtex family. User has to generate the correspondent technology memory file. The following characteristics have to be used to generate a supported memory device:

	Accelerator	ProAsic3E	Virtex
Name	ax_ram512x8	pa3_ram512x8	virtex_ram_512x8
Type	Single Port RAM	Two Port RAM	Single Port RAM
Write Width	8	8	8
Read Width	8	8	8
Write Depth	512	512	512
Enable Pin	Not available	Not available	Yes
Write Enable	High	High	High
Read Enable	High	High	Not available
Pipeline	No	No	No
Reset	No	No	No
Clock	Single	Single	Single

Fig. 5-3 CCIPC RAM Model characteristics.

The generated file has to be included in the appropriate directory according to the target FPGA family.

5.2.1 TESTBENCH Directory

The **TESTBENCH** subdirectory contains the files needed to set up the CCIPC core simulation environment.

The files needed for simulation are reported in the following table:

Name	Description
CCIPC_tb.vhd	CCIPC Generic I/O interface testbench
CCIPC_AHBtb.vhd	CCIPC AHB interface testbench
CCIPC_DIRECTtb.vhd	CCIPC Direct interface testbench
parser.vhd	Emulator of CAN master and HOST interface
EEPROM8b.vhd	EEPROM memory model
am29f08d.vhd	

Name	Description
gen_utils.vhd	
conversion.vhd	
SRAM8b.vhd	SRAM memory model
CAN3MB: Can.vhd Interface.vhd singleshoot_true.vhd singleshoot_false.vhd	AUCAN3_8051 is an hdl model of CAN Controller interface for 8bit microcontroller and it's used to emulate a CAN master node.

Tab. 5-3: Testbench source files.

5.2.2 CAN_CORE Directory

The CCIPC core has been designed to embody the HurriCANE Core in its version 5.2.4. The source files are not directly included in the CCIPC database because subjected to different patent restrictions.

ESA distributes the HurriCANE_5.2.4 core under its specific license.

To insert the HurriCANE controller in the CCIPC database, the HurriCANE_5.2.4 source files have to be simply copied in the **CAN_CORE** directory.

The list of HurriCANE files is reported below:

- CANCore.vhd;
- CANCoreConfiguration.vhd;
- CANMessageStates.vhd;
- CANRx.vhd;
- CANTx.vhd
- CRCCalc.vhd;
- ErrFrameGen.vhd;
- ErrorCounters.vhd;
- StuffHandler.vhd;
- Synchro.

5.3 SIM Script Directory

The SIM_SCRIPT directory contains all the files that compose the IP core test procedures and the scripts needed to build new input stimuli for CCIPC simulation.

Name	Description
edac_rom.c	C-program to generate simulation ROM file from a ROM image of the OD
CCIPCTestWriter.pl	Perl script to generate stimuli file for the parser module
CCIPC_test.sh	Shell script to perform automatically predefined CCIPC test procedures

Tab. 5-4: Simulation scripts.

For each test procedure, these files are included:

- OD description in standard DCF format
- CCIPC ROM image file (see §11.1 for file content)
- VHDL constants configuration file (see §11.2 for file content)
- Input stimuli in .in format (see §8.2 for file format)
- Expected results file.

The VRP_TB sub-directory includes all the test procedures files and the full list of the CCIPC test procedure is contained in Tab. 5-5.

Test Name	Description
TP5113_boot	CCIPC boot-up message test
TP5123_nmt_s1	CCIPC NMT state transition test
TP5123_nmt_s2	
TP5123_nmt_s3	
TP5123_nmt_s4	
TP5133_NMT_filtering	
TP5143_hbt	Heartbeat transmission test
TP5213_asyncRPDO	Asynchronous RPDO test
TP5223_synchRPDO	Synchronous RPDO test
TP5313_synchTPDO	Synchronous RPDO test
TP5323_asyncTPDO	Asynchronous RPDO test
TP5413_sdo_expedited	SDO expedited test
TP5423_sdo_dwseg	SDO Download segmented test
TP5433_sdo_ulseg	SDO Upload segmented test
TP5443_DW_block	SDO Upload segmented test
TP5453_UL_block	SDO Upload segmented test
TP5462_AbortRx	Reception of SDO abort test
TP5462_ccs	SDO Error Decoding and Transmission of Abort message test
TP5462_IndexError	
TP5462_ReadOnly	
TP5462_SizeError	
TP5462_toggle	
TP5462_toggle_stimuli.txt (auxiliary file for testbench)	
TP5462_blksize_err	
TP5462_seq_err	
TP5513_hbc_tce	Heartbeat reception and Ttoggle expiration test
TP5523_Ctoggle	Ctoggle expiration and Bus switch test
TP5613_parallel#1	CCIPC handling of different task simultaneously test
TP5623_parallel#2	
TP5633_parallel#3	
TP5713_AMBA	AMBA interface test
TP5813_DIRECT	Direct interface test

Tab. 5-5: CCIPC test list.

The instructions to run all test procedures are provided in §8.3.

5.4 SYN Script Directory

The SYN_SCRIPT houses the files needed to synthesize the CCIPC.

The directory is furnished with the examples of synthesis scripts for Synplify Synthesizer for Microsemi Axcelerator and ProAsic3E families and for Xilinx Virtex family.

For each family the scripts to synthesise the CCIPC in its Generic I/O , AMBA and Direct interfaces are provided.

Two type of files are included:

- The project (.prj) file holds the reference to all the VHDL source files and includes the technology specific settings.
- The constraint (.sdc) file keeps the system clock and peripheral timing constraints for the CCIPC design.

The section §9.1.1 provides the user with a basic procedure for the core synthesis with Synplify Synthesizer Tool for Microsemi (RT)AX and Xilinx Virtex families.

5.5 FIT Script Directory

The FIT_SCRIPT houses the files needed to execute the Place and Route phase.

The directory is furnished with the examples of Microsemi Designer project for Axcelerator and ProAsic3E FPGA families and of Xilinx ISE for Virtex FPGA family.

For each family the scripts to fit the CCIPC in its Generic I/O , AMBA and Direct interfaces are provided.

The section §9.1.2 provides the user with a basic procedure for the core synthesis with Synplify Synthesizer Tool for Microsemi (RT)AX and Xilinx Virtex families.

5.6 CONFIG File Directory

The directory contains the files produced by the Configuration tool. The Configurator output files have the following extensions:

- .dfc, .eds: these are the standard CANopen Design Configuration and Electronic Datasheet File
- .txt: it is the ROM image of the Object Dictionary
- .vhd: it is a VHDL file that has to be copied in the SRC directory to substitute the CCIPC_conf.vhd file.

All the DCF, ROM image and VHDL files needed to perform the test listed in Tab. 5-5 are available in VRP_CF sub-directory.

5.7 CONFIG Tool Directory

The directory contains the executable file of the CCIPC Configuration tool (a Perl-Tk script) whose step by step user guide is described in §11.

6 CCIPC CONFIGURATION

This section introduces the CCIPC configuration parameters and explains their functions. The §11 guides the user in the graphical tool utilization to set the parameters values.

CCIPC configuration parameters are linked to the specific Core implementation. All the parameters are related to the CANOpen protocol and they are directly mapped in Object Dictionary.

The CCIPC configuration tool leads the user during the configuration phase of the CCIPC core. The list of items that the user can manage is shown in the table below.

Main group	Item	Description
CCIPC Set-Up and configuration	Node-ID	It indicates Node-ID assigned to the CCIPC slave and it is obtained from external port.
	Node count	It indicates the number of consecutive CANopen node that the core occupies. This option affects the maximum number of PDO supported.
	System frequency	It indicates the CCIPC working frequency.
	Bit-Rate	It indicates the CAN interface working bit rate.
	Host Interface	It indicates the host interface used by the CCIPC.
	Target Technology	It indicates the reference technology of the CCIPC core.
	SDO type	It allows defining supported SDO services
	Device Type	It describes the type of device and its functionality.
	Identity Object	It contains general informations about the device
	Synchronous Window length	If selected, indicates the length of time (unit = us) for synchronous PDO.
	Consumer Heartbeat time	It indicates the consumer heartbeat time (unit = ms) together with the expected Producer Node-ID
	Producer Heartbeat time	It indicates the producer heartbeat time (ms)
	SDO server	It indicates if SDO server is enabled or not
	Bus Manager parameters	It configures the Bus Manager parameters(Default Bus, Ttoggle,Ctoggle and Ntoggle)
Communication Objects configuration	TPDO parameters	It configures TPDO communication and mapping parameters
	RPDO parameters	It configures RPDO communication and mapping parameters
Profile Area configuration	Supported AOs	It allows configuring each single Application Objects

Tab. 6-1: List of configurable parameters.

In next paragraphs, a detailed description of each item is reported.

6.1 CCIPC Set-up and Configuration

The first step in the CCIPC configuration phase is to define the general characteristics of the node.

- **NodeID** – This field allows defining the CCIPC Node ID (Assigned from dedicated input port)
- **Node count** – In this field the number of consecutive CANOpen node occupied by the CCIPC. A maximum of 32 Nodes can be defined.

- **System frequency** – This field allows the user to define the CCIPC working frequency. The accepted values are in the range from 10MHz to 16 MHz.
- **Bit-Rate** – It allows selecting the working bit-rate of the CAN interface:
 - 1 Mbps
 - 500 Kbps
 - 250 Kbps
 - 125 Kbps

The tool calculates the BPR,PS1,PS2 and RSJ values according to the system frequency value. The user is in charge of verifying and, eventually, modifying these values.

- **Host interface** – the CCIPC interface towards the Host interface can be selected. Three options are defined:
 - Generic I/O
 - Direct
 - AMBA Bus

The choice between the three options has a direct impact on the CCIPC configuration, in particular on Object Dictionary implementation. Using a direct interface the OD structure is fully implemented exploiting internal FPGA resources, limiting the core communication capabilities. The other two options, using an external memory device to implement part of the OD, allows saving more resources to store Application Objects. (see §10 for more detail on CCIPC memory organization).

- **Target technology** – the target technology for the CCIPC implementation can be selected. Four options are defined:

- Microsemi Axcelerator
- Microsemi ProAsic
- Xilinx Virtex
- Generic Technology

The user has to select the SDO services supported by CCIPC:

SDO Type – Three options are defined:

- Expedited: only SDO expedited;
- Segmented: Expedited and Segmented SDO;
- Block: Expedited and Block SDO;

6.1.1 Synchronous Window Length

Using the dedicated field is possible to enable or disable the CCIPC synchronous window length feature. If enabled, a value different from zero has to be assigned. The synchronous window length time has a granularity of microseconds.

6.1.2 Consumer Heartbeat Time

This field allows specifying the expected Heartbeat cycle time, expressed in millisecond granularity, and the Node-ID of the expected Heartbeat producer.

6.1.3 Producer Heartbeat Time

This field allows specifying the cycle time of the heartbeat object, expressed with millisecond granularity.

6.1.4 SDO Server Configuration

The CCIPC implements a SDO server features. The only parameters that define an SDO server are:

- COB-ID client -> server
- COB-ID server -> client

Since the CCIPC uses the pre-defined connection set these parameters are automatically generated by the core, depending on CCIPC NodeID assigned.

As explained in §4.6.2 the CCIPC can be configured according to SDO services supported:

- **Expedited:** CCIPC supports only SDO Expedited service;
- **Segmented:** CCIPC supports SDO Expedited and Segmented services;
- **Block:** CCIPC supports SDO Expedited and Block service.

6.1.4.1 SDO Implementation: CCIPC FSMs

The implementation of these services are performed by the specific Finite State Machine modules that are available in the **SRC** directory.

All the FSMs are executes the following “macro” operations:

- Object Dictionary Initialisation;
- RPDO elaboration;
- TPDO elaboration;
- SDO elaboration.

The FSMs are implemented exploiting only combinational logic and they work as “big multiplexers” returning the execution code according to the input address. Each “macro” operation is implemented as a sequence of Logic, Arithmetic and Memory operations for:

- Registers loading with constants
- Operands moving among different registers
- Arithmetic operations
- Logic compare and bit check
- Read/write memory operations

Next table summarizes the supported serviced for each FSM:

FSM	Initialization	RPDO		TPDO		SDO		
		Synch	Asynch	Synch	Asynch	Expedited	Segmented	Block
Expedited	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Segmented	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Block	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Direct	No	Yes	Yes	Yes	Yes	Yes	No	No

Tab. 6-2: CCIPC FSMs vs Implemented CANopen services.

The Direct FSM is a particular FSM used when the CCIPC is configured with Direct Interface. Since in this configuration CCIPC is directly connected to an external device and no external memories are required, all the Object Dictionary parameters (RPDO, TPDO, Application Objects) are implemented exploiting internal FPGA resources, and no initialization phase is needed.

6.1.5 Bus Manager Parameters

The parameters available in this section, together with the Heartbeat Consumer time one, define the CCIPC behaviour to manage the CAN Bus switching between Default and Redundant Bus. In particular the following parameters can be set:

- **Ttoggle** counter that defines the number of Heartbeat event that can occur before Bus switching;
- **Ntoggle** counter that defines the maximum number of bus switching.

6.2 Communication Objects Configuration

This section of the configuration tool allows defining the CCIPC communication parameters for the Receive and Transmit PDOs services.

6.2.1 Receive PDO

The RPDO section allows configuring the RPDO communication and mapping parameters. The CCIPC foresees up to 4 RPDOs for each node occupied by the CCIPC; in this way the user is able to define a number of RPDO included in the range [1:4*NodeID count].

For each RPDO supported, the following parameters are configurable:

- RPDO exist or not
- RPDO communication type:
 - Synchronous :
 - Asynchronous
- Mapped Application Objects

6.2.2 Transmit PDO

The TPDO section allows configuring the TPDO communication and mapping parameters. The CCIPC foresees up to 4 TPDOs for each node occupied by the CCIPC; in this way the user is able to define a number of TPDO included in the range [1:4*NodeID count].

For each TPDO supported, the following communication parameters are configurable:

- TPDO exist or not
- TPDO communication type:
 - Synchronous transfer type in the range from 1 to 240
 - Asynchronous. In this case other two parameters can be defined:
 - Inhibit timer.
 - Event timer
- Mapped Application Objects

Since CCIPC supports pre-defined connection set, the TPDO COB-ID is automatically programmed inside the CCIPC, depending on the assigned to Node-ID.

The Event Timer is defined as multiple of 1ms in accordance with CANOpen standard.

Even if CANopen standard defines 100us granularity for the inhibit time, the effective time granularity supported by CCIPC is 1ms (Event and Inhibit timers are managed by a unique task only one time every 1ms). The time intervals in the inhibit timer are expressed in 100us time unit, but only values that are integer multiple of 10 (10 x 100us = 1ms) are legal and exactly managed by CCIPC.

6.3 Application Object Configuration

The application specific data and structures have to be defined as application objects according to the CANOpen protocol and the CCIPC restrictions:

- Only Record, VAR and Array objects are supported
- Only U32, U16 and U8 types are supported
- Implicit Array initialization (DCF Compact Storage option) is still not supported

As additional option the user can distinguish between read-write and read-only objects mapping them in separated areas:

- Read-Write objects: index range 6000h-6FFFh.
- Read-Only objects: index range 7000h-7FFFh.

7 INTERFACE

This section contains information about the CCIPC interface, with a detailed description of its periphery signals and also with an overview of the different interfaces available to allow user to connect to CCIPC.

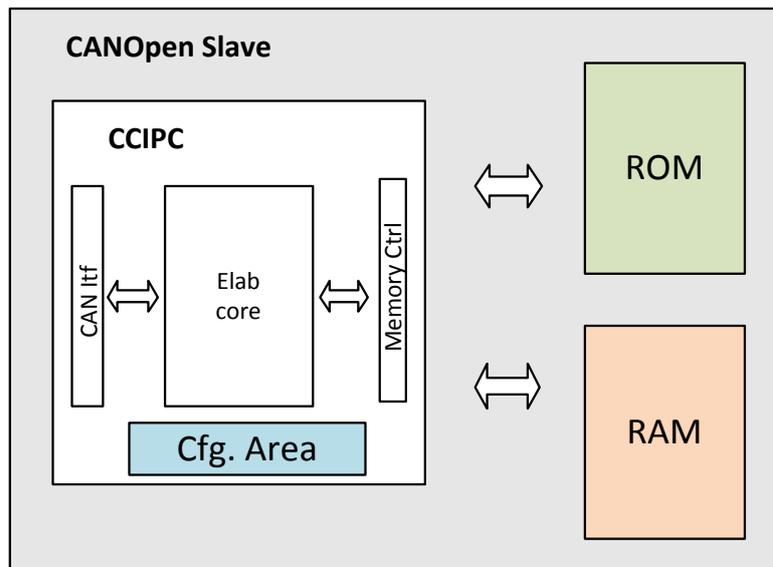


Fig. 7-1: CANOpen slave node.

In order to work as a CANOpen slave node according to the Network Management “Reset Node” and “Reset Communication” services, CCIPC needs to access two external memory area here after simply called ROM and RAM.

- **ROM** area is assigned to store the default values for TPDO/RPDO Communication/Mapping parameters and for Application Objects that have to be loaded after power-on or when the Reset node / Reset Communication requests are received.

Depending on the specific instance this area can be realized with different solutions:

- FPGA internal implementation as a LUT circuit (for example in case of 1-2 node with application objects limited to few hundreds of bytes),
 - FPGA RAM macro-cell. Internal A dedicated FPGA memory has to be preloaded before releasing the CCIPC Rst_n signal (e.g. the case of an external microprocessor or a serial link connected to a remote server/memory devices),
 - External EEPROM memory. This is a simple (and costly) solution achievable with an external non-volatile memory
- **RAM** area is assigned to keep TPDO and RPDO Mapping Parameters and Application object data. This area is shared between the CCIPC core and the External Device. Depending on the number Node supported and Application Object defined, this area can be implemented exploiting the internal FPGA macro-cells or as part of an external RAM device.

The CCIPC includes also an additional memory area, called **Configuration Area** that collects all the CCIPC configuration and status registers.

CCIPC distinguishes between ROM and RAM areas with the two most significant bit of its address bus according to the following map table Tab. 7-1. The third row is assigned to the CCIPC configuration space that if requested by a SDO service requires an external loop-back on the Core Data and Configuration Ports.

Address A[21:20]	Memory AREA
11	ROM
10	RAM
01	CFG

Tab. 7-1: Memory Areas map.

7.1 CCIPC Periphery Signals

The detailed pin-out of the CCIPC interface is described below:

Signals	Dir	Description
Clk	In	System clock
Rst_n	In	System reset
NodeId[6:0]	In	CCIPC Node ID
Mask[2:0]	In	CCIPC Mask size
BusSel	Out	Can Bus selection
CanTx	Out	Can tx line
CanRx	In	Can rx line
D[39:0]	In	Data Read Bus
GNT	In	Bus ownership grant
READY	In	Operation done flag
REQ	Out	Bus ownership request
A[21:0]	Out	Address Bus
Q[39:0]	Out	Data write BUS
WR	Out	Data write command
RD	Out	Data read Command
LOCK	Out	Bus Lock request
Cfg_A[19:0]	In	Cfg address
Cfg_D[31:0]	In	Cfg Input Bus
Cfg_cs	In	Cfg select
Cfg_rd	In	Cfg read command
Cfg_wr	In	Cfg write command
Cfg_Q[31:0]	Out	Cfg Output Bus
IRQ[2:0]	Out	Interrupt lines. It includes IRQ error lines
Synch_rx	Out	Strobe to signal synch message reception
rst_aos	Out	Reset Application Objects. Used only when Direct Interface is selected.

Tab. 7-2: CCIPC periphery.

7.1.1 Memory Interface

The following (Fig. 7-2, Fig. 7-3 and Fig. 7-4) figures show the timing of the CCIPC interface signals during read, write and read-modify operations. The read-modify is needed to keep EDAC consistency when a 32 bit word has to be partially updated for a U8 or U16 write request.

The memory arbiter (*arbiter.vhd*) and AHB master/slave modules (*AHB_mst.vhd*, *AHB_sslv.vhd*) included in the CCIPC data-base work as bridge between CCIPC core and the SRAM/EEPROM components or AMBA AHB bus.

The interface timing charts are here described to allow users to develop new application specific interface for CCIPC.

On the CCIPC interface all the operations are word oriented in order to assure the correct management of the EDAC protection mechanism (1 EDAC byte every 4 data bytes).

The CCIPC works in Little Endian mode: the least significant address contains the least significant byte of a word and the most significant address contains the most significant byte of the word.

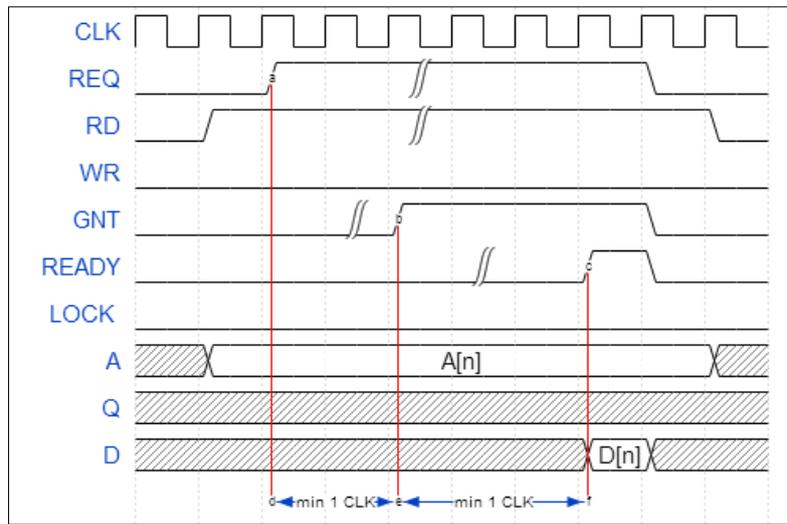


Fig. 7-2: Single Read.

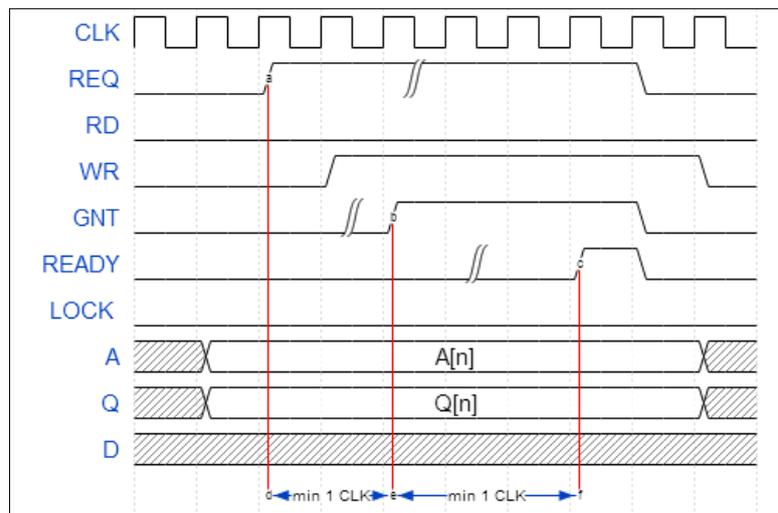


Fig. 7-3: Single write.

Memory operation starts when CCIPC set REQ signal high. At the next clock cycle the RD, WR and A and Q (during write cycle) are stable at the interface.

CCIPC waits for GNT signal high as bus granted feedback. Afterwards data passing takes place only when CCIPC sample the input READY signal high. CCIPC samples the D bus value during read, or assumes the Q bus sampled during write, when READY flag is high in condition of REQ and GNT signal asserted.

CCIPC is also in charge of performing read modify operations when less than four byte have to be written in the memory.

In this case the LOCK signal is used to maintain the control of the memory device after end of read operation. The following write operation is performed without realising the LOCK signal. The timing chart of this operation is shown in Fig. 7-4:

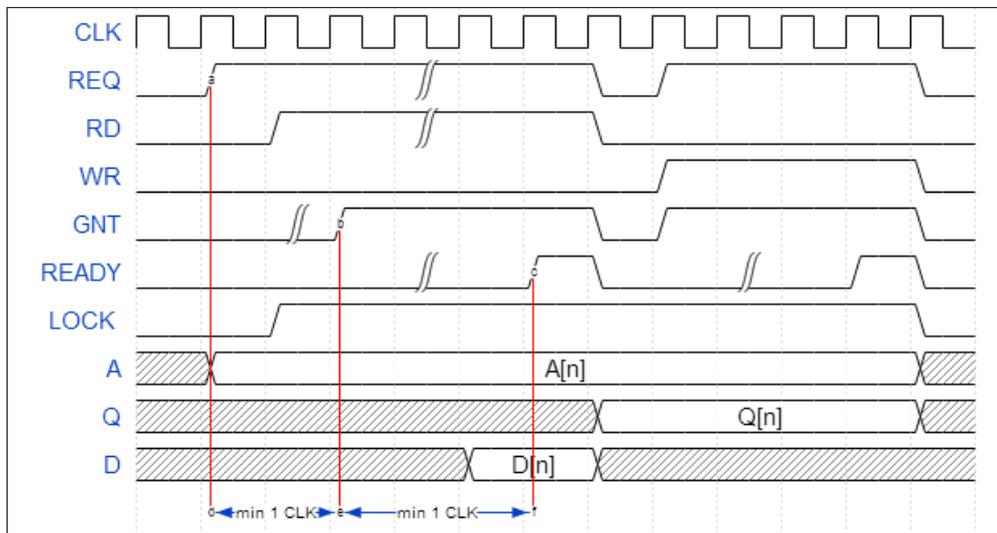


Fig. 7-4: Read-Modify.

7.1.2 Configuration Area Interface

The Configuration Area signals are used to access the CCIPC internal registers. The following timing charts describe the behaviour of these signals during read and write access.

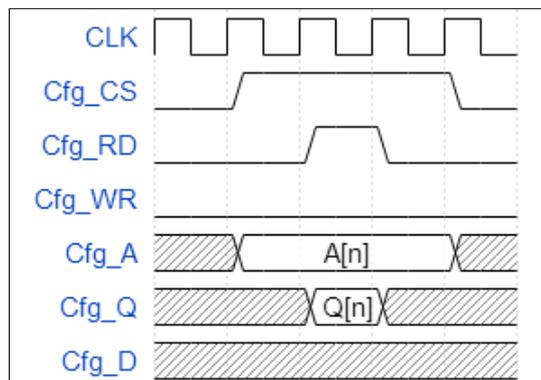


Fig. 7-5: Configuration Area access : Single Read.

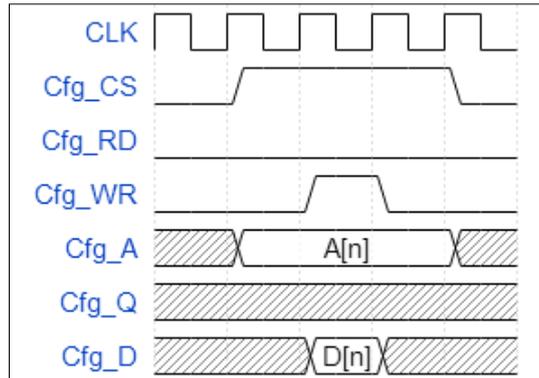


Fig. 7-6: Configuration Area access: Single write.

Accesses on Configuration Area start when *Cfg_cs* signal is set high and they are one clock cycle long. The *Cfg_rd* and *Cfg_wr* signals determine the type of operation. If a read access is requested the read data is immediately available in *Cfg_Q* signal. If a write access is requested the value of *Cfg_D* signal is written in the register.

7.1.3 IRQ & External Trigger

The CCIPC interface is able to handle three IRQ lines:

- IRQ[0]: Transfer IRQ (TRX_IRQ)
- IRQ[1]: Synch elaboration completed (SYNCH_IRQ)
- IRQ[2]: Error IRQ (ERR_IRQ)

The TRX_IRQ is used to signal the following conditions:

- reception of Initiate SDO transfer (Segmented and Block only)
- successful elaboration of an SDO transfer
- successful elaboration of asynchronous T/R PDO
- NMT state transition

The SYNCH_IRQ is used to signal the successfully completion of the Synch processing.

The ERR_IRQ line is used to inform the Host that an error has been detected (EDAC error, CAN bus Error, SDO and PDO errors, Bus switch)

These signals have to be cleared by user through dedicated registers described in §10.2

An additional line is used to inform the Host on reception of the Synch message. This bit is active for a single clock cycle.

7.1.4 Node-ID Acquisition and Modification

The CCIPC acquires information on base Node-ID and number of Node supported using a dedicated external signals (NodeID and Mask signals in Tab. 7-2) in the first eight clock cycles after the reset phase. In Tab. 7-3 and Fig. 7-7 the behaviour of this interface is explained.

Name	Description
NodeID[6:0]	It defines the Base Node-Id assigned to the CCIPC slave
Mask[2:0]	It defines the number of supported Node using the following scheme: "000" – 1 Node "001" – 2 Nodes "010" – 4 Nodes "011" – 8 Nodes "100" – 16 Nodes

Name	Description
	"101" – 32 Nodes

Tab. 7-3: Node-Id interface description.

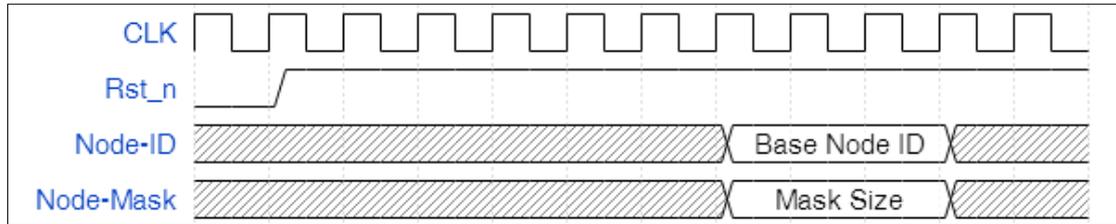


Fig. 7-7: Acquisition of Node-ID information.

FPGA samples Node-ID and Mask values between the 7th and 9th clock cycle after reset. Then user can re-assign corresponding pins to different custom functions without affecting the CCIPC functionalities. CCIPC utilizes this information to perform filtering function of the incoming CANOpen messages and to correctly set the COB-ID of message to be sent. In this way, the Node-ID can be modified without requiring a new FPGA programming, but only a reset procedure in order to allow CCIPC to re-sample the new Node information. Before resetting the core, the user must update the content of the ROM area in order to keep aligned the OD structure with the new Node-ID. The following table reports the association between the value of "Mask" input port and the "ID MASK" field available in the 2002h entry of the Configuration area.

Node count	Mask	ID Mask
1	0	111 1111
2	1	111 1110
4	2	111 1100
8	3	111 1000
16	4	111 0000
32	5	110 0000

Tab. 7-4: Node-Mask vs ID_mask explanation.

7.1.5 CAN Bus Selection

The CCIPC manages the redundancy algorithm and bus multiplexing, that is: CCIPC includes a single CAN controller capable to handle two physical busses.

The **BusSel** output signal select which of the two CAN busses is logically connected to the core according to the Redundancy algorithm.

Since the BusSel output signal is stable to '0' or to '1' according to the active bus selected by the redundancy manager, it can be used as control signal for the multiplexer, which "selects" the active CAN transceiver.

7.1.6 Reset Distribution

The CCIPC is provided by an active-low asynchronous Power-On Reset (*rst_n*). This reset allows initialize all the sequential logic available in the CCIPC in an asynchronous way.

Together with this global reset, the CCIPC has three additional signals that act as synchronous reset to re-establish the default values of CCIPC internal registers. These signals are associated to following conditions:

- **Bus switch** : the whole CCIPC core is reset, including the CAN core;

- **Reset Application** : the Application Objects stored in the External Memory area and the registers defined in the Application Objects area are reset;
- **Reset Communication** : the RPDO and TPDO Communication and Mapping parameter and registers defined in the Communication Objects area are reset.

7.2 Host Interfaces

The CCIPC can be connected to an host device through three different interfaces:

- **Generic I/O**;
- **AMBA**;
- **Direct**.

The **Generic I/O** interface is a “memory-like” interface and it requires an auxiliary arbiter module to manage the communication between CCIPC and Host. This arbiter is in charge of sharing the memory bus between Host and CCIPC.

In the **AMBA** interface the CCIPC is connected to an AHB bus through AHB Master and Slave modules. The Master module allows CCIPC to access external memory devices, while the Slave module allows host to access CCIPC registers.

The **Direct** interface is a specific interface used for simple CCIPC implementation that does not require external memories because the whole Object Dictionary is hardwired. In this interface the Application Objects are directly available at system peripheral as input or Output

7.2.1 Generic Interface

The Generic interface architecture is depicted in Fig. 7-8. In order to make the connection of the Core to external device easy the CCIPC core is distributed with an ancillary Arbiter module that manages the sharing of ROM and RAM areas with the Host.

The actual CCIPC Generic interface periphery includes the signals listed in Tab. 7-5

Signals	Dir	Description
Clk	In	System clock
Rst_n	In	System reset
NodeId[6:0]	In	CCIPC Node ID
Mask[2:0]	In	CCIPC Mask size
BusSel	Out	Can Bus selection
CanTx	Out	Can tx line
CanRx	In	Can rx line
IRQ[2:0]	Out	Interrupt lines. It includes IRQ error lines
Synch_rx	Out	Strobe to signal synch message reception
D[39:0]	In	Host device Write Data BUS
GNT	Out	Host device Bus ownership grant
READY	Out	Host device Operation done flag
REQ	In	Host device Bus ownership request
A[20:0]	In	Host device Address Bus
Q[39:0]	Out	Host device Read Data Bus
WR	In	Host device write command
RD	In	Host device read Command
LOCK	In	Host device Bus Lock request

Signals	Dir	Description
Mem_Q[7:0]	In	Memory device Read Data Bus
Mem_D[7:0]	Out	Memory device Write Data Bus
out_en	Out	Output enable
Mem_A[19:0]	Out	Memory device Address Bus
Mem_wr_n	Out	Memory device write enable
Mem_rd_n	Out	Memory device read enable
Mem_cs_n	Out	Memory device chip select

Tab. 7-5: Generic I/O peripheral.

The arbiter has been designed to manage two narrow 8 bit memories de/serializing parallel data buses (32 bit + EDAC) for CCIPC and Host Device. EDAC address is obtained with the complement of the Data Word Address to occupy the highest 20% memory locations.

The arbiter code provide the CCIPC user with an example for the management of the CCIPC Generic interface and its higher address bit to generate the three select signals for the RAM, ROM and CFG spaces.

The arbiter code fits a single connection scheme and it may easily modified and more in general used as starting point, to insert CCIPC in its specific application.

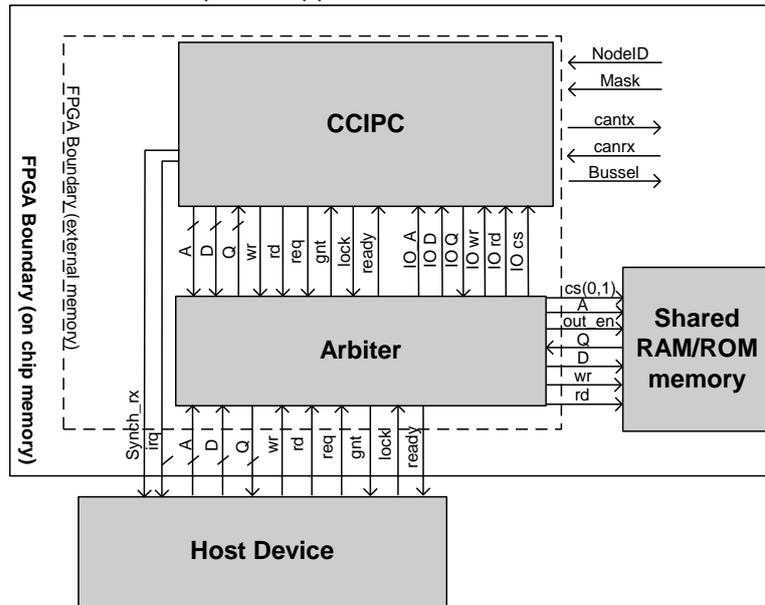


Fig. 7-8: Arbiter module connection.

7.2.2 Direct I/O Interface

The Direct Interface is used for very simple CCIPC implementations in which Application Objects could be directly available at system peripheral as input or Output.

The pin-out of the Direct interface top entity is represented in Tab. 7-6:

Signals	Dir	Description
Clk	In	System clock
Rst_n	In	System reset

Signals	Dir	Description
NodeID[6:0]	In	CCIPC Node ID
Mask[2:0]	In	CCIPC Mask size
BusSel	Out	Can Bus selection
CanTx	Out	Can tx line
CanRx	In	Can rx line
IRQ[2:0]	Out	Interrupt lines. It includes IRQ error lines
Synch_rx	Out	Strobe to signal synch message reception
HI_trig	In	External trigger request
HI_trig_val[7:0]	In	Asynchronous triggers
User defined I/O	In/Out	Application Objects mapped in the top level peripheral

Tab. 7-6: Direct Interface peripheral.

An example of Direct I/O interface implementation is shown in Fig. 7-9. In this solution the Application Objects are directly mapped into FPGA flip-flop and pins.

The simplest example of Direct interface may be a GPIO barrier where the pins values and direction controls are mapped into two OD entries and held by FPGA registers. More complex interfaces for Sensors/Actuators (ADC/DAC) may be realized with a further VHDL editing and customisation by the user. In Fig. 7-9 a simple OD composed of 5 flip-flop barriers directly connected to the FPGA pins is presented.

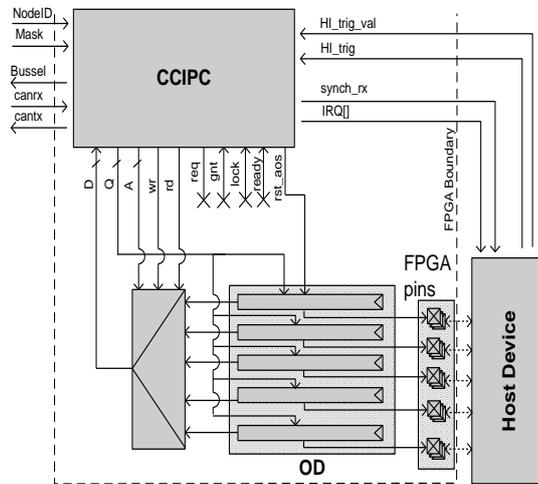


Fig. 7-9: Direct interface.

The HI_trig and HI_trig_val signals are used to implement the TPDO event driven functionality. When HI_trig signal goes high, the CCIPC uses the HI_trig_val as enable to sample the number of the TPDO to be elaborated on the actual CLK rising edge as shown in Fig. 7-10 (in case of Hi_trig/Hi_trig_val sampled on the FPGA periphery the Tsetup and Thold timings are technology dependent). Tab. 7-7 shows an example of the correspondence between TPDO number and HI_trig_val in a single node CCIPC instantiation.

HI_trig_val[1..0]	TPDO number	OD index
"00"	0	1800h
"01"	1	1801h
"10"	2	1802h
"11"	3	1803h

Tab. 7-7: HI_trig_val definition.

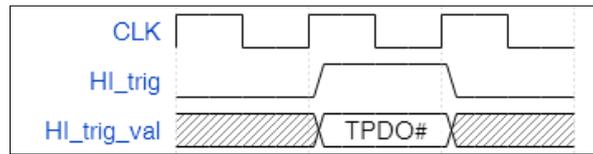


Fig. 7-10: Event driven TPDO timing.

7.2.3 AMBA Bus Interface

The AMBA Bus interface is composed of an AMBA AHB master module used to request read or write operation on the external memory device and an AMBA AHB slave module used to access the CCIPC configuration & status registers. The AHB VHDL models provided with CCIPC are simplified, **not fully standard compliant**, version of the AMBA AHB interfaces.

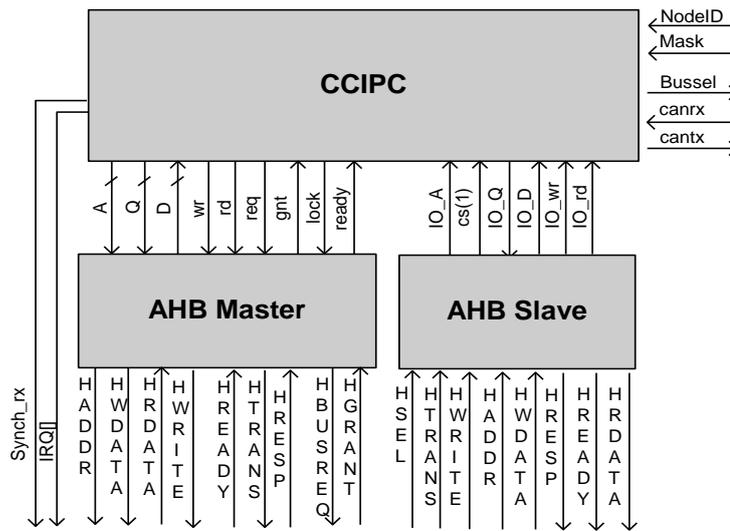


Fig. 7-11: AMBA bus interface.

The AMBA Bus top entity pin out is reported in Tab. 7-8

Signals	Dir	Description
clk	I	Master clock
rst_n	I	Global reset
NodeID[6:0]	I	Slave Node Id
Mask[2:0]	I	Slave Node Mask
canrx	I	CAN bus Rx line
cantx	O	CAN bus Tx line
BusSel	O	CAN Bus selection

Signals	Dir	Description
IRQ[2:0]	O	IRQ lines
synch_rx	O	Synch message reception strobe
M_HGRANT	In	Arbiter grant
M_HREADY	In	Ready flag
M_HRESP	In	Transfer response
M_HRDATA[31:0]	In	Read data bus
M_HBUSREQ	Out	Bus request
M_HTRANS[1:0]	Out	Transfer status
M_HWRITE	Out	Operation type
M_HADDR[31..0]	Out	Address bus
M_HWDATA[31:0]	Out	Write data Bus
M_HSIZE[2:0]	Out	Transfer Data type
M_HBURST[2:0]	Out	Transfer type
M_HLOCK	Out	Transfer Lock
S_HSEL	In	Selection signal
S_HTRANS[1..0]	In	Transfer status
S_HWRITE	In	Operation type
S_HADDR[20..0]	In	Address bus
S_HWDATA[31..0]	In	Write data Bus
S_HRESP[1:0]	Out	Transfer response
S_HREADY	Out	Ready flag
S_HRDATA[31..0]	Out	Read data Bus
S_HSIZE[2:0]	In	Transfer Data type
S_HBURST[2:0]	In	Transfer type
Generics		
AHB_ROM_OFFSET[31:20]	---	AMBA Bus ROM area
AHB_RAM_OFFSET[31:20]	---	AMBA Bus RAM area
AHB_SLV_OFFSET[31:10]	---	AMBA Bus Configuration Area

Tab. 7-8: AMBA Bus signal.

The selected interfaces works with 32-bit data AHB bus and it is able to manage up to 1 Mbyte of addressable space:

- HWDATA[31:0]
- HRDATA[31:0]
- HADDR[31:0]

The user has to declare the AMBA Bus areas assignment setting the following VHDL generics :

- AHB_ROM_OFFSET[31:20]: 1Mbyte ROM size;
- AHB_RAM_OFFSET[31:20]: 1Mbyte RAM size;
- AHB_SLV_OFFSET[31:10]: 1Kbyte Configuration space.

A simplified version of the AHB interface is supported by CCIPC; Tab. 7-9 lists the granted values of the principal AHB transfer signals.

CCIPC slave interface supports only sequential transfer type (HTRANS="11") of 32 bit size (HSIZE = "010") with incrementing burst (HBURST="001").

CCIPC master interface communicates using only sequential transfer type (HTRANS="11") of 32 bit size (HSIZE = "010") with incrementing burst (HBURST="001") and it not accepts ERROR, RETRY or SPLIT response from the addressed slave.

The M_HLOCK signal is always asserted during transfer to prevent grant de-assertion by arbiter (such condition is not managed by the simplified AHB master). With exception of the Read-Modify operation in

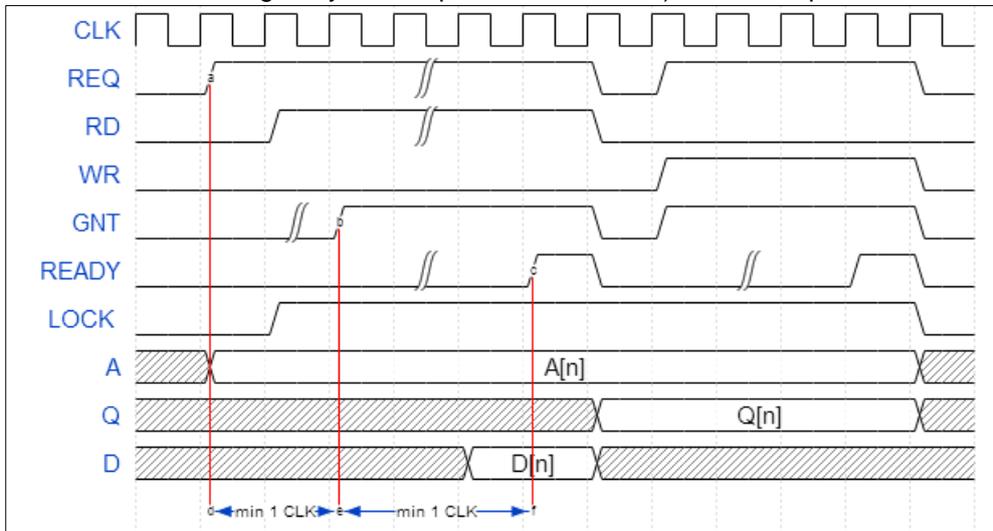


Fig. 7-4 (single write after single read) CCIPC never joins two AHB bus accesses so that it guaranties the insertion of several IDLE cycles between two subsequent locked AHB transactions. The retry or split options are not supported.

AHB signals	Value	Supported
HTRANS[1:0]	00 – IDLE	yes
	01 – BUSY	yes
	10 – NONSEQ	yes
	11 – SEQ	yes
HBURST[2:0]	000 – SINGLE	no
	001 – INCR	yes
	010 – WRAP4	no
	011 – INCR4	no
	100 – WRAP8	no
	101 – INCR8	no
	110 – WRAP16	no
111 – INCR16	no	
HSIZE[2:0]	000 – 8	no
	001 – 16	no
	010 – 32	yes
	011 – 64	no
	100 – 128	no
	101 – 256	no
	110 – 512	no
111 – 1024	no	
HRESP[1:0]	00 – OKAY	yes
	01 – ERROR	no
	10 – RETRY	no
	11 – SPLIT	no

Tab. 7-9: Supported AHB signals value.

8 SIMULATION ENVIRONMENT

This section gives the instructions to set-up and run simulation of the generated CCIPC instance. The simulation set-up procedure is based on the “make” tool and all the tools specific commands are included in the “Makefile” script.

Compilation/analysis and elaboration command for the VHDL files are defined as the VHDL_Com/Elab Variables:

- \$(VHDL_Com) – VHDL compilation command
- \$(VHDL_Elab) – Elaboration command

```
W = worklib

# - VHDL compilation command
VHDL_Comp = ncvhdl -V93 -WORK
# - elaboration command
VHDL_Elab = ncelab -ACCESS +rw
```

The \$(W) variable indicates the output directory for compiled files and tags.

The “setup” command section is assigned to build the tool specific Set-up environment. The following box shows the example of the *Cadence NCSim* simulator, which requires the cds.lib, hdl.var to link “worklib” and “standard” libraries.

```
setup:
  mkdir ./worklib
  mkdir ./axcelerator
  mkdir ./proasic3e
  mkdir ./xilinxcorelib
  echo `DEFINE worklib ./worklib` > cds.lib
  echo `DEFINE ieee $(CDS_ROOT)/tools/inca/files/IEEE` >> cds.lib
  echo `DEFINE std $(CDS_ROOT)/tools/inca/files/STD` >> cds.lib
  echo `DEFINE synopsys / $(CDS_ROOT)/tools/inca/files/SYNOPSYS` >> cds.lib
  echo `DEFINE LIB_MAP (.=> worklib, + => worklib )` > hdl.var
  echo `DEFINE WORK worklib` >> hdl.var
  echo ``timescale 1ns/10ps` > time.v
```

Relationships between the design sub-module is explicitly indicated in the dependency section of the Makefile. For example the notation of the following Box indicates that test-bench VHDL entity CCIPC_tb (Top of simulation hierarchy) depends on the CCIPC.vhd, arbiter.vhd and parser.vhd modules that have to be analysed before it.

```
$(W)/CCIPC_tb.vhmdl : \
  $(W)/CCIPC.vhmdl \
  $(W)/arbiter.vhmdl \
  $(W)/parser.vhmdl
```

When a new design is developed on the CCIPC core these Makefile lines have to be edited to insert the new custom hierarchy.

A specific file (*Makefile.mem*) is included in the Makefile and called during the compilation process defining which technology model has to be applied for CCIPC simulation.

```
M0 = CCIPC_DB/SRC
M1 = $(M0)/LIBRARIES
M2 = $(M1)/TECH_GEN

MPATH = .:$(M0):$(M1):$(M2)
```

```
$(W)/mem_tech.vhmdl : \  
    $(W)/CCIPCconf.vhmdl \  
    $(W)/tech_generic.vhmdl
```

This Makefile includes only the instruction to compile the Generic technology model. User has to add all the files and their dependencies needed to compile and simulate with the specific technology.

NCSim tool example

The simple steps to set-up a new simulation directory and start simulating CCIPC with the Cadence Ncsim tool are explained in this example.

Start creating the new (YourSIM) simulation directory:

```
>mkdir YourSIM  
>cd YourSIM
```

Create a symbolic link to the main directory of CCIPC DataBase directory with CCIPC_DB name:

```
>ln -s /YourPath/CCIPC_DB CCIPC_DB
```

Create a symbolic link to the to the Makefile included in the SRC directory:

```
>ln -s CCIPC_DB/SRC/Makefile .  
>ln -s CCIPC_DB/SRC/Makefile.mem .
```

Set-up the simulator path in the \$CDS_ROOT environment variable

```
>setenv CDS_ROOT YOUR_SIMULATOR_PATH
```

Launch the set-up command

```
> make setup
```

Create the following simulation auxiliary files:

```
> touch dcf.dat  
> touch ram.dat  
> touch none  
> touch stimuli.txt
```

Create the CCIPC_FSM.vhd symbolic link to the correct CCIPC FSM file. For example if SDO tpe selected is Segmented use the following command:

```
>ln -s CCIPC_DB/SRC/CCIPC_FSM_SEG.vhd CCIPC_FSM.vhd
```

Now, launch the make command to generate simulation file:

```
> make
```

After process elaboration, to simulate the system launch the following command:

```
> ncsim CCIPC_tb:A -gui &
```

Other tools

If another tool is utilised, the \$(VHDL_Comp), and \$(VHDL_Elab) variables and the tool Set-up section have to be modified inserting the specific tool commands before proceeding with the compilation process.

8.1 CCIPC Core Test-Bench

A dedicated CCIPC core test-bench has been used to test the CANopen functionalities supported by the CCIPC slave node. All the three interfaces (*Generic I/O*, *AMBA*, *Direct*) and all the four FSMs (*Expedited*, *Segmented*, *Block* and *Direct*) have been tested.

The Fig. 8-1 shows the test-bench architecture implemented when the CCIPC is used with *Generic I/O interface*.

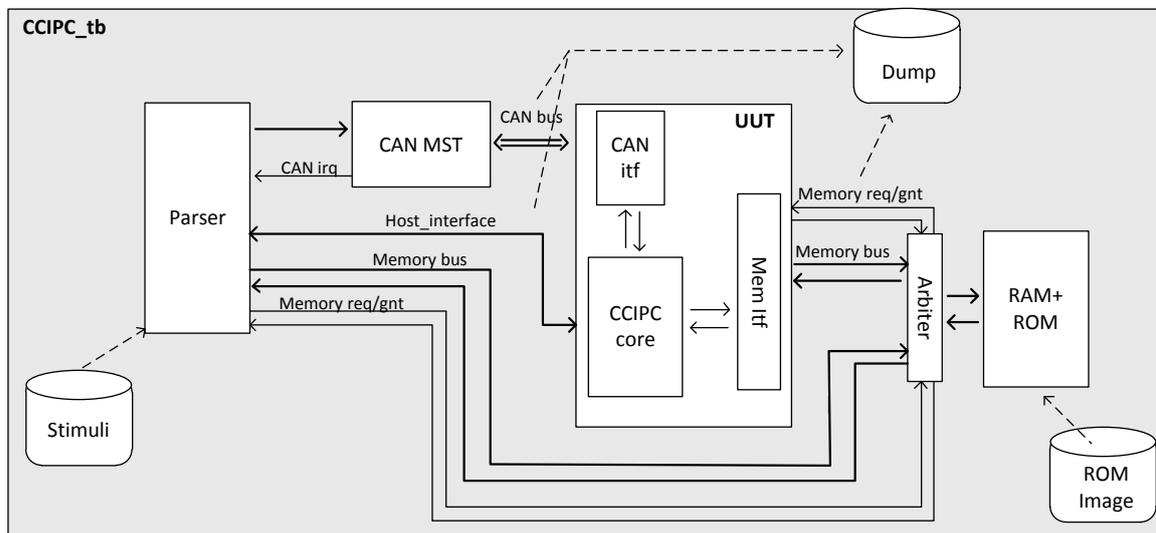


Fig. 8-1: CCIPC test-bench.

The test-bench module (called *CCIPC_tb*) consists of:

- **Parser module:** it traduces input stream from DCF or Stimuli files in specific requests on CAN MST or UUT Memory interface emulating the external device. It controls *CAN MST* and emulates host device in the simulation test
- **CAN MST:** it implements a CAN controller module based on micro-controller interface that allows exchanging CAN data frame with the UUT. This module works with an input frequency of 16MHz and with 1Mbps of CAN bit rate
- **UUT:** it is the Unit under test and it implements the CCIPC slave node.
- **Memory Arbiter:** it handles sharing of memories between the host device (emulated with static stimuli by parser) and the UUT.
- **RAM:** it is the shared memory module accessible both form CCIPC and the host device.
- **ROM:** it is the memory module that stores the parameters for CCIPC initialisation.

The files used during the test procedure are:

- **Stimuli:** it contains the test instruction used to drive the CAN-MST interface and also to perform access on RAM module and in the Configuration Area emulating the behaviour of a generic external device interface
- **Test Log:** this task logs any transition on UUT interface.

8.2 Input Stimuli Format

The input stimuli for the CANOpen testbench are built as sequence of CAN messages and memory accesses on its host interface. Both of them are written in the test files using the intuitive format described below in Tab. 8-1.

Function	Description
can read_msg	CAN master read message received (*)
can clear_irq	CAN master clear message received IRQ (**)
can set_irq	CAN master enable message received IRQ (***)
start_node \$NodeID	CAN Master send a "Enter Operational state" request to \$NodeID
stop_node \$NodeID	CAN Master send a "Stop Node" request to \$NodeID
rst_comm \$NodeID	CAN Master send a "Reset Communication" request to \$NodeID
rst_node \$NodeID	CAN Master send a "Reset Application" request to \$NodeID
enter_preop \$NodeID	CAN Master send a "Enter Pre-Operational state" request to \$NodeID
Heartbeat \$NodeID	CAN master send an Heartbeat message. \$NodeID = Master Node ID
synch	CAN Master send synch object
rpdo \$COBID \$DATA	CAN Master send RPDO message \$COB-ID – RPDO COB-ID \$DATA – RPDO data [max 8 byte]
sdo \$ID \$TYPE \$OPT	CAN Master sends SDO request: \$ID – ID of SDO client \$TYPE – SDO type: <ul style="list-style-type: none"> • InitDL \$expedited \$Index \$subIndex \$data-size <ul style="list-style-type: none"> ○ \$expedited – expedited(1) or segmente(0) ○ \$Index – Pointed index ○ \$subIndex – pointed sub-index ○ \$data-size – data if expedited, size if segmented • Download \$cont \$data <ul style="list-style-type: none"> ○ \$cont – indicates if it is the last segment ○ \$data – stores data objects • InitUL \$Index \$subIndex <ul style="list-style-type: none"> ○ \$Index – Pointed index ○ \$subIndex – pointed sub-index • Upload • BlkIDL \$Index \$subIndex \$size <ul style="list-style-type: none"> ○ \$Index – Pointed index ○ \$subIndex – pointed sub-index ○ \$size –Block download size • BlkDL \$cflag \$byte_num <ul style="list-style-type: none"> ○ \$cflag – status of completed bit ○ \$byte_num – number of bytes to download • BlkEndDW • BlkIUL \$Index \$subIndex \$Blksize <ul style="list-style-type: none"> ○ \$Index – Pointed index ○ \$subIndex – pointed sub-index ○ \$Blksize – Block size • BlkUL \$ackseq \$Blksize <ul style="list-style-type: none"> ○ \$ackseq – Sequence number of last segment rx ○ \$Blksize – Next Block size • EndBlkUL

Function	Description
	<ul style="list-style-type: none"> • Abort \$Index \$subIndex \$abort_code <ul style="list-style-type: none"> ○ \$Index – Pointed index ○ \$subIndex – pointed sub-index ○ \$abort_code – transmitted abort code
Waitsegm \$num_seg	CAN master wait for \$num_seq CCIPC SDO messages during the SDO block upload service
Fillbuf \$ADDRESS \$len	Host device fills \$len consecutive SRAM locations with static data starting from \$Address
Trig \$TPDO_NUM	Host Device requests the transmission of the \$TPDO_NUM TPDO.
Waitirq \$IRQ_TYPE	Wait for a specific IRQ. \$IRQ_TYPE: <ul style="list-style-type: none"> • Canmsg – CAN master waits rx message IRQ • Error – Host device waits CCIPC error IRQ • Pdo – Host device waits the end of PDO processing • Sdo – Host device waits the end of SDO processing • Synch_completed – Host device waits the end of SYNCH processing • Synch_rx – Host device waits synch object reception flag • Bus_switch – Host device waits bus switch flag
Dev read \$Address \$len	Host device reads \$len word starting from \$Address
Dev write \$Address \$data	Host device write data word at specific \$Address
Waitus \$time	Parser module waits for \$time us before reading next instruction

Tab. 8-1: Stimuli instruction definition.

(*) The **read_msg** function returns ID, Data and length of CAN message received and it is composed of the following command:

```

READ CAN 00000020: read RX_ARB_0 register
READ CAN 00000021: read RX_ARB_1 register
READ CAN 00000022: read RX_ARB_2 register
READ CAN 00000023: read RX_ARB_3 register
READ CAN 00000024: read RX_MSG_0 register
READ CAN 00000025: read RX_MSG_1 register
READ CAN 00000026: read RX_MSG_2 register
READ CAN 00000027: read RX_MSG_3 register
READ CAN 00000028: read RX_MSG_4 register
READ CAN 00000029: read RX_MSG_5 register
READ CAN 0000002A: read RX_MSG_6 register
READ CAN 0000002B: read RX_MSG_7 register
READ CAN 0000002C: read RX_STATUS register
    
```

The table below explains functions of the CAN MST receiver interface registers:

Name	Address(Hex)	Function[7:0]							
RX_ARB_0	00000020	ID[10]	ID[9]	ID[8]	ID[7]	ID[6]	ID[5]	ID[4]	ID[3]
RX_ARB_1	00000021	ID[2]	ID[1]	ID[0]	not used="00000"				
RX_ARB_2	00000022	not used							
RX_ARB_3	00000023	not used							
RX_MSG_0	00000024	1 st Byte received							
RX_MSG_1	00000025	2 nd Byte received							
RX_MSG_2	00000026	3 rd Byte received							
RX_MSG_3	00000027	4 th Byte received							
RX_MSG_4	00000028	5 th Byte received							
RX_MSG_5	00000029	6 th Byte received							
RX_MSG_6	0000002A	7 th Byte received							
RX_MSG_7	0000002B	8 th Byte received							
RX_STATUS	0000002C	not used				RX_len			

Tab. 8-2: CAN MST registers of Receiver interface.

(**) The **clear_irq** function allows clearing CAN message IRQ and it performs the following command:

```
WRITE CAN 00000003 A0: write SETUP_3 register
where SETUP_3 register is defined below:
```

Name	Address(Hex)	Function[7:0]					
SETUP_3	00000003	RxClear	Rst	IRQclear	--	TxReq	RSJ

(***) The **can_set** irq function allows enabling CAN IRQ on message reception and it performs the following command:

```
WRITE CAN 00000000 42: write SETUP_0 register
where SETUP_0 register is defined below:
```

Name	Address(Hex)	Function[7:0]						
SETUP_0	00000000	BPR	--	--	--	--	RX-irq	--

The meaning of the constants utilized in the test files is reported in Tab. 8-3:

Name	Value	Description
R_CF_ST_ADDR0	10035Ch	Pointer to IRQ status register (Index 2000 – SubIndex 0)
R_ST_ADDR	100364h	Pointer to HurriCANe & CCIPC status register (Index 2000 – SubIndex 2)
R_IRQ_ST_ADDR	10036Ch	Pointer to IRQ status register (Index 2000 – SubIndex 4)
R_RPDO_IRQ_ADDR0	100374h	Pointer to RPDO IRQ0 register (Index 2000 – SubIndex 6)
R_TPDO_IRQ_ADDR0	100384h	Pointer to TPDO IRQ0 register (Index 2000 – SubIndex 10)
R_IRQ_MK_CL_ADDR	100370h	Pointer to IRQ mask-clear register (Index 2000 – SubIndex 5)
R_SDO_ABORT_ADDR	100368h	Pointer to CCIPC-SDO error (Index 2000 – SubIndex 3)
R_EDAC_ERROR_ADDR	100394h	Pointer to EDAC register (Index 2003 – SubIndex 14)

Tab. 8-3: Test Constant value.

8.3 Output Test Log File

The CCIPC test-bench furnishes a simulation output Log file (*report.txt*) that reports all the traffic recorded on CAN bus and the memory accesses performed by Host interface. The Log file helps user during the

debug phase to understand what data are effectively exchanged between CAN Master, HOST and CCIPC.

The following information are reported in the Log File:

1. **Time stamp:** time reference expressed in ns;
2. **Access Type:** this field reports the type of operation to be executed:
 - a. **READ:** read operation;
 - b. **WRITE:** write operation.
3. **Peripheral:** this field defines which peripheral performs the operation:
 - a. **CAN:** CAN Master;
 - b. **EXT:** Host interface;
4. **Address:** Operation Address. The address refers to CAN Master internal registers when Peripheral field is CAN, while it refers to CCIPC Configuration Area or RAM area when Peripheral field is set to EXT;
5. **Data:** Data read or write according to Access type.

An example of Log File, without time stamp information, is reported in next table with a simple description of the operations performed:

Access Type	Per.	Address	Data	Description
WRITE	CAN	00000000	00000042	Write CAN Master SETUP_0 register.
WRITE	CAN	00000003	000000A0	Write CAN Master SETUP_3 register.
READ	CAN	00000020	000000E0	Read CAN Master Arbitration registers. (RX_ARB_0 – RX_ARB_3)
READ	CAN	00000021	00000041	
READ	CAN	00000022	00000000	
READ	CAN	00000023	00000030	
READ	CAN	00000024	00000000	Read CAN Master Message registers. (RX_MSG_0 – RX_MSG_7).
READ	CAN	00000025	00000030	
READ	CAN	00000026	0000008D	
READ	CAN	00000027	00000078	
READ	CAN	00000028	00000000	
READ	CAN	00000029	00000000	
READ	CAN	0000002A	00000000	
READ	CAN	0000002B	00000000	
READ	CAN	0000002C	00000001	Read CAN Master RX_STATUS register.
READ	EXT	00100364	00000810	HOST reads CCIPC "HurriCANE & CCIPC Status" register
READ	EXT	0010036C	00001000	HOST reads CCIPC "IRQ status" register
WRITE	EXT	00100370	10000000	HOST writes CCIPC "IRQ mask-clear" register
READ	EXT	0010036C	00000000	HOST reads CCIPC "IRQ status" register
READ	EXT	00200000	00000802	HOST reads External RAM memory
READ	EXT	00200004	60000120	

Tab. 8-4: Log File example and explanation.

Because the time stamp value could vary depending on simulator used, could be useful having a Log file without time stamp information. The following command creates a Log file (*report_val.txt*) without time information:

```
> cat report.txt | awk '{printf "%-5s %s %s %s \n", $3, $4, $5, $6}' > report_val.txt
```

8.4 Test Procedures Simulation

In this paragraph, the instructions to perform the tests included in the DataBase and to define new test procedures are described.

8.4.1 CCIPC Default Tests

All the tests corresponding to the CCIPC test procedure can be easily executed exploiting the CCIPC_test.sh shell-script.

The CCIPC_test script executes the following steps:

- Generates specific test configuration file
- Compiles the new VHDL testbench
- Launches the simulation scripts

At the end of simulation, it automatically checks if the test is correctly passed comparing the produced output file with the correspondent reference output file. All the test reference files (*report_XXXX.txt*) are contained in the "SIM_SCRIPT/VRP_TEST" directory.

Before starting with the test procedures, compile the *edac_rom* program that allows generating a ROM file compliant with the test-bench ROM device starting to one generated by the CCIPC Configuration Tool.

Enter in SIM_SCRIPT directory:

```
> cd YOUR_PATH/SIM_SCRIPT
```

and compile the program:

```
> gcc -o edac_rom edac_rom.c -lm
```

Now to perform the tests enter in your simulation directory

```
> cd YourSIM
```

Create a symbolic link to the following executables:

```
> ln -s CCIPC_DB/SIM_SCRIPT/CCIPC_test.sh  
> ln -s CCIPC_DB/SIM_SCRIPT/CCIPCTestWriter.pl  
> ln -s CCIPC_DB/SIM_SCRIPT/edac_rom
```

To run tests included in VRP_TB directory and reported in Tab. 5-5 launch the CCIPC_test.sh command specifying the test name.

```
> ./CCIPC_test.sh -name TP5113_boot
```

At the end of simulation, if the test is passed the following messages appears:

```
> Simulation success: files match
```

Otherwise, the test returns:

```
> Simulation failure: files differ
```

8.4.2 User Defined Tests

To create a new test procedure, generate the specific test file (*MYFILE.in*) using the instructions defined in Tab. 8-1.

Using the *CCIPCTestWriter* script, compile the *MYFILE.in* test and redirect the output to the *stimuli.txt* file.

```
> ./CCIPCTestWriter.pl MYFILE.in > YourSIM/stimuli.txt
```

If a new ROM image file has to be generated, use the following command (*dcf.dat* is the default input file for the test-bench ROM device):

```
> ./edac_rom YOUR_ROM.txt dcf.dat
```

Compile and simulate the new test.

9 SYNTHESIS AND FITTING SCRIPTS

Synthesis and fitting scripts for the different devices and corresponding tools are included in the *SYN_SCRIPT* and *FIT_SCRIPT* directories.

The constraint files (*CCIPC.sdc* for Microsemi and *CCIPC.ucf* for Xilinx), included in the *SYN_SCRIPT* directory, keep the system clock and peripheral timing constraints for the CCIPC design and they are used during synthesis process.

Before proceeding with Synthesis and Fitting processes assure that all files needed to instantiate memory module for the selected technology have been correctly generated and copied in the correspondent directory (see §5.2.1).

9.1 Microsemi (RT)AX FPGA

9.1.1 Synthesis

In *SYN_SCRIPT/ACTEL* directory three different example of synthesis project files are available to synthesize the CCIPC in one of the possible configurations (Generic I/O, AMBA and Direct).

Open the desired file with the Synplify Pro Actel Edition tool.

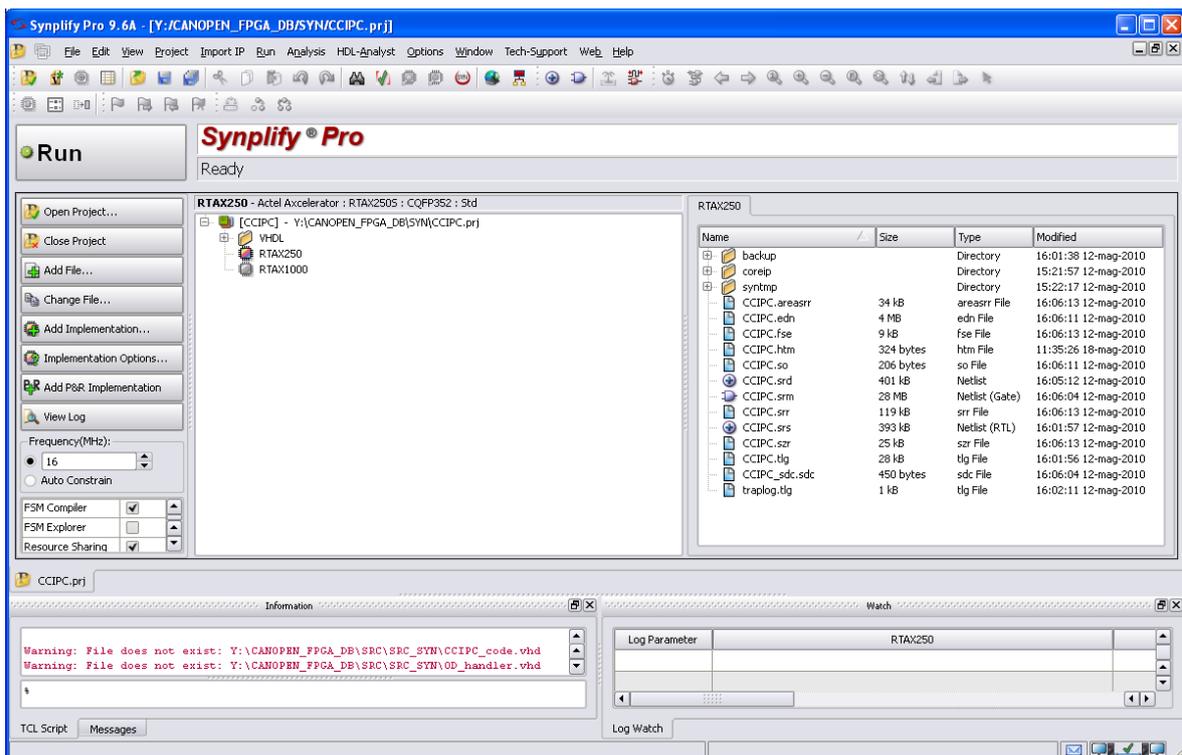


Fig. 9-1: Microsemi synthesis tool.

If necessary, use the Implementation Option menu (on the left edge) to synthesis option:

- Device selection: choose target FPGA
- Options: change synthesis option
- Constraint: add constraint files
- Implementation results: change synthesis output file destination
- VHDL: set VHDL options

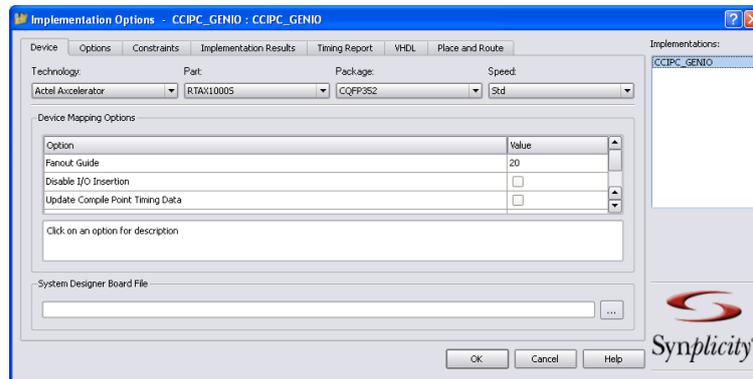


Fig. 9-2: Implementation options window.

After configuration phase, start the synthesis process clicking on Run button in the left-top side of the window.

The Synthesis process generates, in the specified output directory, the correspondent netlist file with .edn extension.

9.1.2 Fitting

When the synthesis process is completed, in the *FIT_SCRIPT/RTAX* directory, open the *CCIPC.adb* file using the Microsemi Designer tool.

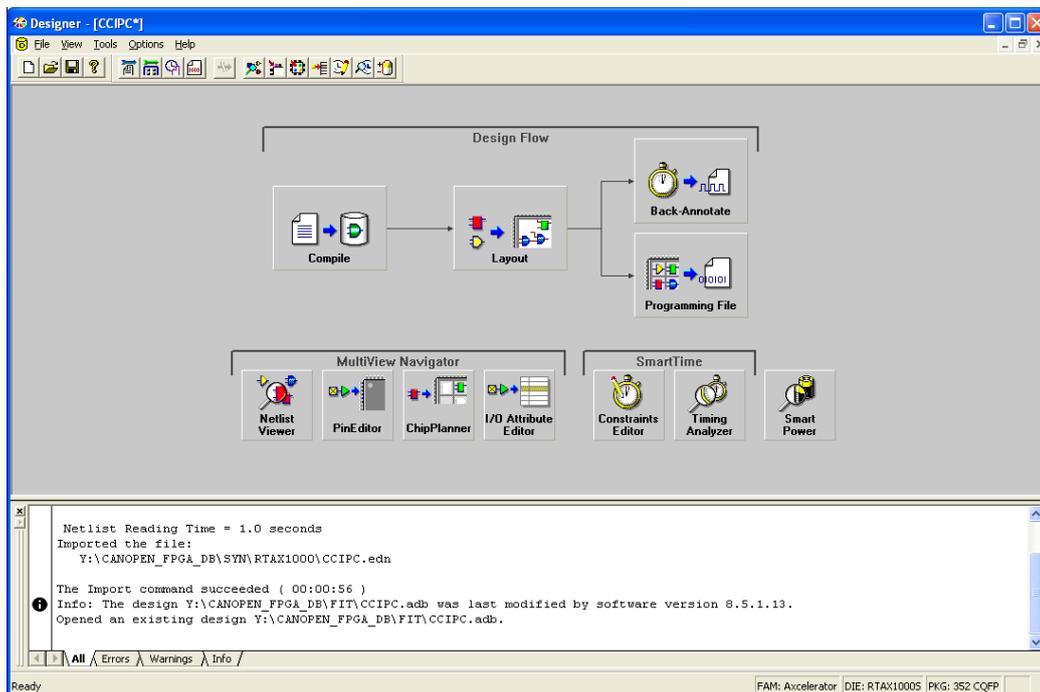


Fig. 9-3: Microsemi fitting tool.

Utilize the Options->Device Selection menu (on the left edge) to change the Device, speed grade, package etc. selections if needed.

Click on the “Programming File” button to start the fitting process.

The fitting process completes with generation of the *CCIPC.pdb* FPGA programming file.

9.2 Xilinx FPGA

9.2.1 Synthesis

The Synthesis process for the Xilinx FPGAs can be executed in two different ways depending on the Xilinx proprietary tool is used or not. In this paragraph, the steps needed to synthesize the CCIPC using a different synthesis tool than Xilinx one are described.

In *SYN_SCRIPT/VIRTEX* directory three different example of synthesis project files are available to synthesize the CCIPC in one of the possible configurations (Generic I/O, AMBA and Direct).

Open the desired file with the Synplify Pro tool:

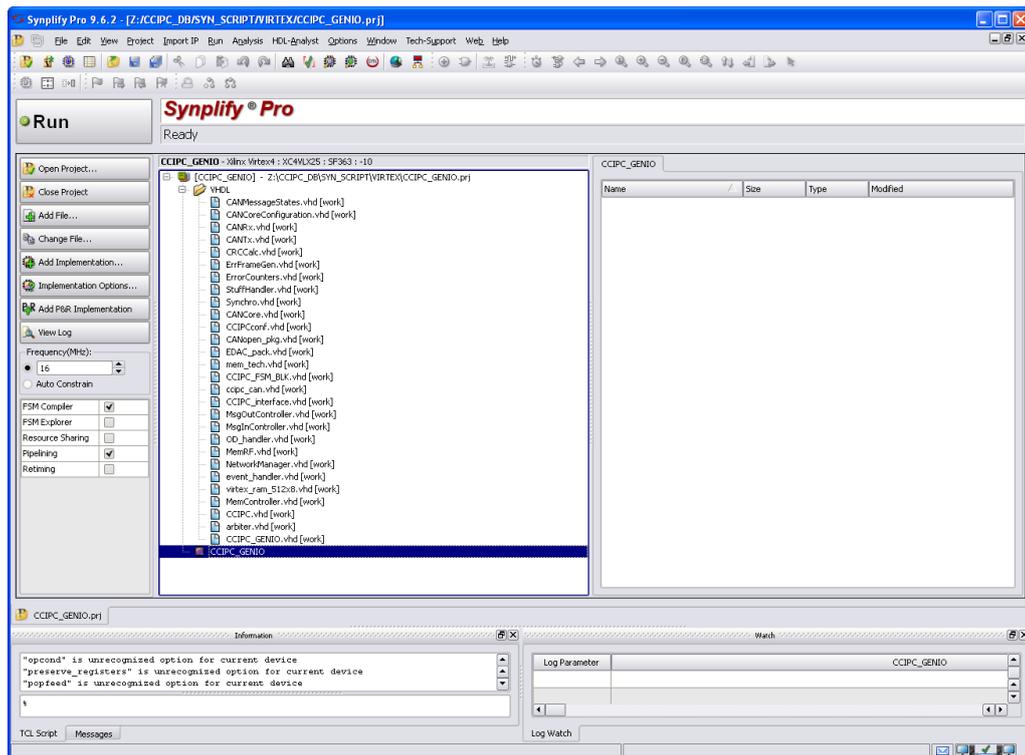


Fig. 9-4: Synplify Pro synthesis tool.

If necessary, use the Implementation Option menu (on the left edge) to modify synthesis options:

- Device selection: choose target FPGA
- Options: change synthesis option
- Constraint: add constraint files
- Implementation results: change synthesis output file destination
- VHDL: set VHDL options

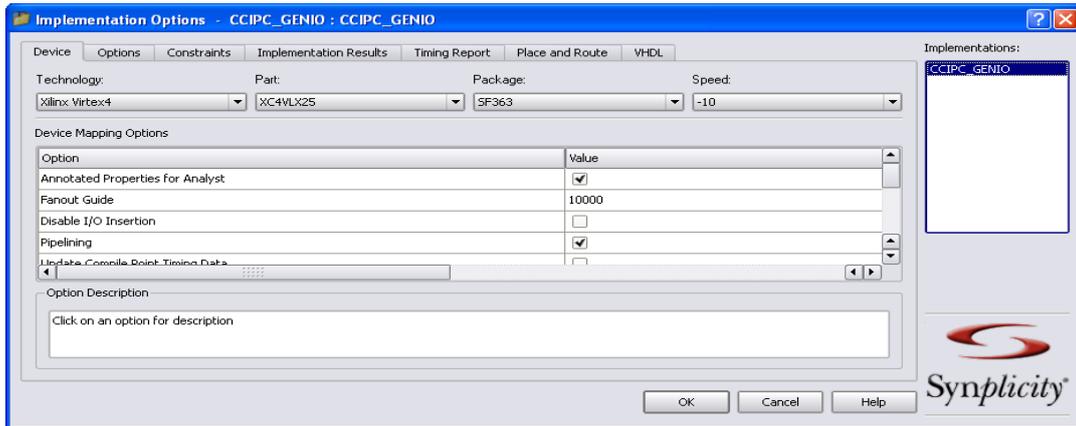


Fig. 9-5: Implementation options window.

After configuration phase, start the synthesis process clicking on Run button in the left-top side of the window.

The Synthesis process generates, in the *ccipc_db/fit* specified output directory, the correspondent netlist file with *.edn* extension.

9.2.2 Fitting

When the synthesis process is completed, in the *FIT_SCRIPT/VIRTEX* directory the example ISE project files correspondent to the three possible CCIPC configurations are available.

Open the correct project file using Xilinx ISE tool. The project automatically include the memory netlist file (*.ncg* file) expecting it under *LIBRARIES/VIRTEX* directory.

Start the Place and Route process double-clicking on "Implement Design" option in "Processes" window.

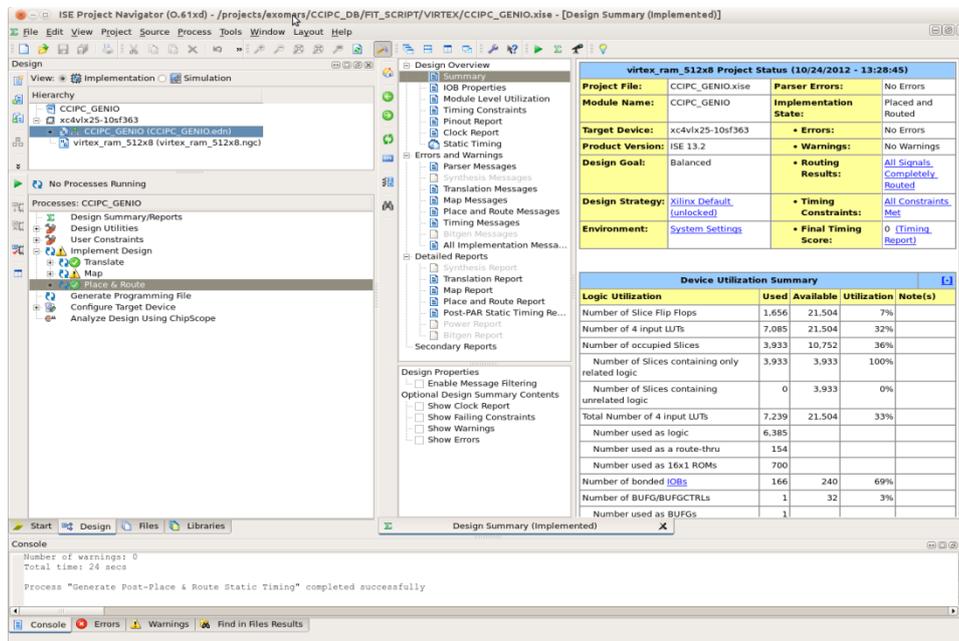


Fig. 9-6: Xilinx ISE tool.

10 CCIPC MEMORY MAPPING

In this paragraph, the Memory Space mapping defined for the CCIPC core is reported. The CCIPC utilizes, in its complete version, four different memory areas:

- **Configuration & Status area.** This is implemented in sparse registers assigned to control the main parameters inside the Core. For each entry of this area two different addresses exists:
 - Index.SubIndex CANopen OD reference to access them via CANbus (by CANopen master with SDO)
 - Cfg Address to be used by the Host Device to map the CCIPC registers it memory area
- **On board memory areas.** These are the communication parameters accessible only from CANbus, and the CCIPC private table index not reachable from external ports
- **Shared memory areas.** These are the mapping parameters and the application object that have to be mapped into an external memory to be shared between the CCIPC and its host device;
- **Initialization Area.** This area contains the default values to initialize the CCIPC and the Object Dictionary after Power-Node or CANopen NMT reset.

The following table summarize the CCIPC memory areas and their access mode.

Memory Area	Section	CANOpen access	Host Interface access
Configuration & Status	--	SDO Read, (partially)Write	Read, Write
On board	Communication Parameters	SDO Read Only	No
	Addressing pointers	No	No
Shared	Mapping Parameters	SDO Read Only	Do not modify
	AOs	PDO/SDO Read Write	Read Write

Tab. 10-1: Memory area accessibility.

10.1 Configuration & Status Area

The CCIPC Configuration & Status area is composed of the following OD entries:

OD entry	Index	Description
Device Type	1000h	Describes the type of device and its functionality
Error register	1001h	Device error register
COB-ID synch	1005h	COB-ID for Synch object
Synchronous window length	1007h	Length of the time window for synchronous PDO
Consumer Heartbeat time	1016h	Defines the expected heartbeat cycle time (ms)
Producer Heartbeat time	1017h	Defines the heartbeat cycle time (ms)
Identity Object	1018h	Contains general information about the device
Serves SDO	1200h	Describes the SDO server parameters
Redundancy Management	2000h	Describes the Bus Redundancy management parameters
Filler entry	2001h	---
Node parameters	2002h	Store the Node-ID and Node Mask defined for the node
CCIPC status and configuration	2003h	Store the main parameters of the CAN controller(e.g. Bit-Rate,Errors..)

Tab. 10-2: I/O area definition.

In Tab. 10-3 the list of registers defined in the Configuration Area is reported together with their address value.

Index	SubIndex	Register Name	Cfg Address	Obj Type
1000h		Device Type	0x100200	VAR_U32
1001h		Error register	0x100204	VAR_U8
1005h		COB-ID synch	0x100008	VAR_U32
1007h		Synchronous Window length	0x10020C	VAR_U32
1016h	0h	Heartbeat consumer time	0x100210	U8
	1h	Master Consumer Heartbeat time	0x100214	U32
1017h		Producer Heartbeat time	0x100218	U16
1018h	0h	Identity object	0x10021C	U8
	1h	Vendor-ID	0x100220	U32
	2h	Product code	0x100224	U32
	3h	Revision number	0x100228	U32
	4h	Serial Number	0x10022C	U32
1200h	0h	Server SDO	0x100130	U8
	1h	COB-ID client-server	0x100134	U32
	2h	COB-ID server-client	0x100138	U32
2000h	0h	Redundancy Management	0x10023C	U8
	1h	Bdefault	0x100240	U8
	2h	Ttoggle	0x100244	U8
	3h	Ntoggle	0x100248	U8
	4h	Ctoggle	0x10024C	U8
2001	0h	Filler entry		U8
2002h	0h	Node parameters	0x100050	U8
	1h	ID base	0x100054	U8
	2h	ID mask	0x100058	U8
2003h	0h	CCIPC status & configuration	0x10035C	U8
	1h	HurriCANe configuration	0x100360	U32
	2h	HurriCANe & CCIPC status	0x100364	U32
	3h	CCIPC-SDO error	0x100368	U32
	4h	IRQ status	0x10036C	U32
	5h	IRQ mask-clear	0x100370	U32
	6h	RPDO IRQ0	0x100374	U32
	7h	RPDO IRQ1	0x100378	U32
	8h	RPDO IRQ2	0x10037C	U32
	9h	RPDO IRQ3	0x100380	U32
	Ah	TPDO IRQ0	0x100384	U32
	Bh	TPDO IRQ1	0x100388	U32

Index	SubIndex	Register Name	Cfg Address	Obj Type
	Ch	TPDO IRQ2	0x10038C	U32
	Dh	TPDO IRQ3	0x100390	U32
	Eh	EDAC error	0x100394	U32
	Fh	TPDO Trig	0x100398	U32
	10h	SDO multiplexor	0x10039C	U32

Tab. 10-3: Configuration Register mapping.

Each register is defined in next tables. Each table is composed of the following fields:

- **field**: it associates a mnemonic name to register field;
- **sdo**: it indicates the access type supported via SDO;
- **host**: it indicated how Host can access to the register;
- **rst**: it indicates the reset value of the field. The following notation is used:
 - **Binary**: single(') or double(") quotes indicate binary values;
 - **Hexadecimal**: 0xnn notation indicates an Hexadecimal number;
 - **RST_VAL**: the reset value is indicated in the correspondent parameter available in the CCICP configuration file (*CCIPCconf.vhd*).

Three different access types are defined:

- **ro** – Read Only;
- **rw** – Read Write;
- **rc** – Read and clean. A write access only cleans the value of the register.

➤ **Device Type Register: Index 1000h - Sub-Index 0h - Address 0x100200**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	addon_info																profile_number															
sdo	ro																ro															
host	ro																ro															
rst	<i>DEV_TYPE[31..16]</i>																<i>DEV_TYPE[15..0]</i>															

Bit Number	Mnemonic	Description
15..0	profile_number	Device Profile
31..16	addon_info	Device Additional info

➤ **Error Register: Index 1001h - Sub-Index 0h - Address 0x100204**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							man_err	reserved	dev_err	comm_err	temp	voltage	current	gen_err	
sdo	--																							ro								
host	ro																							rw								
rst	0x000000																							0x00								

Bit Number	Mnemonic	Description
0	gen_err	Generic Error
1	current	Current error
2	voltage	Voltage error
3	temperature	Temperature error
4	comm_err	Communication error

Bit Number	Mnemonic	Description
5	dev_err	Device profile specific error
7	man_err	Manufacturer specific error

➤ **COB-ID Sync: Index 1005h - Sub-Index 0h - Address 0x100008**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	sync_cobid																															
sdo	ro																															
host	ro																															
rst	0x80																															

Bit Number	Mnemonic	Description
31..0	synch_cobid	COB-ID of Sync message

➤ **Synchronous Window Length: Index 1007h - Sub-Index 0h - Address 0x10020C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	swl																															
sdo	ro																															
host	ro																															
rst	SYNCH_WLEN																															

Bit Number	Mnemonic	Description
31..0	swl	Synchronous window length value

➤ **Heartbeat Consumer Time : Index 1016h - Sub-Index 0h - Address 0x100210**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																hbc_s0															
sdo	--																ro															
host	ro																ro															
rst	0x000000																0x1															

Bit Number	Mnemonic	Description
7..0	hbc_s0	Subindex0 of Heartbeat Consumer Time entry

➤ **Master - Heartbeat Consumer Time: Index 1016h - Sub-Index 1h - Address 0x100214**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved								m-id								hbc															
sdo	--								rw								rw															
host	ro								ro								ro															
rst	0x00								MST_NODEID								CONS_HB															

Bit Number	Mnemonic	Description
15..0	hbc	Heartbeat consumer time value (in ms)
23..16	m-id	Node-Id of expected Heartbeat

➤ **Heartbeat Producer Time: Index 1017h - Sub-Index 0h - Address 0x100218**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																hbp															
sdo	--																rw															
host	ro																ro															
rst	0x000000																PROD_HB															

Bit Number	Mnemonic	Description
15..0	hbo	Heartbeat producer time value (in ms)

➤ **Identity Object : Index 1018h - Sub-Index 0h - Address 0x10021C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							identity_s0								
sdo	--																							ro								
host	ro																							ro								
rst	0x000000																							0x4								

Bit Number	Mnemonic	Description
7..0	identity_s0	SubIndex-0 of Identity Object entry

➤ **Vendor-Id: Index 1018h - Sub-Index 1h - Address 0x100220**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	Vendor-Id																															
sdo	ro																															
host	ro																															
rst	0x0305																															

Bit Number	Mnemonic	Description
31..0	Vendor-Id	Vendor Id

➤ **Product Code: Index 1018h - Sub-Index 2h - Address 0x100224**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	Product code																															
sdo	ro																															
host	ro																															
rst	0																															

Bit Number	Mnemonic	Description
31..0	Product code	Product code

➤ **Revision Number Code: Index 1018h - Sub-Index 3h - Address 0x100228**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	rev_num																															
sdo	ro																															
host	ro																															
rst	0x2000																															

Bit Number	Mnemonic	Description
31..0	Rev_num	Revision Number

➤ **Serial Number Code: Index 1018h - Sub-Index 4h - Address 0x10022C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	ser_num																															
sdo	ro																															
host	ro																															
rst	0x0																															

Bit Number	Mnemonic	Description
31..0	ser_num	Serial Number

➤ **Server SDO Parameter : Index 1200h - Sub-Index 0h - Address 0x100130**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																sdo_s0															
sdo	--																ro															
host	ro																ro															
rst	0x000000																0x2															

Bit Number	Mnemonic	Description
7..0	sdo_s0	SubIndex-0 of Server SDO Parameter entry

 ➤ **COB-ID client-server: Index 1200h - Sub-Index 1h - Address 0x100134**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	id_cs																															
sdo	ro																															
host	ro																															
rst	0x600 + CCIPC_NodeID																															

Bit Number	Mnemonic	Description
31..0	id_cs	COB-ID client server

 ➤ **COB-ID server-client: Index 1200h - Sub-Index 2h - Address 0x100138**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	id_sc																															
sdo	ro																															
host	ro																															
rst	0x580 + CCIPC_NodeID																															

Bit Number	Mnemonic	Description
31..0	id_sc	COB-ID server client

 ➤ **Redundancy Management : Index 2000h - Sub-Index 0h - Address 0x10023C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																brm_s0															
sdo	--																ro															
host	ro																ro															
rst	0x000000																0x4															

Bit Number	Mnemonic	Description
7..0	brm_s0	SubIndex-0 of Bus Redundancy management entry

 ➤ **Bdefault: Index 2000h - Sub-Index 1h - Address 0x100240**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																reserved										b_def					
sdo	--																ro															
host	ro																ro															
rst	0x0																B_DEF															

Bit Number	Mnemonic	Description
1	b_def	Default CAN Bus

➤ **Ttoggle: Index 2000h - Sub-Index 2h - Address 0x100244**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							t_toggle								
sdo	--																							rw								
host	ro																							rw								
rst	0x0																							T TOGGLE								

Bit Number	Mnemonic	Description
7..0	t_toggle	Ttoggle value

➤ **Ntoggle: Index 2000h - Sub-Index 3h - Address 0x100248**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							n_toggle								
sdo	--																							rw								
host	ro																							rw								
rst	0x0																							N TOGGLE								

Bit Number	Mnemonic	Description
7..0	n_toggle	Ntoggle value

➤ **Ctoggle: Index 2000h - Sub-Index 4h - Address 0x10024C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							c_toggle								
sdo	--																							rc								
host	ro																							rc								
rst	0x0																							0x00								

Bit Number	Mnemonic	Description
7..0	c_toggle	Ctoggle value

➤ **Node Parameters : Index 2002h - Sub-Index 0h - Address 0x100050**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							Nodepar_s0								
sdo	--																							ro								
host	ro																							ro								
rst	0x000000																							0x2								

Bit Number	Mnemonic	Description
7..0	Nodepar_s0	SubIndex-0 of Node Parameters entry

➤ **ID Base: Index 2002h - Sub-Index 1h - Address 0x100054**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							node_id								
sdo	--																							ro								
host	ro																							ro								
rst	0x0																							0x00								

Bit Number	Mnemonic	Description
7..0	node_id	CCIPC Node-ID value sampled from external dedicated pins

➤ **ID Mask: Index 2002h - Sub-Index 2h - Address 0x100058**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							node_mask								
sdo	--																							ro								
host	ro																							ro								
rst	0x0																							0x00								

Bit Number	Mnemonic	Description
7..0	node_mask	CCIPC Node Mask value calculated from external dedicated pins

➤ **CCIPC Status & Configuration : Index 2003h - Sub-Index 0h - Address 0x10035C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							st&cfg_s0								
sdo	--																							ro								
host	ro																							ro								
rst	0x000000																							0x10								

Bit Number	Mnemonic	Description
7..0	st&cfg	SubIndex-0 of CCIPC Status and Configuration entry

➤ **HurriCANE configuration: Index 2003h - Sub-Index 1h - Address 0x100360**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved													can_rst	RSJ	PS2		PS1		BPR												
sdo	--													ro	ro	ro		ro		ro												
host	ro													rw	rw	rw		rw		rw												
rst	0x0													'1'	RSJ_VAL	PS2_VAL		PS1_VAL		BPR_VAL												

Bit Number	Mnemonic	Description
1..0	BPR	Frequency scaler value
5..2	PS1	Phase Segment 1 value
9..6	PS2	Phase Segment 2 value
12..10	RSJ	Resynchronization Jump width
13	can_rst	Active High CAN Core reset

➤ **HurriCANE & CCIPC status: Index 2003h - Sub-Index 2h - Address 0x100364**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	tx_err_cnt						rx_err_cnt						reserved						auto_op	bus_off	err_passive	reserved	msgin_full	active_bus	ccipc_state							
sdo	ro						ro						ro						ro	ro	ro	ro	ro	ro	ro	ro						
host	ro						ro						ro						ro	ro	ro	ro	ro	rw	ro	ro						
rst	0x0						0x0						0x0						AUTO_OPER	'0'	'0'	'0'	'0'	'0'	"000000"							

Bit Number	Mnemonic	Description
5..0	ccipc_state	CCIPC NMT state value: <ul style="list-style-type: none"> • 000001 → Initialising • 000010 → Reset Application • 000100 → Reset Communication • 001000 → Pre-Operational • 010000 → Operational • 100000 → Stopped
6	active_bus	Currently CAN bus used
7	msgin_full	<i>MsgIn Queue full</i> condition. This condition arises when number of received CAN message is greater than 32 , that corresponds to the size of queue of incoming message. This condition is handle as a CAN interface error and is associated to Error IRQ line. User has to clear this bit during the IRQ handling.
9	err_passive	CAN core Error Passive condition.
10	bus_off	CAN core Bus Off condition.
11	auto_op	This flag indicates if the Auto-operational condition is activated or not. If '1' CCIPC automatically enters in Operational state after Initialization phase.
23..16	rx_err_cnt	CAN core Receiver error counter value.
31..24	tx_err_cnt	CAN core Transmitter error counter value.

 ➤ **SDO Abort Code: Index 2003h - Sub-Index 3h - Address 0x100368**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	sdo_abort_code																															
sdo	ro																															
host	ro																															
rst	0x00000000																															

Bit Number	Mnemonic	Description
31..0	sdo_abort_code	SDO Abort Code. It stores the last abort code received or transmitted.

The CCIPC returns the following abort codes:

Abort Code (Hex)	Description
05030000	Toggle bit not alternated
05040001	Client Server command specifier not valid or unknown
06010002	Attempt to write a read only object
06070010	Data type does not match, length of service parameter does not match
06090011	Subindex does not exist
05040002	Invalid Block size
05040003	Invalid Sequence number
08000000	General Error

Tab. 10-4: CCIPC Abort condition and test configurations

➤ **IRQ status: Index 2003h - Sub-Index 4h - Address 0x10036C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
field	reserved																can_err	edac_err	bus_sw	nmt_cos	init_sdo	tpdo_serr	tpdo_genererr	rpdo_flush	rpdo_ff	rpdo_genererr	swl_exp	end_sync	sdo_abort	end_sdo	pdo_tx	pdo_rx				
sdo	--																ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	
host	--																ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro
rst	0x0000																'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'

Bit Number	Mnemonic	Description	IRQ line														
0	pdo_rx	RPDO successfully processed	TRX_IRQ														
1	pdo_tx	TPDO successfully processed	TRX_IRQ														
2	end_sdo	SDO successfully completed	TRX_IRQ														
3	sdo_abort	SDO abort detected	ERR_IRQ														
4	end_sync	SYNC elaboration completed	SYNC_IRQ														
5	swl_exp	Synchronous Window expired	ERR_IRQ														
6	rpdo_generr	RPDO generic error includes the following conditions: <ul style="list-style-type: none"> Addressed RPDO has a not valid ID; Wrong Mapping parameter; Wrong length of service 	ERR_IRQ														
7	rpdo_ff	Synch Fifo Full. Synch Fifo full condition arises when number of synchronous RPDO messages received is greater than Synch Fifo size. The supported size is reported in the next table and is dependent on number of nodes supported: <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Nodes Number</th> <th>Fifo size</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>4</td> </tr> <tr> <td>2</td> <td>8</td> </tr> <tr> <td>4</td> <td>16</td> </tr> <tr> <td>8</td> <td>16</td> </tr> <tr> <td>16</td> <td>16</td> </tr> <tr> <td>32</td> <td>16</td> </tr> </tbody> </table>	Nodes Number	Fifo size	1	4	2	8	4	16	8	16	16	16	32	16	ERR_IRQ
Nodes Number	Fifo size																
1	4																
2	8																
4	16																
8	16																
16	16																
32	16																
8	rpdo_flush	Synch Fifo Flush.	ERR_IRQ														

Bit Number	Mnemonic	Description	IRQ line
		This condition arises when Synchronous Window expired and some RPDOs are still in the Synch Fifo.	
9	tpdo_generr	TPDO generic error includes the following conditions: <ul style="list-style-type: none"> Addressed TPDO has a not valid ID; Wrong Mapping parameter 	ERR_IRQ
10	tpdo_serr	Synchronous TPDO not managed. Synchronous Window expires and some TPDOs have to be elaborated.	ERR_IRQ
11	init_sdo	Initiate SDO request accepted.	TRX_IRQ
12	nmt_cos	CCIPC change of state.	TRX_IRQ
13	bus_sw	CCIPC executes a Bus switch operation	ERR_IRQ
14	edac_err	EDAC error flag.	ERR_IRQ
15	can_err	Error on CAN interface.	ERR_IRQ

The following register allows masking and clearing different CCIPC IRQ. The following notation is used to mask and unmask IRQ:

- '0' – not masked. IRQ propagated
- '1' – masked. IRQ not propagated.

➤ **IRQ Mask-clear: Index 2003h - Sub-Index 5h - Address 0x100370**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	can_err_clear	edac_err_clear	bus_sw_clear	nmt_cos_clear	init_sdo_clear	tpdo_serr_clear	tpdo_generr_clear	rpdo_flush_clear	rpdo_ff_clear	rpdo_generr_clear	swl_exp_clear	end_sync_clear	sdo_abort_clear	end_sdo_clear	pdo_tx_clear	pdo_rx_clear	can_err_mask	edac_err_mask	bus_sw_mask	nmt_cos_mask	init_sdo_mask	tpdo_serr_mask	tpdo_generr_mask	rpdo_flush_mask	rpdo_ff_mask	rpdo_generr_mask	swl_exp_mask	end_sync_mask	sdo_abort_mask	end_sdo_mask	pdo_tx_mask	pdo_rx_mask
sdo	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro
host	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'

Bit Number	Mnemonic	Description
0	pdo_rx_mask	Mask RPDO successfully processed IRQ.
1	pdo_tx_mask	Mask TPDO successfully processed IRQ.
2	end_sdo_mask	Mask SDO successfully completed IRQ.
3	sdo_abort_mask	Mask SDO abort detected IRQ.
4	end_sync_mask	Mask SYNC elaboration completed IRQ.
5	swl_exp_mask	Mask Synchronous Window expired IRQ.
6	rpdo_generr_mask	Mask RPDO generic error IRQ.
7	rpdo_ff_mask	Mask Synch Fifo Full IRQ.
8	rpdo_flush_mask	Mask Synch Fifo Flush IRQ.
9	tpdo_generr_mask	Mask TPDO generic error IRQ.
10	tpdo_serr_mask	Mask Synchronous TPDO not managed IRQ.
11	init_sdo_mask	Mask Initiate SDO request accepted IRQ.
12	nmt_cos_mask	Mask CCIPC change of state IRQ.
13	bus_sw_mask	Mask CCIPC executes a Bus switch operation IRQ.
14	edac_err_mask	Mask EDAC error flag IRQ.
15	can_err_mask	Mask Error on CAN interface IRQ.

Bit Number	Mnemonic	Description
16	pdo_rx_clear	Clear RPDO successfully processed IRQ. When set to '1' it automatically clear also all the RPDO IRQ registers.
17	pdo_tx_clear	Clear TPDO successfully processed IRQ. When set to '1' it automatically clear also all the RPDO IRQ registers.
18	end_sdo_clear	Clear SDO successfully completed IRQ.
19	sdo_abort_clear	Clear SDO abort detected IRQ.
20	end_sync_clear	Clear SYNC elaboration completed IRQ.
21	swl_exp_clear	Clear Synchronous Window expired IRQ.
22	rpdo_generr_clear	Clear RPDO generic error IRQ.
23	rpdo_ff_clear	Clear Synch Fifo Full IRQ.
24	rpdo_flush_clear	Clear Synch Fifo Flush IRQ.
25	tpdo_generr_clear	Clear TPDO generic error IRQ.
26	tpdo_serr_clear	Clear Synchronous TPDO not managed IRQ.
27	init_sdo_clear	Clear Initiate SDO request accepted IRQ.
28	nmt_cos_clear	Clear CCIPC change of state IRQ.
29	bus_sw_clear	Clear CCIPC executes a Bus switch operation IRQ.
30	edac_err_clear	Clear EDAC error flag IRQ.
31	can_err_clear	Clear Error on CAN interface IRQ.

The following registers (RPDO-IRQ0-3 and TPDO-IRQ0-3) indicate which RPDOs and TPDOs have been successfully elaborated.

The number of RPDOs and TPDOs available depends on the number of node (**Node count**) allocated for the CCIPC core. The following table describes the relationship between node count, T/RPDOs and IRQ register allocation:

Node Count	Max R(T)PDOs	Allocated Registers
1	4 (4)	R(T)PDO_IRQ0[3..0]
2	8 (8)	R(T)PDO_IRQ0[7..0]
4	16 (16)	R(T)PDO_IRQ0[15..0]
8	32 (32)	R(T)PDO_IRQ0
16	64 (64)	R(T)PDO_IRQ0 R(T)PDO_IRQ1
32	128 (128)	R(T)PDO_IRQ0 R(T)PDO_IRQ1 R(T)PDO_IRQ2 R(T)PDO_IRQ3

Tab. 10-5: NodeCount vs PDO IRQ registers.

➤ **RPDO-0 IRQ: Index 2003h - Sub-Index 6h - Address 0x100374**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
field	rpdo_31	rpdo_30	rpdo_29	rpdo_28	rpdo_27	rpdo_26	rpdo_25	rpdo_24	rpdo_23	rpdo_22	rpdo_21	rpdo_20	rpdo_19	rpdo_18	rpdo_17	rpdo_16	rpdo_15	rpdo_14	rpdo_13	rpdo_12	rpdo_11	rpdo_10	rpdo_9	rpdo_8	rpdo_7	rpdo_6	rpdo_5	rpdo_4	rpdo_3	rpdo_2	rpdo_1	rpdo_0	
sdo	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro																					
host	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro																					
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	

Bit Number	Mnemonic	Description
31 .. 0	rpdo_i	RPDO from 1 to 31 correctly elaborated. To clear this register write '1' on "pdo_rx_clear" field of "IRQ Mask-Clear register".

➤ **RPDO-1 IRQ: Index 2003h - Sub-Index 7h - Address 0x100378**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
field	rpdo_63	rpdo_62	rpdo_61	rpdo_60	rpdo_59	rpdo_58	rpdo_57	rpdo_56	rpdo_55	rpdo_54	rpdo_53	rpdo_52	rpdo_51	rpdo_50	rpdo_49	rpdo_48	rpdo_47	rpdo_46	rpdo_45	rpdo_44	rpdo_43	rpdo_42	rpdo_41	rpdo_40	rpdo_39	rpdo_38	rpdo_37	rpdo_36	rpdo_35	rpdo_34	rpdo_33	rpdo_32	
sdo	ro	ro																															
host	ro	ro																															
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	

Bit Number	Mnemonic	Description
31 .. 0	rpdo_i	RPDO from 32 to 63 correctly elaborated. To clear this register write '1' on "pdo_rx_clear" field of "IRQ Mask-Clear register".

➤ **RPDO-2 IRQ: Index 2003h - Sub-Index 8h - Address 0x10037C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
field	rpdo_95	rpdo_94	rpdo_93	rpdo_92	rpdo_91	rpdo_90	rpdo_89	rpdo_88	rpdo_87	rpdo_86	rpdo_85	rpdo_84	rpdo_83	rpdo_82	rpdo_81	rpdo_80	rpdo_79	rpdo_78	rpdo_77	rpdo_76	rpdo_75	rpdo_74	rpdo_73	rpdo_72	rpdo_71	rpdo_70	rpdo_69	rpdo_68	rpdo_67	rpdo_66	rpdo_65	rpdo_64	
sdo	ro	ro																															
host	ro	ro																															
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'

Bit Number	Mnemonic	Description
31 .. 0	rpdo_i	RPDO from 64 to 95 correctly elaborated. To clear this register write '1' on "pdo_rx_clear" field of "IRQ Mask-Clear register".

➤ **RPDO-3 IRQ: Index 2003h - Sub-Index 9h - Address 0x100380**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
field	rpdo_127	rpdo_126	rpdo_125	rpdo_124	rpdo_123	rpdo_122	rpdo_121	rpdo_120	rpdo_119	rpdo_118	rpdo_117	rpdo_116	rpdo_115	rpdo_114	rpdo_113	rpdo_112	rpdo_111	rpdo_110	rpdo_109	rpdo_108	rpdo_107	rpdo_106	rpdo_105	rpdo_104	rpdo_103	rpdo_102	rpdo_101	rpdo_100	rpdo_99	rpdo_98	rpdo_97	rpdo_96	
sdo	ro	ro	ro	ro	ro	ro																											
host	ro	ro	ro	ro	ro	ro																											
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'

Bit Number	Mnemonic	Description
31 .. 0	rpdo_i	RPDO from 96 to 127 correctly elaborated. To clear this register write '1' on "pdo_rx_clear" field of "IRQ Mask-Clear register".

TPDO-0 IRQ: Index 2003h - Sub-Index Ah - Address 0x100384

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
field	tpdo_31	tpdo_30	tpdo_29	tpdo_28	tpdo_27	tpdo_26	tpdo_25	tpdo_24	tpdo_23	tpdo_22	tpdo_21	tpdo_20	tpdo_19	tpdo_18	tpdo_17	tpdo_16	tpdo_15	tpdo_14	tpdo_13	tpdo_12	tpdo_11	tpdo_10	tpdo_9	tpdo_8	tpdo_7	tpdo_6	tpdo_5	tpdo_4	tpdo_3	tpdo_2	tpdo_1	tpdo_0	
sdo	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro																					
host	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro	ro																					
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	

Bit Number	Mnemonic	Description
31 .. 0	tpdo_i	TPDO from 1 to 31 correctly elaborated. To clear this register write '1' on "pdo_tx_clear" field of "IRQ Mask-Clear register".

TPDO-1 IRQ: Index 2003h - Sub-Index Bh - Address 0x100388

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
field	tpdo_63	tpdo_62	tpdo_61	tpdo_60	tpdo_59	tpdo_58	tpdo_57	tpdo_56	tpdo_55	tpdo_54	tpdo_53	tpdo_52	tpdo_51	tpdo_50	tpdo_49	tpdo_48	tpdo_47	tpdo_46	tpdo_45	tpdo_44	tpdo_43	tpdo_42	tpdo_41	tpdo_40	tpdo_39	tpdo_38	tpdo_37	tpdo_36	tpdo_35	tpdo_34	tpdo_33	tpdo_32	
sdo	ro	ro																															
host	ro	ro																															
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	

Bit Number	Mnemonic	Description
31 .. 0	tpdo_i	TPDO from 32 to 63 correctly elaborated. To clear this register write '1' on "pdo_tx_clear" field of "IRQ Mask-Clear register".

TPDO-2 IRQ: Index 2003h - Sub-Index Ch - Address 0x10038C

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
field	tpdo_95	tpdo_94	tpdo_93	tpdo_92	tpdo_91	tpdo_90	tpdo_89	tpdo_88	tpdo_87	tpdo_86	tpdo_85	tpdo_84	tpdo_83	tpdo_82	tpdo_81	tpdo_80	tpdo_79	tpdo_78	tpdo_77	tpdo_76	tpdo_75	tpdo_74	tpdo_73	tpdo_72	tpdo_71	tpdo_70	tpdo_69	tpdo_68	tpdo_67	tpdo_66	tpdo_65	tpdo_64	
sdo	ro	ro																															
host	ro	ro																															
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	

Bit Number	Mnemonic	Description
31 .. 0	tpdo_i	TPDO from 64 to 95 correctly elaborated. To clear this register write '1' on "pdo_tx_clear" field of "IRQ Mask-Clear register".

➤ **TPDO-3 IRQ: Index 2003h - Sub-Index Dh - Address 0x100390**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
field	tpdo_127	tpdo_126	tpdo_125	tpdo_124	tpdo_123	tpdo_122	tpdo_121	tpdo_120	tpdo_119	tpdo_118	tpdo_117	tpdo_116	tpdo_115	tpdo_114	tpdo_113	tpdo_112	tpdo_111	tpdo_110	tpdo_109	tpdo_108	tpdo_107	tpdo_106	tpdo_105	tpdo_104	tpdo_103	tpdo_102	tpdo_101	tpdo_100	tpdo_99	tpdo_98	tpdo_97	tpdo_96	
sdo	ro	ro	ro	ro	ro	ro																											
host	ro	ro	ro	ro	ro	ro																											
rst	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	

Bit Number	Mnemonic	Description
31 .. 0	tpdo_i	TPDO from 96 to 127 correctly elaborated. To clear this register write '1' on "pdo_tx_clear" field of "IRQ Mask-Clear register".

 ➤ **EDAC error: Index 2003h - Sub-Index Eh - Address 0x100394**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved														derr_mask	serr_mask	reserved					queue_en	mem_en	derr	serr	serr_cnt						
sdo	ro														ro	ro	ro					ro	ro	ro	ro	ro						
host	ro														rw	rw	ro					rw	rw	rc	rc	rw						
rst	"00000000000000"														'0'	'0'	0x0					'0'	'0'	'0'	'0'	0x00						

Bit Number	Mnemonic	Description
7..0	serr_cnt	Counter of Single error events. The counter of single error events can be initialized at a predefined value through a write access to this field.
8	serr	Single error flag. This bit is set when the serr_cnt exceeds the 255 errors. This bit is auto cleared every time this register is written.
9	derr	Double error flag. This bit is auto cleared every time this register is written.
10	mem_en	Enable EDAC capability on CCIPC memories.
11	queue_en	Enable EDAC capability on CCIPC queues.
16	serr_mask	Mask single error event <ul style="list-style-type: none"> '0' – not masked. IRQ propagated. '1' – masked. IRQ not propagated.
17	derr_mask	Mask double error event <ul style="list-style-type: none"> '0' – not masked. IRQ propagated. '1' – masked. IRQ not propagated.

The EDAC capabilities are disabled at start-up. User has to enable it after CCIPC initialization.

Since Double error detection represents a critical condition for the core because the management of wrong values can compromise the CCIPC behaviour, it is strictly recommended that user resets the CCIPC core if this error occurs.

➤ **TPDO Trig: Index 2003h - Sub-Index Fh - Address 0x100398**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved																							tpdo_num								
sdo	ro																							ro								
host	ro																							rw								
rst	0x0																							0x0								

Bit Number	Mnemonic	Description
7..0	tpdo_num	Number of TPDO to be sent.

The TPDO trig register is used to implement the TPDO event driven functionality. A write operation on this register enables the transmission of selected TPDO. Tab. 10-6 shows the correspondence between TPDO number and TPDO trig register value.

TPDO trig value	TPDO number	OD index
"000000"	0	1800h
"000001"	1	1801h
"000010"	2	1802h
"000011"	3	1803h
"000100"	4	1800h
"000101"	5	1805h
"000110"	6	1806h
"000111"	7	1807h
...
"111111"	127	187F

Tab. 10-6: TPDO number vs Trig value

 ➤ **SDO multiplexor: Index 2003h - Sub-Index 10h - Address 0x10039C**

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	reserved				sdo_up	sdo_dw	index															sub_index										
sdo	ro				ro	ro	ro															ro										
host	ro				rw	rw	rw															rw										
rst	"000000"				'0'	'0'	0x0000															0x00										

Bit Number	Mnemonic	Description
7..0	sub_index	Sub Index value of last SDO service
23..8	index	Index value of last SDO service
24	sdo_dw	Download SDO type
25	sdo_up	Upload SDO type

The following table shows which events cause the initialization of the registers:

Register	Field	Event				Note
		Power-On	Reset Application	Reset Communication	Bus Switch	
Device Type	addon_info	--	--	--	--	Constant value
	profile_number	y	y			

Register	Field	Event				Note
		Power-On	Reset Application	Reset Communication	Bus Switch	
Error Register	gen_err	y	n	y	y	
	current	y	n	y	y	
	voltage	y	n	y	y	
	temp	y	n	y	y	
	comm_err	y	n	y	y	
	dev_err	y	n	y	y	
	man_err	y	n	y	y	
COB-ID Synch	synch_cobid	y	n	y	y	
Synchronous Window Lenght	swl	--	--	--	--	Constant value
Heartbeat Consumer Time	hbc_s0	--	--	--	--	Constant value
Master Heartbeat Consumer Time	hbc	y	n	y	n	
	m-id	y	n	y	n	
Heartbeat Consumer Time	hbp	y	n	y	n	
Identity Object	identity_s0	--	--	--	--	Constant value
Vendor-Id	Vendor-Id	--	--	--	--	Constant value
Product Code	Product Code	--	--	--	--	Constant value
Revision Number Code	Revision Number	--	--	--	--	Constant value
Serial Number Code	Serial Number	--	--	--	--	Constant value
Server SDO Parameter	sdo_s0	--	--	--	--	Constant value
COB-ID client-server	id_cs	--	--	--	--	Constant value
COB-ID server-client	id_sc	--	--	--	--	Constant value
Redundacy Management	brm_s0	--	--	--	--	Constant value
Bdefault	b_def	y	y	n	n	
Ttoggle	t_toggle	y	y	n	n	
Ntoogle	n_toggle	y	y	n	n	
C_toggle	c_toggle	--	--	--	--	Constant value
Node Parameters	nodepar_s0	--	--	--	--	Constant value
ID Base	node_id	y	n	n	n	Value sampled at boot
ID Mask	node_mask	y	n	n	n	Value sampled at boot
CCIPC status & Configuration	st&cfg_s0	--	--	--	--	Constant value
HurriCANE Configuration	BPR	y	y	n	y	
	PS1	y	y	n	y	
	PS2	y	y	n	y	
	RSJ	y	y	n	y	
	can_rst	y	y	n	y	
HurriCANE & CCIPC status	ccipc_state	y	n	n	n	
	active_bus	y	n	n	n	
	msgin_full	y	y	n	y	
	err_passive	y	y	n	y	
	bus_off	y	y	n	y	
	auto_op	--	--	--	--	Constant value
	rx_err_cnt	y	y	n	y	
tx_err_cnt	y	y	n	y		
SDO Abort Code	sdo_abort_code	y	y	n	y	
IRQ status	pdo_rx	y	y	n	y	
	pdo_tx	y	y	n	y	
	end_sdo	y	y	n	y	
	sdo_abort	y	y	n	y	
	end_sync	y	y	n	y	
	swl_exp	y	y	n	y	
	rpdo_generr	y	y	n	y	
rpdo_ff	y	y	n	y		

Register	Field	Event				Note
		Power-On	Reset Application	Reset Communication	Bus Switch	
	rpdo_flush	y	y	n	y	
	tpdo_generr	y	y	n	y	
	tpdo_serr	y	y	n	y	
	init_sdo	y	y	n	y	
	nmt_cos	y	y	n	y	
	bus_sw	y	y	n	y	
	edac_err	y	y	n	y	
	can_err	y	y	n	y	
IRQ Mask-clear		y	y	n	y	
RPDO-0 IRQ		y	y	n	y	
RPDO-1 IRQ		y	y	n	y	
RPDO-2 IRQ		y	y	n	y	
RPDO-3 IRQ		y	y	n	y	
TPDO-0 IRQ		y	y	n	y	
TPDO-1 IRQ		y	y	n	y	
TPDO-2 IRQ		y	y	n	y	
TPDO-3 IRQ		y	y	n	y	
EDAC error	serr_cnt	y	y	n	y	
	serr	y	y	n	y	
	derr	y	y	n	y	
	mem_en	y	n	n	n	
	queue_en	y	n	n	n	
	serr_mask	y	y	n	y	
TPDO Trig	derr_mask	y	y	n	y	
	tpdo_num	y	y	n	y	
SDO multiplexor	sub_index	y	y	n	y	
	index	y	y	n	y	
	sdo_dw	y	y	n	y	
	sdo_ul	y	y	n	y	

Tab. 10-7: Initialization event for Configuration and Status Area Registers.

10.1.1 IRQ Handling

As already defined in §7.1.3 the CCIPC handles three IRQ lines:

- Transfer IRQ (TRX_IRQ)
- Synch elaboration completed (SYNCH_IRQ)
- Error IRQ (ERR_IRQ)

User has to clear the IRQ request. Three simple “pseudo-code” examples of CCIPC IRQ manager are written below:

TRX_IRQ

```

irq_status = IRQ_status_reg;
ARPDO_IRQ = irq_status[0];
ATPDO_IRQ = irq_status[1];
SDO_IRQ = irq_status[2];
INITSDO_IRQ = irq_status[11];
NMT_CS_IRQ = irq_status[12];           ## NMT change state

if (ARPDO_IRQ) {
    elab_rpdo = RPDO_IRQx_reg;
    while(elab_rpdo){
        <Elaborate Data>
    }
}
    
```

```

    IRQ_MASK_CLEAR_reg = 0x10000;      ## clear RPDO part of PDO_IRQ_reg
                                       ## clear also RPDO IRQ registers
}

if (ATPDO_IRQ) {
    elab_tpdo = TPDO_IRQX_reg;
    while(elab_tpdo){
        <Elaborate Data>
    }
    IRQ_MASK_CLEAR_reg = 0x20000;      ## clear TPDO part of PDO_IRQ_reg
                                       ## clear also TPDO IRQ registers
}

if (SDO_IRQ) {
    <Elaborate Data>
    IRQ_MASK_CLEAR_reg = 0x40000;      ## clear SDO IRQ
}

if (INITSDO_IRQ){
    <Read Muliplier register>
    IRQ_MASK_CLEAR_reg = 0x8000000;    ## clear SDO IRQ
}

if (NMT_CS){
    nmt_state = HurriCANE_&_CCIPC_status_reg & 0x1F>      ## read NMT state
    IRQ_MASK_CLEAR_reg = 0x10000000;    ## clear SDO IRQ
}

```

SYNCH_IRQ

```

irq_status = read(IRQ_status_reg);
SRPDO_IRQ = irq_status[0];
STPDO_IRQ = irq_status[1];

if (SRPDO_IRQ) {
    elab_rpdo = RPDO_IRQx_reg;
    while(elab_rpdo){
        <Elaborate Data>
    }
    IRQ_MASK_CLEAR_reg = 0x10000;      ## clear RPDO part of PDO_IRQ_reg
                                       ## clear also RPDO IRQ registers
}

if (STPDO_IRQ) {
    elab_tpdo = TPDO_IRQx_reg;
    while(elab_tpdo){
        <Elaborate Data>
    }
    IRQ_MASK_CLEAR_reg = 0x20000;      ## clear TPDO part of PDO_IRQ_reg
                                       ## clear also TPDO IRQ registers
}

IRQ_MASK_CLEAR_reg = 0x100000;        ## clear SYNCH_IRQ

```

If SYNCH_IRQ is cancelled before clearing the full PDO_IRQ register, the TRX_IRQ line rises up again for effect of CCIPC T/RPDO requests still pending and to be processed.

ERROR_IRQ

```
irq_status = read(IRQ_status_reg);
<Manage error>

IRQ_MASK_CLEAR_reg = ERROR_CLEAR_BIT;  ## clear specific error bit
```

The *MsgIn Queue full* (incoming message queue full) condition is associated to the HurriCANE error flag (bit 15 of IRQ status register). Before clean the IRQ status register, user has to clear the MsgIn Queue full error by writing the correspondent bit of *HurriCANE and CCIPC status*.

MsgIn Queue full error

```
irq_status = read(IRQ_status_reg);
can_err = irq_status[15];

if (can_err) {
    msgin_full = (read(HurriCANE&CCIPC_status_reg)) >> 7) ## Msgin full error
    if (msgin_full){
        HurriCANE&CCIPC_status_reg = 0                ## Clear Msgin error
    }
}

IRQ_MASK_CLEAR_reg = 0x8000000;                        ## clear Can error
```

10.2 On-board Area

This memory section is used to store:

- PDO Communication entries
- AOs index pointers

10.2.1 PDO Communication Entries

The PDO Communication area stores the OD entries used to define the RPDO and TPDO Communication parameters. The list of parameters available in this area is reported in Tab. 10-8.

This area is mapped inside the CCIPC core to speed up the elaboration time avoiding conflict with external devices.

OD entry		Name	Description
Index	SubIndex		
1400	0	Number of entries	1 st Receive PDO communication parameters
	1	RPDO COB-ID	
	2	RPDO Transmission type	
1401	0	Number of entries	2 nd Receive PDO communication parameters
	1	RPDO COB-ID	
	2	RPDO Transmission type	
....
15FF	0	Number of entries	512 th Receive PDO communication parameters

OD entry		Name	Description
Index	SubIndex		
	1	RPDO COB-ID	
	2	RPDO Transmission type	
1800	0	Number of entries	1 st Transmit PDO communication parameters
	1	TPDO COB-ID	
	2	TPDO Transmission type	
	3	TPDO Inhibit Time	
	4	Reserved	
	5	TPDO Event Timer	
1801	0	Number of entries	2 nd Transmit PDO communication parameters
	1	TPDO COB-ID	
	2	TPDO Transmission type	
	3	TPDO Inhibit Time	
	4	Reserved	
	5	TPDO Event Timer	
19FF	0	Number of entries	512 th Transmit PDO communication parameters
	1	TPDO COB-ID	
	2	TPDO Transmission type	
	3	TPDO Inhibit Time	
	4	Reserved	
	5	TPDO Event Timer	

Tab. 10-8: T/R PDO Communication parameters.

10.2.2 AOs Addressing Pointers

The AOs addressing area stores the pointers to the Shared Memory utilized by CCIPC to access Application Objects during SDO and PDO elaboration. These pointers are computed off-line at configuration time and loaded by CCIPC at power-on. This area is neither mapped in the object dictionary nor readable from the Host interface

This area is composed of two section:

- Index pointers area.
- Sub-Index pointers area.

The Index pointers area contains, for each Application Objects defined, the pointers to the Sub-Index0 of the entry.

The Sub-Index area contains the memory addresses of each Sub-index supported by the specific Index. In case of ARRAY or VAR entry type, the second stage of addressing (Sub-Index pointer area) is missing. In this case, the value returned by the Index area is the Shared memory address of Sub-Index0. A simple algorithm allows reaching the other sub-indexes.

If an entry is defined as RECORD both addressing stages are needed to correctly locate the specific Application Objects.

10.3 Shared Memory Area

The data stored in this area are:

- PDO Mapping parameters entries
- Application Objects

The Mapping parameters are stored starting from Shared Memory base address. The Application Objects can occupy the remaining part of the memory except for a dedicated area containing all EDAC bytes.

10.3.1 PDO Mapping Parameters

The PDO Mapping section stores the mapping parameters (Index, Sub-Index and Data type) defined for each RPDO and TPDO.

The list of parameters available in this area is reported in Tab. 10-9:

OD entry		Name	Description
Index	SubIndex		
1600	0	Number of entries	1 st Receive PDO mapping parameters
	1	1 st mapping parameter	
	
	8	8 th mapping parameter	
1601	0	Number of entries	2 nd Receive PDO mapping parameters
	1	1 st mapping parameter	
	
	8	8 th mapping parameter	
17FF	0	Number of entries	512 th Receive PDO mapping parameters
	1	1 st mapping parameter	
	
	8	8 th mapping parameter	
1A00	0	Number of entries	1 st Transmit PDO mapping parameters
	1	1 st mapping parameter	
	
	8	8 th mapping parameter	
1A01	0	Number of entries	2 nd Transmit PDO mapping parameters
	1	1 st mapping parameter	
	
	8	8 th mapping parameter	
1BFF	0	Number of entries	512 th Transmit PDO mapping parameters
	1	1 st mapping parameter	
	
	8	8 th mapping parameter	

Tab. 10-9: PDO mapping parameters.

Each PDO Mapping is stored in the Shared memory in a structure containing the sub-index 0 and all the 8 mapping parameters, including the ones not defined. Since each index occupies a single word (4 bytes), each Mapping parameter entry is 9 word length (36 bytes).

Since each CAN node supports 4 TPDOs and 4 RPDOs, the area dedicated to store the Mapping parameters is fixed and depends on number of nodes supported:

Node count	RPDOs	TPDOs	Tot. PDOs	Word Length	Address Range (*)
1	4	4	8	72	[base : base+ 11C]h
2	8	8	16	144	[base : base+ 23C]h
4	16	16	32	288	[base : base+ 47C]h
8	32	32	64	576	[base : base+ 8FC]h
16	64	64	128	1152	[base : base+ 11FC]h
32	128	128	256	2304	[base : base+ 23FC]h

Node count	RPDOs	TPDOs	Tot. PDOs	Word Length	Address Range (*)
(*) 'base' is the base address of the Shared Memory.					

Tab. 10-10: Memory occupation for Mapping parameters.

Since these parameters are stored in the external Shared memory, they are reachable by the Host interface.

To preserve Object Dictionary structure consistency, data in this area must not be modified by Host device.

10.3.2 Application Objects Area

Application Objects corresponding to OD entries from 6000h - 7000h (Read-Write) and 7000h - 8000h (Read-Only) are statically mapped in a memory area shared between CCIPC and its Host Device. The mapping of these parameters is assigned at configuration time and stored in the CCIPC ROM Image file (see §11.1).

To calculate the address of each Application Objects, user has to apply the following rules:

- 1) The Application Objects area starts immediately after the Mapping parameter area. The base address is a fixed value depending on number of node supported

Node count	Base Address
1	120h
2	240h
4	480h
8	900h
16	1200
32	2400

Tab. 10-11: Application Objects Area base address.

- 2) Each new entry starts at the next word aligned address (0h,4h,8h,Ch);
- 3) The sub-index 0 occupies a single word;
- 4) U8,U16 and U32 Application Objects objects occupies respectively 1, 2 or 4 bytes;
- 5) A VAR object occupies a single word, independently from its type (U8,U16 or U32);
- 6) Each sub-index within an entry occupies consecutive memory locations;
- 7) Each missing entry between two non consecutive entries is mapped with a 4 bytes filler entry;
- 8) The first entry of Read-Only Area is placed immediately after the last Read-Write.

The Host Device can perform read and write operations on data contained in this area.

To preserve the CCIPC correct elaboration of the Application Objects, Host Device must not overwrite the Sub-Index0 value of the AOs.

Fig. 10-1 shows an example of memory allocation for a simple Application Objects structure.

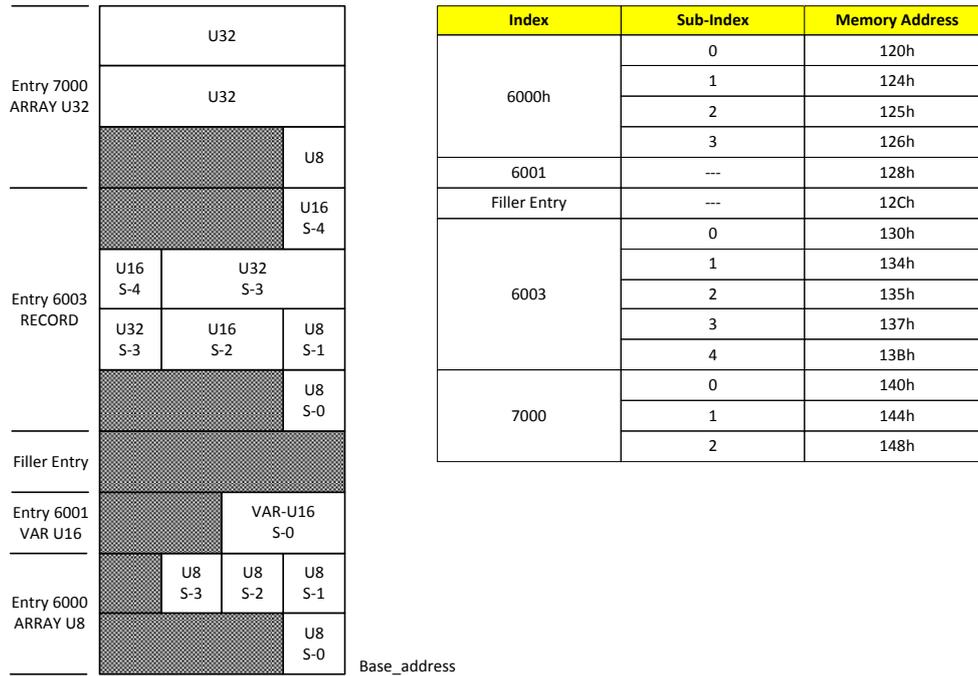


Fig. 10-1: Example of Application Objects memory allocation.

11 CONFIGURATION TOOL

This section explains how to configure the CCIPC core into a specific CANOpen node.

Each CANOpen node instance is defined by the supported services and by its object dictionary. The protocol services implementation is embedded in the static VHDL core, the object dictionary has to be defined by the final user exploiting the CCIPC configuration tool hereafter simply called CCIPC GUI.

The CCIPC GUI reads, edits and writes out both the standard protocol DCF/EDS files and its specific core files that are the ROM image and VHDL configuration files.

- The VHDL configuration files have to be included in the Core data base before synthesizing and fitting it inside FPGA.
- The ROM image is loaded by CCIPC into its memories at power-on or after restart/reset commands.

The following steps have to be executed to configure and build a new CCIPC instance:

- 1) Start from an existing DCF CANOpen node description, or from scratch, and utilizes the graphical configurator tool to define the new node parameters and object dictionary. Follow the instructions from §Fig. 11-2 to §11.7 set following values
 - Core Interface and clock frequency
 - Configuration objects default values
 - Communication parameters
 - Application Objects
 - Mapping parameters
- 2) Save the new node configuration in DCF (YOURNODE.dcf) format and export it as ROM image txt file (*yourdesign.txt*). Only ROM image and VHDL files can hold the entire CCIPC instance information
- 3) Use the *edac_rom* program to convert ROM_image into EDAC protected numeric format. The ROM_edac program is a C-function that generates an output file containing the EDAC protected image of the input ROM image.
To compile the program use the following command:
> gcc -o edac_rom edac_rom.c -lm
To execute the program type the following command:
> edac_rom yourdesign.txt yourdesign_edac.dat 0
- 4) Copy yourdesign_edac.dat in your SIM directory replacing the dcf.dat file:
> cp yourdesign_edac.dat YourSIM/dcf.dat
- 5) Copy CCIPC_CONF.vhdl VHDL package file in the SRC directory
- 6) If you selected Direct I/O interface copy COM_ROM.vhd, IDX_ROM.vhd, MAP_AO_ITF.vhd files also.

After these steps the new core instance is ready for further simulation, synthesis and fitting design tasks.

11.1 ROM IMAGE FILE

The CCIPC ROM image file holds the default image of the CANOPEN Node object dictionary and the internal index tables utilized by CCIPC to address the implementation specific application objects.

Information is encoded in a verbose textual (*YOURNODE.txt*) format:

- In each line the first 2 hexadecimal strings are interpreted as byte address and memory location content in a 32 bit word aligned arrangement
- “#” indicates the begin of a single line comment
- Constant string indicates the insertion of a VHDL configuration line.

The *edac_rom* program translates the ROM.txt file in the effective memory image file pruning comment/VHDL lines and adding EDAC protection to 32 bit word data.

The ROM image file is composed of the following sections:

- ROM/RAM Pointers Section. The pointers values needed to manage the rest of the ROM image file
- Configuration area OD entries values. A copy of the VHDL CCIPC_Conf.vhd file needed to initialize the configuration area objects
- Index Table. Core specific memory pointer to application objects indexes
- Sub-index Table Core specific memory pointer to application objects sub-indexes
- PDO mapping parameters. Mapping parameters values for each PDO
- PDO communication parameters. Communication parameters values for each PDO
- Application Objects Initialize Values. Default (initialize) values for the application objects

11.2 VHDL Configuration File

The VHDL package configuration collects the VHDL constants used to configure the main aspects of the CCIPC Core like interface/technology and the default values of its configuration area objects.

The following constants are defined in the configuration file:

- HOSI: it defines the HOST interface between:
 - GENERIC_ITF: constant value for Generic IO HOST interface
 - AMBA_AHB: constant value for AMBA HOST interface
 - DIRECT_IO: constant value for DIRECT HOST interface
- TARGET_DEV: it defines the CCIPC core target technology:
 - AXCELERATOR: constant value for Microsemi Axcelerator device
 - PROASIC3: constant value for Microsemi Proasic device
 - VIRTEX: constant value for Xilinx Virtex device
 - GENERIC_TECH: constant for technology independent memory model
- SDO_FUNCTION
 - SDO_EXP_EN: SDO expedited service enable
 - SDO_SEG_EN: SDO expedited and segmented enable
 - SDO_BLOCK_EN: SDO expedited and block enable
- NODE_COUNT: number of supported Node
- PDO_IRQCNT: maximum number of R(T)PDO IRQ supported
- PDO_IRQIDX: number of bit used to select R(T)PDO IRQ line ($\log_2(\text{PDO_IRQCNT})$);
- NODEID: CCIPC node ID. Tthis value is used only in simulation mode. During real mode this value is triggerd using the dedicated external pins. (Index:2002h, SubIndex:01h)
- NODEMASK: CCIPC node mask. Tthis value is used only in simulation mode. During real mode this value is triggerd using the dedicated external pins(Index:2002h, SubIndex:02h).
- DEV_TYPE: Device type constant (Index: 1000h)
- ERR_REG: error register default value (Index: 1001h)
- ID_SYNCH: Synch Object COB-ID constant value (Index: 1005h)
- SYNCH_WLEN: default value of synchronous windows length (Index: 1007h)
- MSTID_CONSHB: constant value of Heartbeat consumer time (index:1016h):
 - MST_NODEID: it defines the Node-ID of the Master
 - CONS_HB: it defines the value of the heartbeat time
- PROD_HB: constant value of Heartbeat producer time (index: 1017h)
- VENDOR_ID: constant value of the Vendor ID (index:1018h, SubIndex:01h)
- PRODUCT_CODE: constant value of the Product Code (index:1018h, SubIndex:02h)
- REV_NUM: constant value of the Revision number (index:1018h, SubIndex:03h)
- SERIAL_NUM: constant value of the Serial number (index:1018h, SubIndex:04h)
- CS_SDO_ID: constant value of the SDO Client-Server COB-ID (index:1200h, Sub-Index:01h)
- SC_SDO_ID: constant value of the SDO Server-Client COB-ID (index:1200h, Sub-Index:02h)
- B_DEF: constant value of Default bus parameter (index:2000h, Sub-Index:01h)
- T_TOGGLE: constant value of Ttoggle parameter (index:2000h, Sub-Index:02h)
- N_TOGGLE: constant value of Ntoggle parameter (index:2000h, Sub-Index:03h)

- HuCANE_Cfg: default value of the HurriCANE configuration (Index:2003h, Sub-Index:01h)
 - RSJ_VAL: RSJ default value
 - PS2_VAL: PS2 default value
 - PS1_VAL: PS1 default value
 - BPR_VAL: BPR default value
- CCIPC_ST: default value of the CCIPC status register (Index:2003h, Sub-Index:02h)
 - AUTO_OP: defines the possibility of the CCIPC to enter automatically in Operational state at the end of the initialization phase
- EDAC_REG_VAL: define the default value of the EDAC Controller register (Index:2003h, Sub-Index:07h)
- MAX_TPDO: defines the maximum number of TPDO supported by CCIPC
- MAX_RPDO: defines the maximum number of RPDO supported by CCIPC
- MAX_ROAO: defines the maximum number of read only AOs supported by CCIPC
- MAX_RWDO: defines the maximum number of read write AOs supported by CCIPC
- MAX_SRPDO: length of Synch message queue

11.3 CCIPC GUI

This section contains instructions concerning the usage of CCIPC Configuration tool. The CCIPC GUI leads the user to define a specific Object Dictionary layout.

To launch the CCIPC core configuration tool set the following environment variable (for example with cshell).

- `setenv CCIPC_ROOT YOUR+CCIPC_DATABASE_PATH`
- `set path = ($path CCIPC_ROOT/SIM_SCRIPT CCIPC_ROOT/CONFIG_TOOL)`

Then type the command:

- `CCIPC_gui.pl`

The following naming rules have to be observed to correctly use the CCIPC GUI:

- Only letters [A-Z, a-z], numbers [0-9] and underscore[_] are accepted;
- First character of each name has to be a letter[A-Z, a-z];
- Space [] are interpreted as name delimiter and should be never used in CCIPC GUI forms;

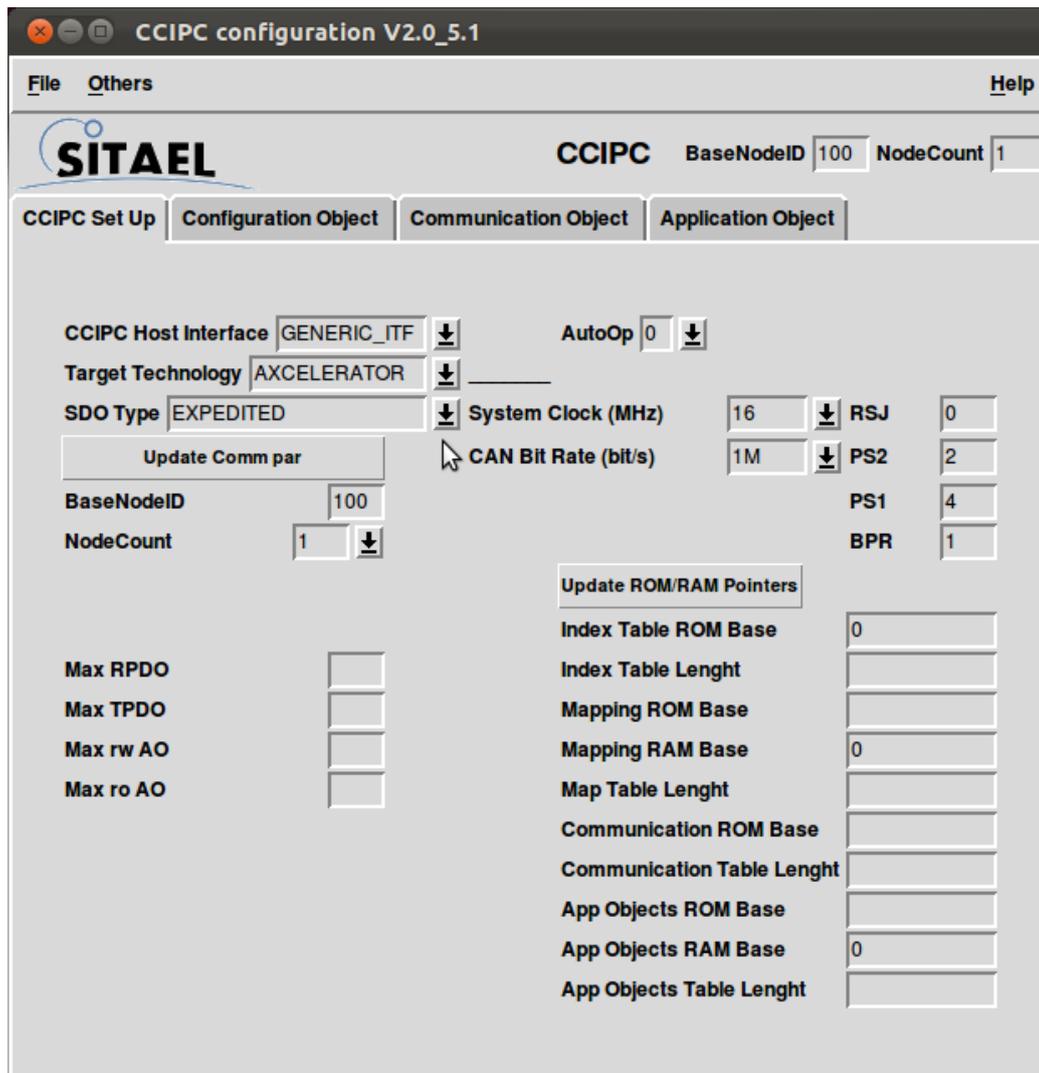


Fig. 11-1: CCIPC GUI.

The main window is composed of four tabs:

- CCIPC Set Up window that allows setting the main CAN and CANOpen parameters;
- Configuration Object window allows setting the parameters available in the Configuration Area;
- Communication Object window allows define the RPDO and TPDO communication and mapping parameters;
- Application Object window allows defining the Application Object for the node

The File menu is composed of the following options:

- Open: open a DCF file
- Save: save the current Object Dictionary layout in a DCF format
- Import: create the Object Dictionary layout reading the ROM image file (.txt)
- Export: create the ROM image file(.txt) of the actual Object Dictionary layout and the CCIPC VHDL configuration file.

11.4 CCIPC Set Up Window

The first step to edit the Object Dictionary structure is to define the general CCIPC core parameters using the dedicated CCIPC Set Up window.

The CCIPC Host Interface scroll bar allows selecting the core Host Interface between:

- Generic I/O interface
- AMBA interface
- DIRECT interface

The Target Technology scroll bar allows selecting the CCIPC technology implementation of the RAM modules between:

- Axcelerator
- ProAsic
- Virtex
- Generic

The SDO type scroll bar allows defining SDO services supported by CCIPC between:

- Expedited
- Segmented

- Block

The SDO Expedited service is active also when Segmented or Block service is selected. The CAN Bus transfer rate parameters can be defined through the following section:

Fig. 11-2: Frequency setting.

The CAN core RSJ,PS2,PS1 and BPR parameters are automatically calculated by the GUI once the system frequency and the CAN bit Rate values are entered.

Fig. 11-3: Node ID setting.

The CCIPC Node parameters are used to define the Base Node ID and Node count of the CCIPC core and these are used to automatically generate the default values of the OD entries dependent on the Node ID. The Node count value can be selected between 1,2,4,8,16 and 32 predefined values.

Because of the CCIPC core Node ID is also defined using dedicated external pins, it's recommended that the selected GUI values reflect the external configuration.

The **AutoOp** option defines the behaviour of the Network Management state machine at the end of the Initialisation phase:

- '1' means that CCIPC core enters in Operational State
- '0' means that CCIPC core enters in Pre-Operational State

The **Update Comm par** button, when pressed, creates a default OD structure, updating the Configuration and the Communication Objects using the information defined in this window.

The CCIPC external and internal memory organization of the core is available in the bottom right side of the window and it contains the following information:

- ROM base address
- Index table word length
- Communication, Mapping and Application Object ROM and RAM base address and word length

Fig. 11-4: CCIPC memory settings.

Information about the maximum index defined for RPDO,TPDO, RW and RO Application Objects are also visible in the window.

Max RPDO	1407
Max TPDO	1807
Max rw AO	6005
Max ro AO	

Fig. 11-5: Communication and Application Object settings.

11.5 Configuration Object Window

The Configuration Object window, illustrated in Fig. 11-6, allows setting the OD entries defined in the Configuration area and reported in Tab. 10-2 of §10.1.



Fig. 11-6: Configuration Object window.

Each object value can be modified using the **Edit Var** or **Edit Subindex** button.
 To modify a VAR object, select the specific Object and press the **Edit Var** button.

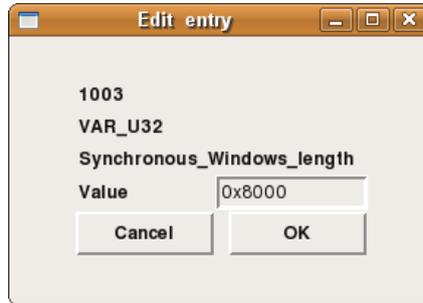


Fig. 11-7: Edit Var window.

Index	SubIndex	Register Name	Obj Type	Default val	Note
1000h		Device Type	VAR_U32	0	Don't edit
1001h		Error register	VAR_U8	0	Don't edit
1005h		COB-ID synch	VAR_U32	80h	Don't edit. The ID of the synch message must be constant to 80h
1007h		Synchronous Window length	VAR_U32	0	
1016h	0h	Heartbeat consumer time	U8	1	Don't edit
	1h	Master Consumer Heartbeat time	U32	0	
1017h		Producer Heartbeat time	U16	0	
1018h	0h	Identity object	U8	4	Don't edit
	1h	Vendor-ID	U32	0	Don't edit
	2h	Product code	U32	0	Don't edit
	3h	Revision number	U32	0	Don't edit
	4h	Serial Number	U32	0	Don't edit
1200h	0h	Server SDO	U8	2	Don't edit
	1h	COB-ID client-server	U32		
	2h	COB-ID server-client	U32		
2000h	0h	Redundancy Management	U8	4	Don't edit
	1h	Bdefault	U8	0	
	2h	Ttoggle	U8	0	
	3h	Ntoggle	U8	0	
	4h	Ctoggle	U8	0	
2002h	0h	Node parameters	U8	2	Don't edit
	1h	ID base	U8	0	
	2h	ID mask	U8	0	
2003h	0h	CCIPC status & configuration	U8	7	Don't edit
	1h	HurriCANE configuration	U32	0	

Index	SubIndex	Register Name	Obj Type	Default val	Note
	2h	HurriCANE & CCIPC status	U32	0	
	3h	CCIPC-SDO error	U32	0	
	4h	IRQ status	U32	0	
	5h	IRQ mask-clear	U32	0	
	6h	RPDO IRQ0	U32	0	
	7h	RPDO IRQ1	U32	0	
	8h	RPDO IRQ2	U32	0	
	9h	RPDO IRQ3	U32	0	
	Ah	TPDO IRQ0	U32	0	
	Bh	TPDO IRQ1	U32	0	
	Ch	TPDO IRQ2	U32	0	
	Dh	TPDO IRQ3	U32	0	
	Eh	EDAC error	U32	0	
	Fh	TPDO trig	U32	0	
	10h	SDO multiplexor	U32	0	

Tab. 11-1 List of Configuration Object.

In the value field, enter the new object value and press **OK** to update it.

To modify an Array or Record entry type, select the specific object and press **Load Subindex** button. In the lower window all the sub-indexes defined for the object are displayed. To modify the single sub-index, select it and click on the Edit Subindex button.

In the value field, enter the new object value and press **OK** to update it.

11.6 Application Object

The Application Object window, shown in Fig. 11-8, allows setting the CCIPC core Application Objects.

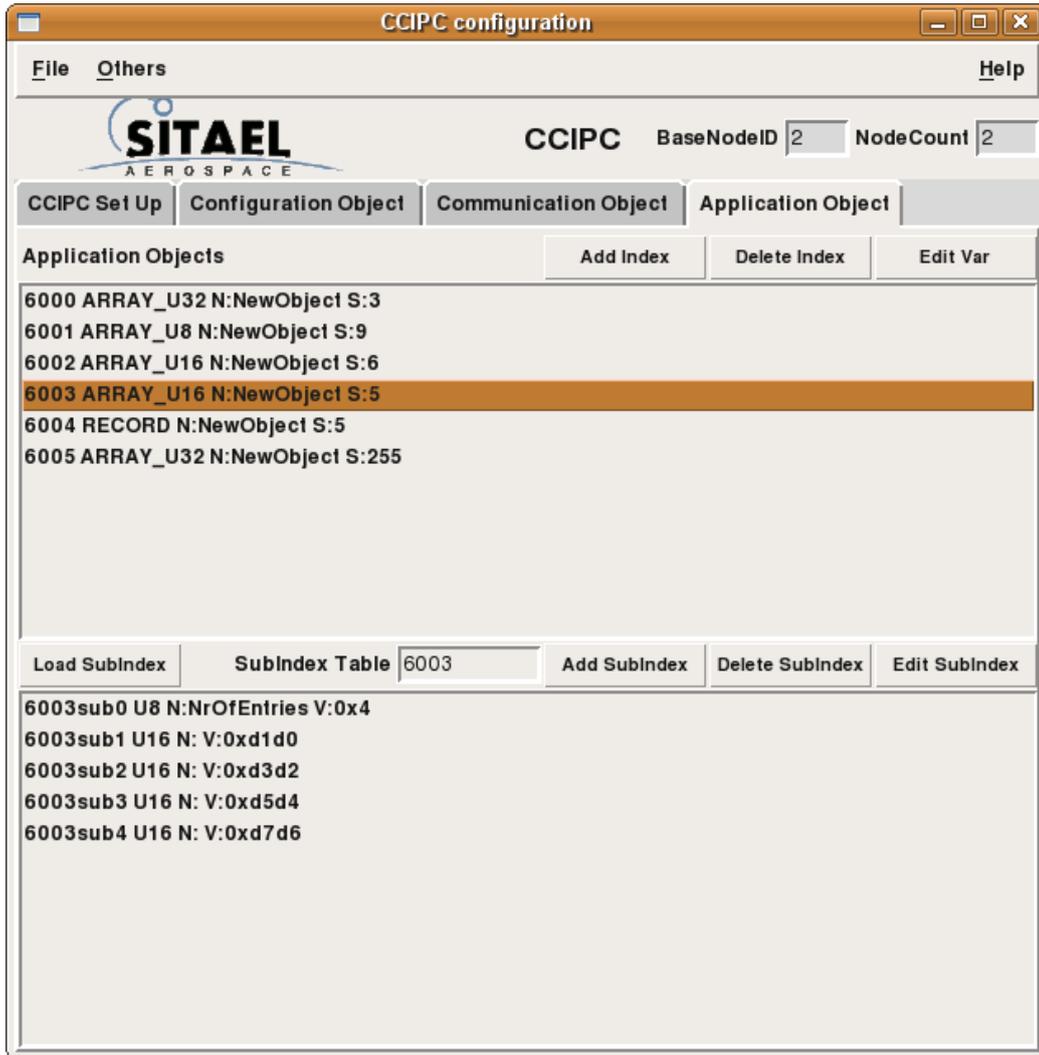


Fig. 11-8: Application Object window.

To insert a new Application Objects click on **Add Index** button.

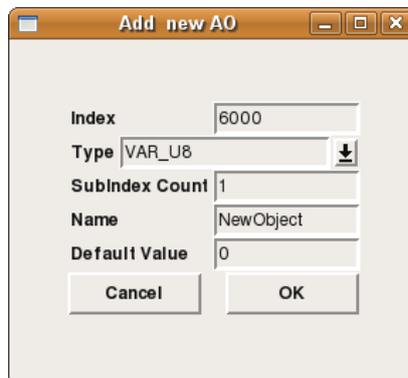


Fig. 11-9: New AO Index definition.

In the new window the following parameters can be set:

- Index: it defines the Index value for the new AO
- Type: it defined the AO type. This field can be:
 - VAR_U8
 - VAR_U16
 - VAR_U32
 - RECORD
 - ARRAY_U8
 - ARRAY_U16
 - ARRAY_U32
- SubIndex Count: number of sub-indexes supported by the entry.
- Name : name of the entry.
- Default value: reset value.

Press **OK** button to accept the Index settings.

If Index is Array or Record type, next step is to define each sub-index that composes the entry.

Select the desired Index and click on **Load SubIndexes** button to load the entry structure. To edit a single sub-index, select it and click on **Edit SubIndex**. To add a new sub-index click on **Add SubIndex** button.

The Sub-index0 value is automatically updated every time a new sub-index is defined.

The characteristic of the opened window depends on the HOST interface selected in the Configuration Object window.

The Fig. 11-10 shows the window in case of Generic I/O or AMBA interface. The sub-index parameters are:

- Type: it defines the sub-index data type between:
 - U8
 - U16
 - U32
- Name : it defines the name of the sub-index.
- Value : it defines the reset value
-



Fig. 11-10: AMBA-Generic I/O AO sub-index definition.

If DIRECT interface is selected the window that appears is shown in the figure below

The screenshot shows a dialog box titled "Edit entry" with a standard Windows-style title bar (minimize, maximize, close buttons). The dialog content is as follows:

- Header: 6000 sub 1
- Type: U8 (dropdown menu)
- Name: Last_Entry1 (text field)
- Value: 0 (text field)
- Port Assignment: OutDFF (dropdown menu)
- Buttons: Cancel and OK

Fig. 11-11: Direct - AO sub-index definition.

The **Port Assignment** field defines how the AO is defined in the port mapping. The following type are available:

- OutDFF: it defines a Dflip-flop output port
- InDFF: it defines a Dflip-flop input port
- InOutDFF: it defines a bidirectional Dflip-flop port

11.7 Communication Object

The Communication Object Window, shown in Fig. 11-12, allows defining the Communication and Mapping parameters for each RPDO and TPDO.

When a new OD structure has been defined, all the RPDO and TPDO are configured in not existing mode and no mapping parameter is defined.

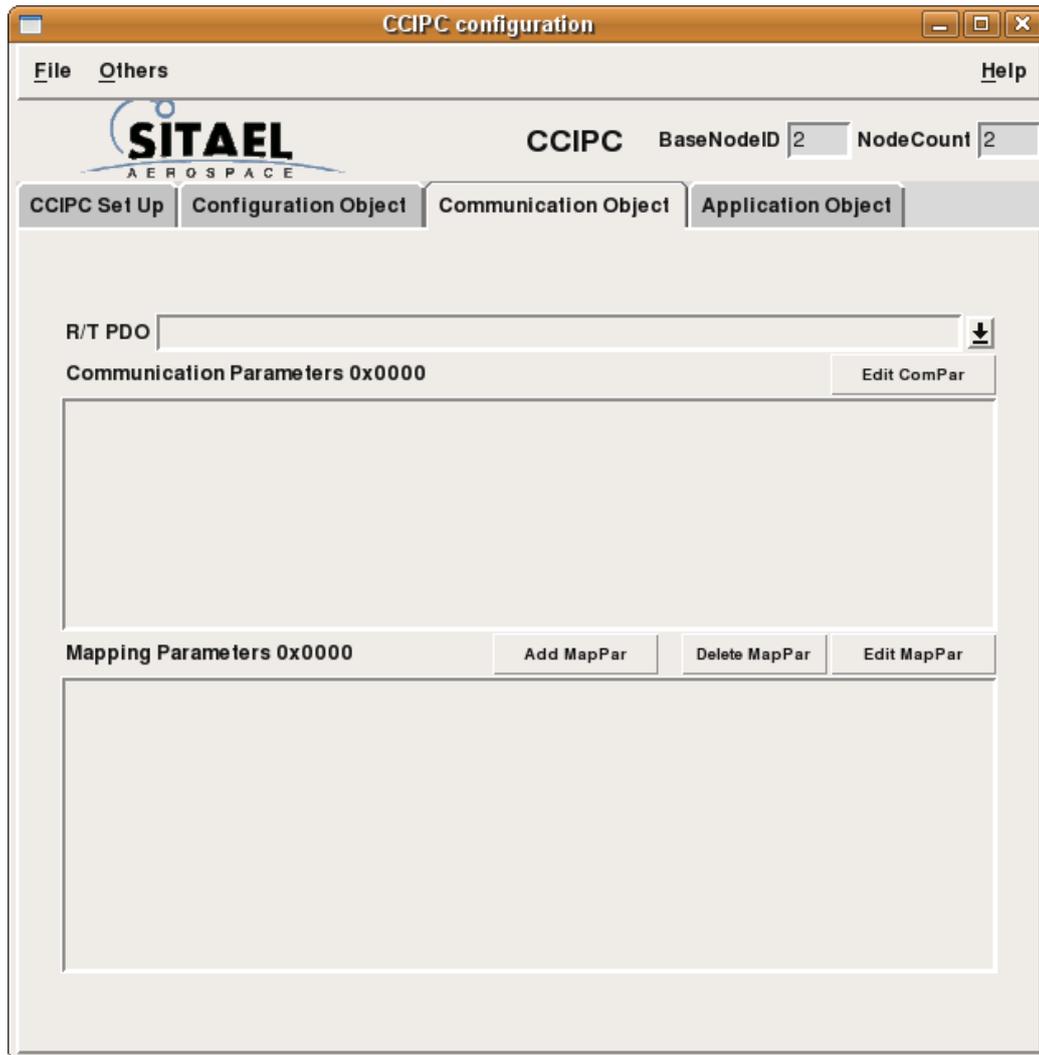


Fig. 11-12: Communication Object window.

The **R/T PDO** scroll bar lists all the RPDO and TPDO that are supported by the CCIPC core. To modify a specific RPDO object, select it from the scroll bar menu. In the two text areas below the Communication and Mapping parameters values are displayed.

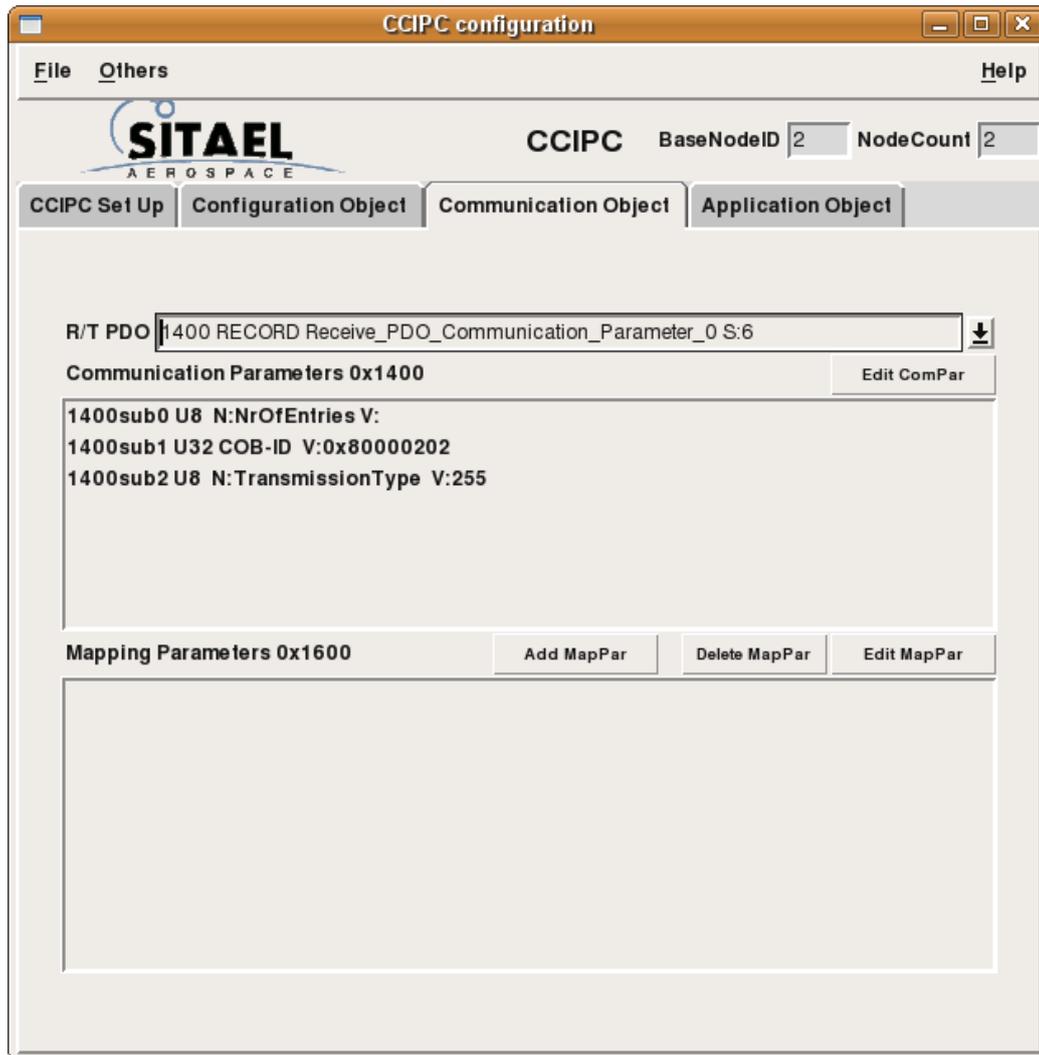


Fig. 11-13: RPDO settings.

The Communication parameters fields are:

- **COB-ID:** define the RPDO COB-ID. Modify only the last bit (31) of this field in order to select if the RPDO exists or not.. The ID part of the field (bit 10 to 0) doesn't affect the behaviour of the CCIPC core because it works with the default COB-ID assignment and using the external Node-ID information;
- **Transmission Type:** it defines the RPDO transmission type. Asynchronous RPDOs must be indicated with 0xFF(255) value. Other values are for synchronous RPDO.

To configure new mapping parameters, press the **Add MapPar** button.

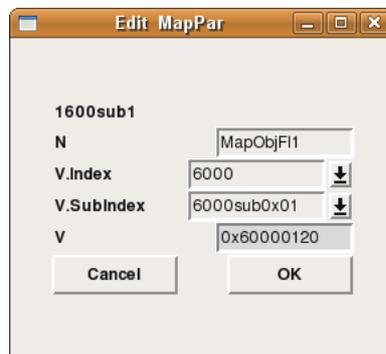
For the SubIndex0, no value(V field) has to be inserted because this value is automatically updated when new sub-indexes are defined. Only the sub-index name (N field) can be modified.



1600sub0
N MapObjFI_0
V 0
Cancel OK

Fig. 11-14: Mapping Parameter - Subindex0.

For other sub-indexes, the definition of the mapped Application Objects has done using the scroll bar menu which lists all the available Application Objects, as shown in Fig. 11-15. To define a new Application Objects see §11.6.



1600sub1
N MapObjFI1
V.Index 6000
V.SubIndex 6000sub0x01
V 0x60000120
Cancel OK

Fig. 11-15: RPDO Mapping parameter definition.

The **V.Index** allows selecting the mapping Index between the available AO Index, while the **V.SubIndex** lists all the sub-indexes supported by the selected Index.

The actual mapping value is shown in the **V** field.

To modify a specific TPDO object, select it from the scroll bar menu. In the two text areas below the Communication and Mapping parameters values are displayed.

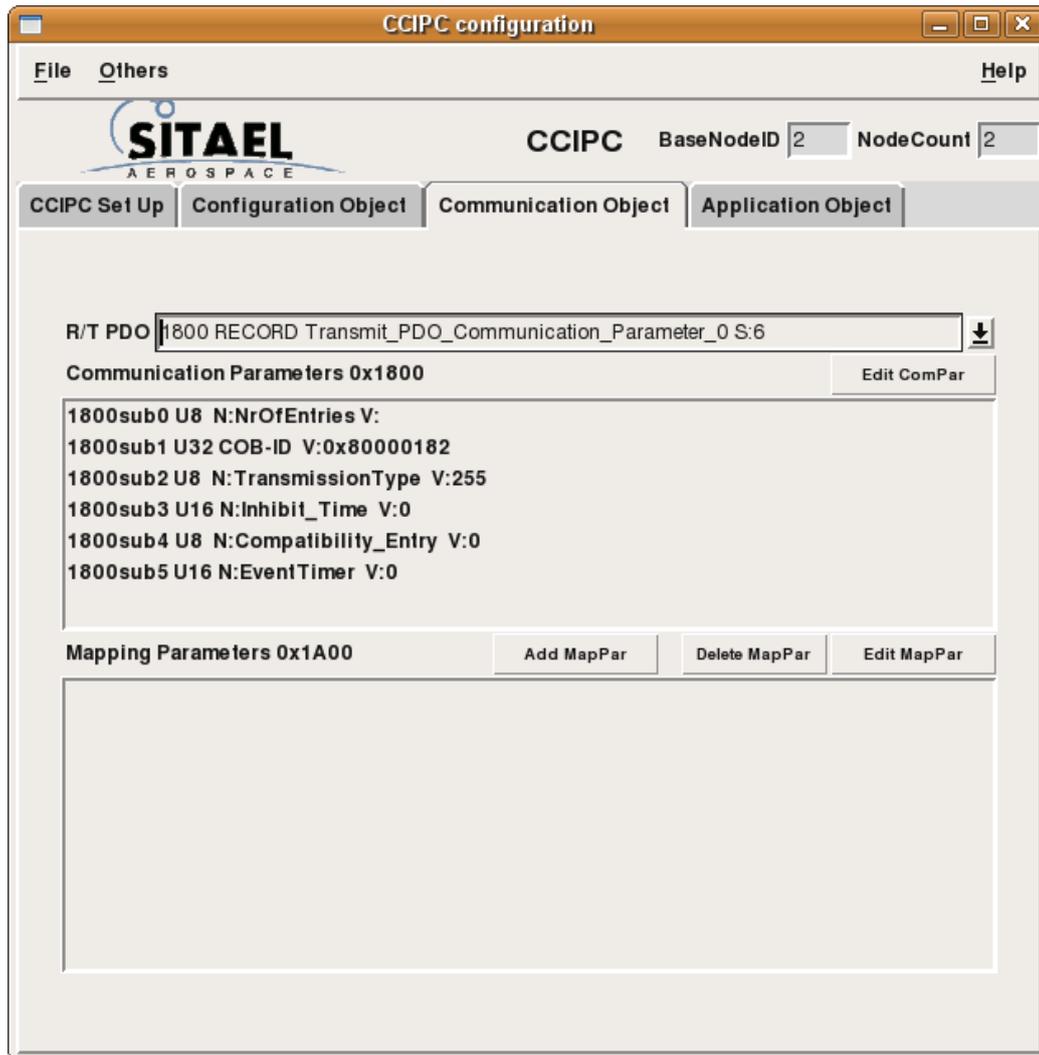


Fig. 11-16: TPDO settings.

The Communication parameters fields are:

- **COB-ID:** define the TPDO COB-ID. Modify only the last bit(31) of this field in order to select if the TPDO exists or not.
- **Transmission Type:** it defines the TPDO transmission type. Asynchronous TPDOs must be indicated with 0xFF(255) value. Other values are for synchronous TPDO.
- **Inhibit Time:** it defines the inhibit time as multiple of 100us for the TPDO. Because the CCIPC updates this value every 1ms, insert a value with this granularity. For inhibit time of 2ms, insert 0x14 (2ms/100us=20).
- **Event Timer;** it defines the TPDO event time as multiple of 1ms.

To configure new mapping parameters, press the **Add MapPar** button.

For the SubIndex0, no value(V field) has to be inserted because this value is automatically updated when new sub-indexes are defined. Only the sub-index name (N field) can be modified.



Fig. 11-17: Mapping Parameter - Subindex0.

For other sub-indexes, the definition of the mapped Application Objects has done using the scroll bar menu, which lists all the available Application Objects. To define a new Application Objects see §11.6.

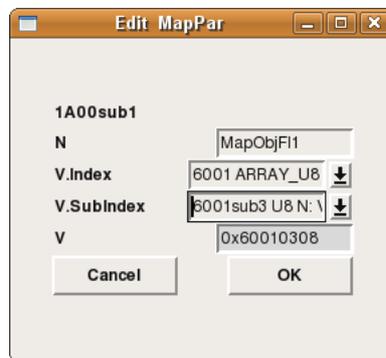


Fig. 11-18: TPDO Mapping parameter definition.

The **V.Index** allows selecting the mapping Index between the available AO Index, while the **V.SubIndex** lists all the sub-indexes supported by the selected Index. The actual mapping value is shown in the **V** field.

12 CCIPC RESOURCE OCCUPATION

In this paragraph, the synthesis reports for different CCIPC configuration and FPGA technologies are reported. The results are obtained with the following CCIPC configuration:

- 2 Node-Id supported;
- 8 RPDOs and 8 TPDOs;
- 6 Application Objects:
 - Array of U32 with 2 sub-indexes;
 - Array of U8 with 8 sub-indexes;
 - Array of U16 with 5 sub-indexes;
 - Record with 4 sub-indexes;
 - Array of U32 with 254 sub-indexes;
- Consumer Heartbeat Time = 4ms;
- Producer Heartbeat Time = 255ms;
- Ttoggle = 255;
- Ntoggle = 2;

The following table reports the number of internal FPGA RAM modules (512x8 bits) used by CCIPC. The occupation of the Communication Parameter Table module (Comm Table) depends on the number of node supported that determines the number of RPDO and TPDO that have to be instantiated.

Node Count	Modules					Tot
	MsgIn queue	MsgOut queue	Comm Table	Index Table	Mem RF	
1	1	1	5(*)	5(*)	5(*)	17
2	1	1	5(*)	5(*)	5(*)	17
4	1	1	5(*)	5(*)	5(*)	17
8	1	1	5(*)	5(*)	5(*)	17
16	1	1	10(**)	10(**)	5(*)	27
32	1	1	15(***)	10(**)	5(*)	32
(*) $4 \cdot (512 \cdot 8) = 512 \cdot 32$ of data + $512 \cdot 8$ of EDAC						
(**) $2 \cdot (4 \cdot (512 \cdot 8)) = 1024 \cdot 32$ of data + $2 \cdot (512 \cdot 8) = 1024 \cdot 8$ of EDAC						
(***) $3 \cdot (4 \cdot (512 \cdot 8)) = 1536 \cdot 32$ of data + $3 \cdot (512 \cdot 8) = 1536 \cdot 8$ of EDAC						

Tab. 12-1 CCIPC RAM occupation

For Xilinx device *Combinational* field refers to Number of 4 input LUTs, *Sequential* field refers to Number of Slice Flip Flops and *Total* field refers to total number of slices occupied.

Tables Tab. 12-2, Tab. 12-3 and Tab. 12-4 furnish synthesis reports of CCIPC in its Generic interface when Segmented, Block and Expedited SDO are activated.

Family	Device	Cells			RAM	Utilization	Fmax (MHz)
		Combinational	Sequential	Total			
Axcelerator	RTAX1000S	7807	1775	9582	17	52.8%	19.9
ProAsic	A3PE3000	12785	1779	14564	17	19.35%	20.9
Xilinx	XC4VLX25	7106	1701	3930	17	36%	34.2

Tab. 12-2: CCIPC Segmented - Area Occupation.

Family	Device	Cells	RAM	Utilization	Fmax
--------	--------	-------	-----	-------------	------

		Combinational	Sequential	Total			(MHz)
Axcelerator	RTAX1000S	7823	1785	9608	17	52.95%	19
ProAsic	A3PE3000	13216	1772	14988	17	19.91%	20.7
Xilinx	XC4VLX25	7203	1702	3976	17	36%	31.8

Tab. 12-3: CCIPC Block - Area Occupation.

Family	Device	Cells			RAM	Utilization	Fmax (MHz)
		Combinational	Sequential	Total			
Axcelerator	RTAX1000S	7493	1764	9257	17	51.02%	21
ProAsic	A3PE3000	12320	1782	14102	17	18.74%	19.6
Xilinx	XC4VLX25	6747	1699	3749	17	34%	34.2

Tab. 12-4: CCIPC Expedited - Area Occupation.

Table Tab. 12-5 furnishes synthesis reports of CCIPC in its AMBA configuration when SDO Block is selected.

Family	Device	Cells			RAM	Utilization	Fmax (MHz)
		Combinational	Sequential	Total			
Axcelerator	RTAX1000S	8182	2174	10356	17	57%	17.9
ProAsic	A3PE3000	13480	2223	15703	17	20.8%	19.4
Xilinx	XC4VLX25	7273	1686	4053	17	37%	29.8

Tab. 12-5: CCIPC AMBA Block - Area Occupation.

Next table furnishes synthesis reports of CCIPC in its Direct configuration. The results are obtained with the following CCIPC configuration:

- 1 Node-Id supported;
- 4 RPDOs and 4 TPDOs;
- 1 Application Objects with 3 sub-indexes of U32:
 - 2 defined as Output;
 - 1 defined as Input
- Consumer Heartbeat Time = 100ms;
- Producer Heartbeat Time = 100ms;
- Ttoggle = 3;
- Ntoggle = 3;
-

Family	Device	Cells			RAM	Utilization	Fmax (MHz)
		Combinational	Sequential	Total			
Axcelerator	RTAX1000S	7112	1720	8832	7	48.6%	19.9
ProAsic	A3PE3000	11703	1762	13465	7	17.89%	20.05
Xilinx	XC4VLX25	6420	1623	3585	7	33%	34

Tab. 12-6: CCIPC Direct- Area Occupation.

13 CCIPC TIMING CHARACTERISTICS

The CCIPC timings characterization is reported in next tables and it gives an estimation of elaboration time in terms of clock cycles of the main CANopen features.

All the timings are characterized without considering congestions on CAN network and performing one task at time avoiding any processing delays at CCIPC system level.

The CCIPC presents an internal scheduler that activates the elaboration of the main CANopen services with the following priorities:

- Asynchronous Event trigger TPDO → high priority;
- SDO requests, asynchronous RPDO and SYNC routine (Synchronous PDOs);
- SDO Block Upload;
- Asynchronous Timer Driven TPDO → low priority.

When CCIPC is elaborating a task the others have to wait until the end of scheduled task causing a possible delay on expected elaboration time.

All timing values are measured using a 16 MHz clock frequency.

Application Objects Elaboration time

The following table reports the elaboration of the three types of Application Objects (U8, U16 and U32) associated to RPDO, TPDO and SDO services.

PDOs elaboration time includes both decoding of mapping parameter and memory access, while SDO elaboration time reports only memory access time.

Parameters		Elaboration time	Note
PDO - Read U8	T_{PrdU8}	204	Time to read any U8 AO
PDO - Read U16	T_{PrdU16}	280	Time to read any U16 AO
PDO - Read U32	T_{PrdU32}	416	Time to read any U32 AO
PDO – Write U8	T_{PwrU8}	233	Time to write any U8 AO
PDO – Write U16	T_{PwrU16}	328	Time to write any U16 AO
PDO – Write U32	T_{PwrU32}	498	Time to write any U32 AO
SDO – Read U8	T_{Srd}	62	Time to read single byte
SDO – Write U8	T_{Swr}	86	Time to write single byte

SDO timings are affected by the value of the Index and Sub-Index multiplexor fields. The following two parameters report the time needed to point correct Index and Sub-Index.

Parameters		Elaboration time	Note
SDO – Index Pointer	T_{idx}	$I*36$	Time to point correct Index
SDO – Sub-Index Pointer	T_{sidx}	$S*54$	Time to point correct Sub-Index
$I = ((\text{Index}-0x6000)+1)$ if index is in the Read-Write area; $I = ((\text{Index}-0x7000)+1)$ if index is in the Read-Only area; $S = \text{Sub-Index}$			

Heartbeat time

Parameters	Elab. time	Note
Heartbeat producer	$(\text{HBP}) * 1\text{ms}$	CCIPC guarantees the correct management of Heartbeat producer time with granularity of 1ms.
Heartbeat consumer counter reset	$[\text{HBC}*1\text{ms}-1\text{ms}; \text{HBC}*1\text{ms}]$	The reception of Heartbeat resets the Heartbeat Consumer timer. In the worst case the mismatch between the expected Heartbeat expiration and the real Heartbeat expiration is 1ms at maximum.

Synchronous TPDO

CCIPC scans all available TPDOs during elaboration of Synchronous TPDO. Each TPDO is characterized by four parameters:

- COB-ID;
- Transmission type
- Inhibit Time (not supported for Synchronous TPDO)
- Event Time (not supported for Synchronous TPDO)

Parameters		Elaboration time	Note
COB-ID	T_{id}	26	Time to check TPDO COB-ID
Transmission type	T_{trx}	32	Time to check TPDO Transmission type (including the update of synchronous counter)
Inhibit time	T_{inh}	NA	NA
Event time	T_{evn}	NA	NA
TPDO with 3 mapping (1*U8, 1*U16, 1*U32)			
$T_{tot} = T_{id} + T_{trx} + T_{PRdU8} + T_{PRdU16} + T_{PRdU32} + T_{elab} = 26+32+204+280+416+150=1108$			

When a Synch message is received, the CCIPC elaborates received Synchronous RPDOs before processing the TPDO ones.

Synchronous RPDO

Parameters		Elaboration time	Note
Read Message	T_{msg}	146	Time to read message form Synch queue
Length check	T_{lck}	43	Time to check RPDO vs CAN message byte length
ASynch RPDO with 3 mapping (1*U8, 1*U16, 1*U32)			
$T_{tot} = T_{msg} + T_{lck} + T_{PwrU8} + T_{PwrU16} + T_{PwrU32} + T_{elab} = 146+43+233+328+498+50=1298$			

Synchronous Window lenhgth

CCIPC guarantees the expiration of Synchronous Window length according to value defined in correspondent Object Dictionary entry (1007h).

The indication of synchronous elaboration error (synchronous RPDO or TPDO not managed) is available on the Host interface (IRQ[2]) when synchronous elaboration phase is completed.

Timer Driven Asynchronous TPDO

CCIPC scans all available TPDOs during elaboration of Timer driven Asynchronous TPDO. Each TPDO is characterized by four parameters:

- COB-ID;
- Transmission type
- Inhibit Time
- Event Time

Parameters		Elaboration time	Note
COB-ID	T_{id}	26	Time to elaborate an invalid TPDO
Transmission type	T_{trx}	32	Time to elaborate a TPDO defined as Synchronous
Inhibit time	T_{inh}	65	Time to elaborate the Inhibit Time
Event time	T_{evn}	99	Time to elaborate the Event Time
TPDO with 3 mapping (1*U8, 1*U16, 1*U32):			
$T_{tot} = T_{id} + T_{trx} + T_{inh} + T_{evn} + T_{PRdU8} + T_{PRdU16} + T_{PRdU32} + T_{elab} = 26+32+65+99+204+280+416+120=1242$			

Event Driven Asynchronous TPDO

To elaboration Event driven Asynchronous TPDO, CCIPC has to valuate:

- COB-ID;

- Transmission type
- Inhibit Time
- Update Event Timer (if supported)

In next table the elaboration time of each parameters is indicated:

Parameters		Elaboration time	Note
COB-ID	T_{id}	26	Time to check TPDO COB-ID
Transmission type	T_{trx}	32	Time to check TPDO Transmission type
Inhibit time	T_{inh}	50	Time to elaborate the Inhibit Time
Event Time	T_{evn}	99 (40)	Time to control and reset event time (event timer not supported).
TPDO with 3 mapping (1*U8, 1*U16, 1*U32): $T_{tot} = T_{id} + T_{trx} + T_{inh} + T_{evn} + T_{PRdU8} + T_{PRdU16} + T_{PRdU32} + T_{elab} = 26+32+50+99+204+280+416+120=1168$			

Asynchronous RPDO

Parameters		Elaboration time	Note
Read Message	T_{msg}	75	Time to read message form queue
COB-ID	T_{id}	26	Time to check RPDO COB-ID
Transmission type	T_{trx}	32	Time to check RPDO transmission type
Length check	T_{lck}	43	Time to check RPDO vs CAN message byte length
ASynch RPDO with 3 mapping (1*U8, 1*U16, 1*U32) $T_{tot} = T_{msg} + T_{id} + T_{trx} + T_{lck} + T_{PwrU8} + T_{PwrU16} + T_{PwrU32} + T_{elab} = 75+26+32+43+233+328+498+50=1285$			

SDO Expedited

Parameters		Elaboration time	Note
SDO Download – Elaboration Time	T_{Delab}	1063	SDO Download Expedited elaboration time
SDO Upload – Elaboration Time	T_{Uelab}	900	SDO upload Expedited elaboration time
Download Index IDX, Sub-Index S , U32: $T_{tot} = T_{idx} + T_{sidx} + 4*T_{WrB} + T_{Delab} = I*36 + S*54 + 4*86 + 1063 = 1406 + I*36 + S*54$			
Upload Index Idx, Sub-Index S , U32: $T_{tot} = I*T_{idx} + S*T_{sidx} + 4*T_{RdB} + T_{Uelab} = I*36 + S*54 + 4*62 + 900 = 1148 + I*36 + S*54$			

As shown in table above the SDO expedited elaboration time is mainly affected by the check of received packet rather than data processing.

SDO Download Segmented

Parameters		Elaboration time	Note
Initiate request	T_{SDwIn}	$1086 + T_{idx} + T_{sidx}$	Elaboration of Initiate Download request
Download request	T_{SDw}	$871 + n*T_{Swr}$	Elaboration of Download request (n byte)

SDO Upload Segmented

Parameters		Elaboration time	Note
Initiate request	T_{SUpln}	$1120 + T_{idx} + T_{sidx}$	Elaboration of Initiate Upload request
Upload request	T_{SUpl}	$1032 + n*T_{Swr}$	Elaboration of Upload request (n byte)

SDO Download Block

Parameters		Elaboration time	Note
Initiate request	T_{BDwIn_min}	$= T_{elab} + T_{idx} + T_{sidx}$ $T_{elab} = 1218$	Elaboration of Initiate Download request if Entry size is greater than 889 byte (127 segment threshold). Block size is set to 127.

Parameters	Elaboration time	Note
	$T_{BDwn_max} = T_{elab} + T_{idx} + T_{sidx} + T_{blksize};$ $T_{elab} = 1218$ $T_{blksize} = (size/7)*348$	Elaboration of Initiate Download request if Entry size is less than 889 byte (127 segment threshold). Timing value is affected by time requested to calculate correct Block size.
Download Segment request	$T_{BDw} = T_{elab} + 7*(T_{Swr} + T_{upd})$ $T_{elab} = 160$ $T_{upd} = 42$	Elaboration of Download request (7 byte)
Download Segment response	$T_{BDwR} = T_{elab} + n*(T_{Swr} + T_{upd})$ $T_{elab} = 267$ $T_{upd} = 42$	Elaboration of Last segment of a block. If last segment has been received with n byte.
	$T_{BDwLBlk} = T_{elab} + 7*(T_{Swr} + T_{upd}) + T_{blksize};$ $T_{elab} = 1218$ $T_{upd} = 42$ $T_{blksize} = (size/7)*348$	Elaboration of Last segment of a block. If another block has to be received. Timing value is affected by time requested to calculate correct remaining Block size.
End Block Request	$T_{BDwEnd} = 502$	Elaboration of End SDO Block Download request

SDO Upload Block

Parameters	Elaboration time	Note
Initiate request	$T_{BUpIn_min} = T_{elab} + T_{idx} + T_{sidx}$ $T_{elab} = 1254$	Elaboration of Initiate Upload request. If Entry size is greater than 889 byte (127 segment threshold). Block size is 127.
	$T_{BUpIn_max} = T_{elab} + T_{idx} + T_{sidx} + T_{blksize};$ $T_{elab} = 1254$ $T_{blksize} = (size/7)*348$	Elaboration of Initiate Upload request. If Entry size is less than 889 byte (127 segment threshold). Timing value is affected by time requested to calculate and check correct Block size.
Start request	$T_{BUpStart} = 187$	Elaboration of Upload Start request
Upload segment request	$T_{BUp} = T_{elab} + 7*(T_{Srd} + T_{upd})$ $T_{elab} = 195$ $T_{upd} = 42$	Elaboration of 1 Upload segment (7 byte)
Block segment response	$T_{BUpLBlk} = 346$	Elaboration of a block reply. Last block sent.
	$T_{BUpBlkR} = T_{elab} + T_{blksize};$ $T_{elab} = 1304$ $T_{blksize} = (size/7)*348$	Elaboration of a block reply. Another block has to be sent. Timing value is affected by time requested to calculate and check correct remaining Block size.
End Block request	$T_{BUpEnd} = 222$	Elaboration of End SDO Upload Block request

13.1 TPDO Numerical Example

The critical tasks for the CCIPC core is the time needed to elaborate asynchronous and synchronous TPDOs. Every millisecond and every synchronous message received, the CCIPC core has to scan all the TPDOs supported in order to establish which TPDO has to be sent.

The following paragraphs report numerical examples of the time needed to elaborate TPDOs for all the possible value of CCIPC Node supported (32,16,8,4,2,1).

The timing parameters (in milliseconds) reported are:

1. **Ta** that represents:
 - Elaboration time to elaborate one group of Asynchronous TPDOs when Inhibit time is 0 (inhibit time is 0 at start-up and is set when TPDO is sent for first time);
 - Elaboration time to elaborate one group of Synchronous TPDOs without send it (correct number of SYNCH message is not received);
2. **Tb** that represents:
 - Elaboration time to send one group of Asynchronous TPDO;
 - Elaboration time to send one group of Synchronous TPDOs;
3. **Tc** that represents:
 - Elaboration time of a single group of Asynchronous TPDOs when Inhibit time is not 0.

Each tables shows two timing values corresponding to the upper and lower frequency limits supported by CCIPC (16 and 10 MHz).

From the analysis of the results the following conclusions can be drawing:

- **Tb** decreases with the decrease of the number of TPDO per group, because the CCIPC has to send a smaller number of TPDO;
- **Ta** and **Tc** depend exclusively by number of TPDO supported by CCIPC;
- CCIPC is affected by elaboration delay when it's configured with 32 Nodes due to high number of TPDO to scan;

Moreover, since CCIPC executes a single service (TPDO, RPDO, SDO, SYNCH) at a time, the elaboration time for asynchronous timer-driven TPDO could be affected by further delay due to the waiting of the end of the current service. In this report, only asynchronous and synchronous TPDOs elaborations are activated.

13.1.1 CCIPC with 32 Node

The maximum number of TPDO supported is 128. The table reports the timings for 4 different TPDO configurations:

- Cfg-A: 128 asynchronous TPDOs with Inhibit and Event timer;
- Cfg-B: 64 asynchronous TPDOs with Inhibit and Event timer and 64 synchronous TPDOs;
- Cfg-C: 32 asynchronous TPDOs with Inhibit and Event timer and 96 synchronous TPDOs;
- Cfg-D: 32 asynchronous TPDOs with Inhibit and Event timer and 64 synchronous TPDOs and others disable.

For each configuration, the TPDOs have been divided in three set:

- SET#1: 8 groups of 16 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs);
- SET#2: 16 groups of 8 TPDOs Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs);
- SET#3: 32 groups of 4 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs).

Configuration		SET#1			SET#2			SET#3		
		Ta	Tb	Tc	Ta	Tb	Tc	Ta	Tb	Tc
Cfg-A		2.09	2.88	2.4	2.09	2.47	2.4	2.09	2.2	2.4
		3.34	4.6	3.84	3.34	3.95	3.84	3.34	3.52	3.84
Cfg-B	Asyn	1.54	2.33	1.69	1.54	1.92	1.69	1.54	1.70	1.69
		2.46	3.72	2.7	2.46	3.07	2.7	2.46	2.72	2.7
Cfg-B	Sync	1.07	1.98	-----	1.07	1.50	-----	1.07	1.29	-----
		1.71	3.16		1.71	2.4		1.71	2.06	
Cfg-C	Asyn	1.25	2.04	1.33	1.25	1.64	1.33	1.25	1.43	1.33
		2	3.26	2.13	2	2.624	2.13	2	2.28	2.13

		SET#1			SET#2			SET#3		
Configuration		Ta	Tb	Tc	Ta	Tb	Tc	Ta	Tb	Tc
Cfg-D	Sync	1.16 1.85	2.03 3.24	-----	1.16 1.85	1.63 2.6	-----	1.16 1.85	1.39 2.22	-----
	Asyn	1.16 1.85	1.96 3.13	1.24 1.98	1.16 1.85	1.54 2.46	1.24 1.98	1.16 1.85	1.33 2.13	1.24 1.98
	Sync	0.99 1.58	1.68 2.68	-----	0.99 1.58	1.46 2.33	-----	0.99 1.58	1.21 1.93	-----

Tab. 13-1: TPDO Elaboration for 32 Node count.

13.1.2 CCIPC with 16 Node

The maximum number of TPDO supported is 64. The table reports the timings for 4 different TPDO configurations:

- Cfg-A: 64 asynchronous TPDOs with Inhibit and Event timer;
- Cfg-B: 32 asynchronous TPDOs with Inhibit and Event timer and 32 synchronous TPDOs;
- Cfg-C: 16 asynchronous TPDOs with Inhibit and Event timer and 48 synchronous TPDOs;
- Cfg-D: 16 asynchronous TPDOs with Inhibit and Event timer and 32 synchronous TPDOs and others disable.

For each configuration the TPDOs have been divided in three set:

- SET#1: 4 groups of 16 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs);
- SET#2: 8 groups of 8 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs);
- SET#3: 16 groups of 4 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs).

		SET#1			SET#2			SET#3		
Configuration		Ta	Tb	Tc	Ta	Tb	Tc	Ta	Tb	Tc
Cfg-A		1.04 1.664	1.8 2.88	1.2 1.92	1.04 1.664	1.48 2.36	1.2 1.92	1.04 1.664	1.26 2.01	1.2 1.92
Cfg-B	Asyn	0.77 1.232	1.6 2.56	0.84 1.34	0.77 1.232	1.2 1.92	0.84 1.34	0.77 1.232	0.94 1.5	0.84 1.34
	Sync	0.5 0.8	1.5 2.4	-----	0.5 0.8	1.064 1.7	-----	0.5 0.8	0.70 1.12	-----
Cfg-C	Asyn	0.63 1	1.46 2.33	0.67 1.07	0.63 1	1.06 1.69	0.67 1.07	0.63 1	0.85 1.36	0.67 1.07
	Sync	0.58 0.92	1.34 2.14	-----	0.58 0.92	1.01 1.61	-----	0.58 0.92	0.81 1.29	-----
Cfg-D	Asyn	0.58 0.92	1.4 2.24	0.62 0.99	0.58 0.92	1.01 1.61	0.62 0.99	0.58 0.92	0.81 1.29	0.62 0.99
	Sync	0.49 0.78	1.25 2	-----	0.49 0.78	0.92 1.47	-----	0.49 0.78	0.72 1.15	-----

Tab. 13-2: TPDO Elaboration for 16 Node count.

13.1.3 CCIPC with 8 Node

The maximum number of TPDO supported is 32. The table reports the timings for 4 different TPDO configurations:

- Cfg-A: 32 asynchronous TPDOs with Inhibit and Event timer;

- Cfg-B: 16 asynchronous TPDOs with Inhibit and Event timer and 16 synchronous TPDOs;
- Cfg-C: 8 asynchronous TPDOs with Inhibit and Event timer and 24 synchronous TPDOs;
- Cfg-D: 8 asynchronous TPDOs with Inhibit and Event timer and 16 synchronous TPDOs and others disable.

For each configuration the TPDOs have been divided in three set:

- SET#1: 2 groups of 16 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs);
- SET#2: 4 groups of 8 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs);
- SET#3: 8 groups of 4 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs).

Configuration		SET#1			SET#2			SET#3		
		Ta	Tb	Tc	Ta	Tb	Tc	Ta	Tb	Tc
Cfg-A		0.52	1.53	0.6	0.52	0.98	0.6	0.52	0.8	0.6
		0.83	2.44	0.96	0.83	1.56	0.96	0.83	1.28	0.96
Cfg-B	Asyn	0.39	1.39	0.42	0.39	0.84	0.42	0.39	0.66	0.42
		0.62	2.22	0.67	0.62	1.34	0.67	0.62	1.05	0.67
Cfg-B	Sync	0.27	1.09	-----	0.27	0.68	-----	0.27	0.46	-----
		0.43	1.74	-----	0.43	1.08	-----	0.43	0.73	-----
Cfg-C	Asyn	-----	-----	-----	0.31	0.77	0.33	0.31	0.59	0.33
		-----	-----	-----	0.49	1.23	0.52	0.49	0.94	0.52
Cfg-C	Sync	-----	-----	-----	0.29	1.26	-----	0.29	0.55	-----
		-----	-----	-----	0.46	2.01	-----	0.46	0.88	-----
Cfg-D	Asyn	-----	-----	-----	0.29	0.75	0.31	0.29	0.57	0.31
		-----	-----	-----	0.46	1.2	0.51	0.46	0.91	0.51
Cfg-D	Sync	-----	-----	-----	0.25	0.81	-----	0.25	0.5	-----
		-----	-----	-----	0.4	1.29	-----	0.4	0.8	-----

Tab. 13-3: TPDO Elaboration for 8 Node count.

13.1.4 CCIPC with 4 Node

The maximum number of TPDO supported is 16. The table reports the timings for 4 different TPDO configurations:

- Cfg-A: 16 asynchronous TPDOs with Inhibit and Event timer;
- Cfg-B: 8 asynchronous TPDOs with Inhibit and Event timer and 8 synchronous TPDOs;
- Cfg-C: 4 asynchronous TPDOs with Inhibit and Event timer and 12 synchronous TPDOs;
- Cfg-D: 4 asynchronous TPDOs with Inhibit and Event timer and 8 synchronous TPDOs and others disable.

For each configuration the TPDOs have been divided in three set:

- SET#1: 1 groups of 16 TPDO characterized by specific values of event and inhibit timers;
- SET#2: 2 groups of 8 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs);
- SET#3: 4 groups of 4 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs).

Configuration		SET#1			SET#2			SET#3		
		Ta	Tb	Tc	Ta	Tb	Tc	Ta	Tb	Tc
Cfg-A		0.26	1.21	0.3	0.26	0.72	0.3	0.26	0.46	0.3
		0.41	1.93	0.48	0.41	1.15	0.48	0.41	0.73	0.48

Configuration		SET#1			SET#2			SET#3		
		Ta	Tb	Tc	Ta	Tb	Tc	Ta	Tb	Tc
Cfg-B	Asyn	-----	-----	-----	0.19 0.3	0.65 1.04	0.21 0.33	0.19 0.3	0.39 0.62	0.21 0.33
	Sync	-----	-----	-----	0.14 0.22	0.64 1.02	-----	0.14 0.22	0.38 0.6	-----
Cfg-C	Asyn	-----	-----	-----	-----	-----	-----	0.16 0.25	0.36 0.57	0.17 0.27
	Sync	-----	-----	-----	-----	-----	-----	0.15 0.24	0.41 0.65	-----
Cfg-D	Asyn	-----	-----	-----	-----	-----	-----	0.14 0.22	0.34 0.54	0.15 0.24
	Sync	-----	-----	-----	-----	-----	-----	0.13 0.2	0.38 0.6	-----

Tab. 13-4: TPDO Elaboration for 32 Node count.

13.1.5 CCIPC with 2 Node

The maximum number of TPDO supported is 8. The table reports the timings for 2 different TPDO configurations:

- Cfg-A: 8 asynchronous TPDOs with Inhibit and Event timer;
- Cfg-B: 4 asynchronous TPDOs with Inhibit and Event timer and 4 synchronous TPDOs;

For each configuration the TPDOs have been divided in two set:

- SET#1: 1 groups of 8 TPDO characterized by specific values of event and inhibit timers;
- SET#2: 2 groups of 4 TPDOs. Each group is characterized by specific values for event and inhibit timers (asynchronous TPDOs), or transfer type (synchronous TPDOs).

Configuration		SET#1			SET#2		
		Ta	Tb	Tc	Ta	Tb	Tc
Cfg-A		0.13 0.2	0.49 0.78	0.15 0.24	0.13 0.2	0.32 0.51	0.15 0.24
Cfg-B	Asyn	-----	-----	-----	0.096 0.15	0.29 0.46	0.11 0.17
	Sync	-----	-----	-----	0.072 1.15	0.24 0.38	-----

Tab. 13-5: TPDO Elaboration for 2 Node count.

13.1.6 CCIPC with 1 Node

The maximum number of TPDO supported is 4. The table reports the timings for 2 different TPDO configurations:

- Cfg-A: 4 asynchronous TPDOs with Inhibit and Event timer;
- Cfg-B: 2 asynchronous TPDOs with Inhibit and Event timer and 2 synchronous TPDOs;

Configuration		SET#1		
		Ta	Tb	Tc
Cfg-A		0.065 0.1	0.25 0.4	0.075 0.12
Cfg-B	Asyn	0.048 0.076	0.14 0.22	0.053 0.084

Configuration		SET#1		
		Ta	Tb	Tc
	Sync	0.039	0.01	-----
		0.062	0.02	

Tab. 13-6: TPDO Elaboration for 1 Node count.

14 CCIPC PORTABILITY

14.1 CAN Bus Controller

The CCIPC has been designed and in conjunction with the ESA HurriCANE core (v. 5.2.4). To insert this CAN Bus controller in the CCIPC database, all its source files have to be copied in the **CAN_CORE** directory, contained in **SRC** directory.

The CAN Bus Controller specific interface details are managed in the CCIPC_interface Block. This VHDL module has to be modified if a different controller than HurriCANE is used.

14.2 FPGA Technology

The CCIPC DataBase is equipped to support the following FPGA families:

- Microsemi (RT)AX family;
- Microsemi ProAsic family;
- Xilinx Virtex IV family
- Generic

The CCIPC Database is furnished with the memory model for the Generic technology. The selection of the target technology has to be specified in the CCIPC Configuration file (*CCIPCconf.vhd*). This file already supports the Microsemi and Xilinx options.

To insert other technology memory models follows these steps:

- 1) In SRC/LIBRARIES folder create a "MY_TECH" directory;
- 2) Copy "my_technology_memory.vhd" file in "MY_TECH" directory. **Note that the target memory model supported by CCIPC is 512x8 bit.**
- 3) Modify "CCIPCconf.vhd" file inserting a new technology constant. For example:
constant my_technology: std_logic_vector(3 downto 0) := "1001";
- 4) Add the memory model instantiation to "tech_mem.vhd" file considering that CCIPC target memory (ram512x8) supports the following signals:

Signals	Direction	Function
Data[7:0]	I	Write bus
Q[7:0]	O	Read Bus
WAddress[8:0]	I	Write Address
RAddress[8:0]	I	Read Address
WE	I	Write Enable (active high)
RE	I	Read Enable (active high)
Clock	I	Clock

Tab. 14-1: CCIPC 512x8 target memory peripheral.

14.3 Rad-Hard Technique

The CCIPC has been developed considering Microsemi RTAX FPGAs as target technology.

Microsemi RTAX technology uses SEU-Hardened Registers avoiding the need to add Triple-Module Redundancy (TMR).

CCIPC uses FPGA on-chip RAM and it includes an Error Detection And Correction (EDAC) module in order to:

- detect and correct single-bit error;
- detect double-bit error

The equation below shows the Hamming code used during generation of EDAC bits (EB):

$$EB(0) := D(0) \wedge D(1) \wedge D(2) \wedge D(4) \wedge D(5) \wedge D(7) \wedge D(10) \wedge D(11) \wedge D(13) \wedge D(16) \wedge D(20) \wedge D(21) \wedge D(23) \wedge D(26) \wedge D(30);$$

$$EB(1) := D(0) \wedge D(1) \wedge D(3) \wedge D(4) \wedge D(6) \wedge D(8) \wedge D(10) \wedge D(12) \wedge D(14) \wedge D(17) \wedge D(20) \wedge D(22) \wedge D(24) \wedge D(27) \wedge D(31);$$

$$EB(2) := D(0) \wedge D(2) \wedge D(3) \wedge D(5) \wedge D(6) \wedge D(9) \wedge D(11) \wedge D(12) \wedge D(15) \wedge D(18) \wedge D(21) \wedge D(22) \wedge D(25) \wedge D(28);$$

$$EB(3) := D(1) \wedge D(2) \wedge D(3) \wedge D(7) \wedge D(8) \wedge D(9) \wedge D(13) \wedge D(14) \wedge D(15) \wedge D(19) \wedge D(23) \wedge D(24) \wedge D(25) \wedge D(29);$$

$$EB(4) := D(4) \wedge D(5) \wedge D(6) \wedge D(7) \wedge D(8) \wedge D(9) \wedge D(16) \wedge D(17) \wedge D(18) \wedge D(19) \wedge D(26) \wedge D(27) \wedge D(28) \wedge D(29);$$

$$EB(5) := D(10) \wedge D(11) \wedge D(12) \wedge D(13) \wedge D(14) \wedge D(15) \wedge D(16) \wedge D(17) \wedge D(18) \wedge D(19) \wedge D(30) \wedge D(31);$$

$$EB(6) := D(20) \wedge D(21) \wedge D(22) \wedge D(23) \wedge D(24) \wedge D(25) \wedge D(26) \wedge D(27) \wedge D(28) \wedge D(29) \wedge D(30) \wedge D(31);$$

A dedicated register is used to control EDAC functionality.

Two dedicated flags are used to enable or disable EDAC protection on Memory or Queue modules:

- EDAC Enable on CCIPC memory device allows enabling or disabling EDAC protection on External and Internal memory devices;
- EDAC Enable on CCIPC queue device allows enabling or disabling EDAC protection on CCIPC internal message queues.

Bit Number	Description	Reset value
7-0	Single error event counter	0
8	Single error flag	0
9	Double error flag	0
10	EDAC Enable on CCIPC memory device	0
11	EDAC Enable on CCIPC queue device	0
15-12	Reserved	0
16	Single error mask	0
17	Double error mask	0

Tab. 14-2: EDAC error register.

The Single error flag is set when single error counter exceeds the threshold value of 255 errors. User can set the Single error counter value in order to control single error flag behaviour.

A write operation on the register forces both single and double error flag to reset value.

The EDAC function is disabled at start-up. User is in charge of enabling it after CCIPC initialization.

Since Double error detection represents a critical condition for the core because the management of wrong values can compromise the CCIPC behaviour, it is strictly recommended that user resets the CCIPC core if this error occurs.

END OF DOCUMENT