Journal of Information Hiding and Multimedia Signal Processing ©2010 ISSN 2073-4212
Ubiquitous International Volume **1**, Number **4**, October **2010**

Approach for VHDL and FPGA Implementation of Communication Controller of FlexRay Controller

Milind Khanapurkar[1], Jayant Y. Hande[2] and Dr. Preeti Bajaj[1]

[1]Assistant Professor & Research Associate, IEEE Member, ECE Dept.
[2]Research Student, ECE Dept.
[3]Professor and Senior Member IEEE,
G. H. Raisoni College of Engineering, Nagpur, India
[1]m_khanapurkar@rediffmail.com
[2]hande_jayant@rediffmail.com

Received April 2010; revised July 2010

Abstract. *The FlexRay communication system is an emerging standard for advanced automotive control applications, such as drive-by-wire. The detailed micro-architectural level of the FlexRay specification facilitates implementations where the internal operation closely reflects the protocol specification definitions. Corresponding functional coverage models in the verification environment can also be defined relative to the FlexRay specification which enables consistent understanding, easier maintenance and better reuse. This paper explores the general issues of functional coverage pertaining to the FlexRay specifications. VHDL simulation & FPGA synthesis results of communication controller of flexray controller are tested on xilinix and lenardo Spectrum tool are presented.*
Keywords: ECU V Electronic Control Units, FPGA: Field Programmable Gate Array, LIN: Local Interconnect Network, CAN: Controller Area Network, TDMA: Time Division Multiple Access,CC: Communication Controller.

1. **Introduction.** Electronics in vehicles is increasing at a phenomenal rate every year, every now and then new born applications are arriving and existing applications are getting modified with sophisticated recent technologies such as soft computing tools, VHDL/ VLSI tools, advanced tools for designing real time embedded systems. Intra-vehicular communication describes as exchange of data within the ECUs of the vehicle which are involved in vehicular applications. FlexRay is a new communication protocol designed to provide large bunches of data to be exchanged in real-time and with high dependability between electronic control units (ECU), sensors and actuators. It features data-rates up to 10 Mb/s and is accounting for time and event triggered transmissions. FlexRay serves as the next step beyond CAN and LIN, enabling the reliable management of many more safety and comfort features. In some node designs, the host and controller part can be combined into one entity if the microcontroller has an integrated FlexRay controller. This combination allows the designer to implement the FlexRay protocol using a single chip without the need for companion devices. This paper discusses data communication controller, its states and input and output signals. First two sections discuss a brief overview of the FlexRay ECU along with flex-ray data frame and block diagram of a flex-ray communication node. Followed by VHDL simulation and FPGA synthesis results of communication controller of FlexRay.

1.1. **Architecture of FlexRay ECU.** Each flexRay node consists of a host, communication controller, power supply unit, two bus guardians, and two bus drivers. Each communication channel has one bus driver to connect the node to the channel. The Host is an embedded microprocessor running the software which controls the communication process. Communication Controller provides the operation (via the Bus Driver) and is responsible for synchronization. Bus Guardian provides mainly the error detection, generating interrupts and blocking the line when critical problems occur. Flex-Ray node is equipped with two physical Channels V A and B.

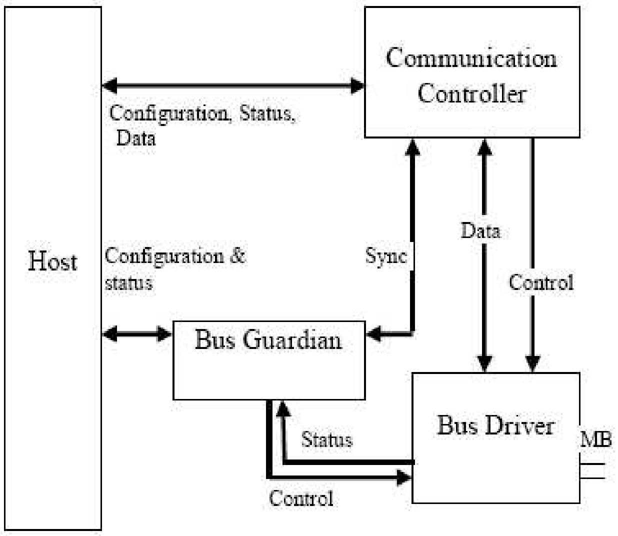


FIGURE 1. Architecture of Flex-ray Node.

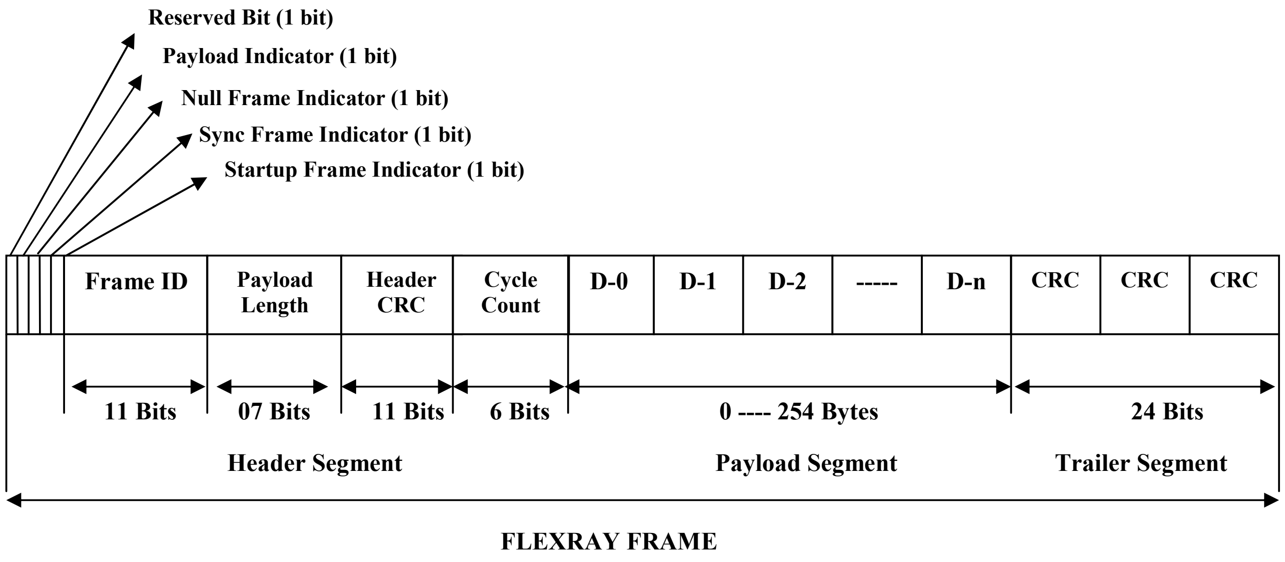


FIGURE 2. Flex ray Frame Format.

1.2. **Flex-Ray Frame.** Flex-Ray protocol is based on frames, containing data organized in bytes, but transmitted serially. Figure 2 shows the flex-ray frame, consists of 3 segmentsVheader, Payload and Trailer.

- The Header begins with 5 indicators Vthe single bits defining basic features of the frame.

- Payload section contains the main data; its length may be variable between 0 and 254 bytes.
- The Trailer section contains 24 bits of CRC, calculated for the Header and Payload section together.

Each cycle is a complex structure, containing static segment, dynamic segment, symbol window and idle time. Static segment the first part of cycle contains series of static slots each of these slots is allotted to transmit a single frame. Another part of the cycle is the dynamic segment consists of mini slots. This part of cycle may be used for the frames transmission again but the amount of time allotted a current frame may vary, depending on its length.

2. **Communication Controller.** Communication controller (CC) is responsible for implementing the protocol aspects of the Flex Ray communication system. It performs all communications tasks such as reception and transmission of messages in a TTP cluster without interaction of the host CPU. Communication controller (CC) provides status information to the host and delivers payload data received from communication frames.

2.1. **Flex ray Protocol Operation and Control.** The flexRay communication protocol uses TDMA (time-division multiple access) to the media for providing communication between nodes which must be synchronous for participation in data exchange process. In the paper we have considered two processes viz wakeup and startup.

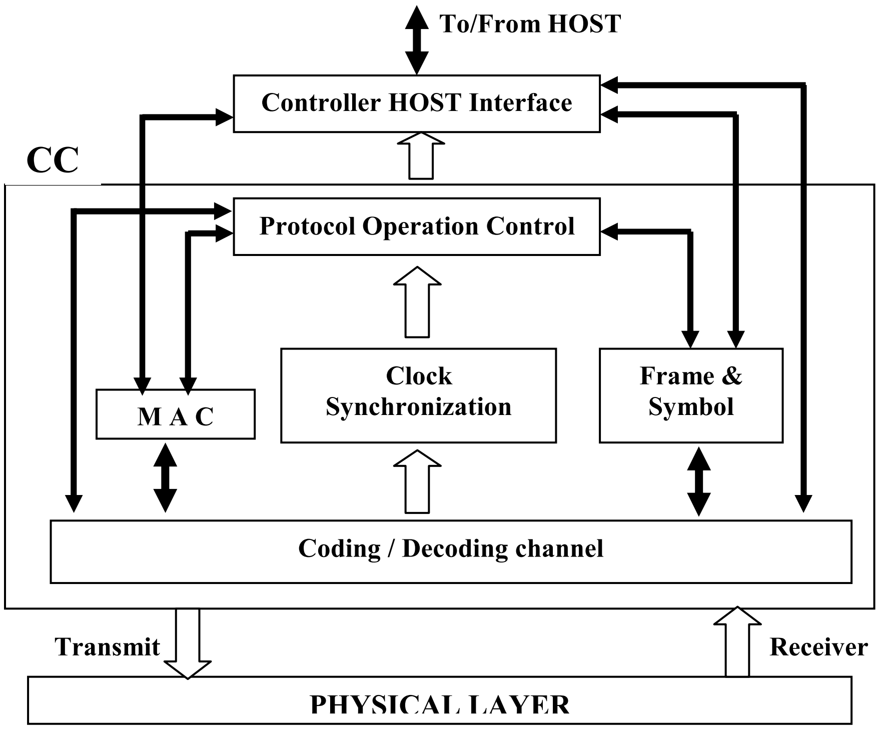


Figure 3. Block Diagram of Communication Controller.

The node shall enter the default config state when the node is supplied with power. In this state static configuration of the communication controller is performed. Host configures number of channels, media specification, etc. The state is left as soon as the bus driver receives a wakeup event which can be an external wakeup event or receiving the wakeup pattern on its channel.

In the config state, the host configures the communication controller. Then the host checks wakeup event, based on this information decides what state will be next. If the bus driver has received an external wakeup event, the host must command to wakeup other nodes on an attached channel. Otherwise, if the bus driver has received a wakeup pattern from the channel, the host may command to wakeup the second channel, or to proceed to startup state.

In the wakeup state, the node tries to wakeup the defined channel. If errors occur during the wakeup state, the communication controller informs the host and the node switches to the config state. Otherwise the node switches to startup state.

In the startup state, the node checks for ongoing communication on the media. If the bus driver detects communication signals on the attached channel, then the node integrates to the cluster communication, otherwise the node starts the startup procedure. After startup, node proceeds to the normal active state.

2.2. **Communication Controller States.** The states of communication controller and respective commands are as in state table (figure 4) and the state flow is as described in state diagram shown in figure 5. Its frame structure by frame table (figure 7) and protocol operation for particular states using operation table (figure 8).

STATE_TYPE	COMMAND [4-0]
DEFAULT_CONFIG	00000
CONFIG	00001
READY	00010
HALT	00011
WAKEUP_LISTEN	00100
WAKEUP_SEND	00101
WAKEUP_DETECT	00110
COLDSTART_LISTEN	00111
INTEGRATION_LISTEN	01000
INITIALIZE_SCHEDULE	01001
NORMAL_ACTIVE	01010
NORMAL_PASSIVE	01011
COLDSTART_COLLISION_RESOLUTION	01100
INTEGRATION_COLDSTART_CHECK	01101
INTEGRATION_CONSISTENCY_CHECK-	01110
COLDSTART_CONSISTANCY_CHECK	01111
COLDSTART_JOIN	10000
COLDSTART_GAP	10001

Figure 4. State Table.

- DEFAULT_CONFIG state: The CC enters this state when leaving hard reset or when exiting from HALT state. To leave DEFAULT_CONFIG state, the Host has to write Command [4:0] = “00001”’ then transits to CONFIG state.
- CONFIG state: The CC enters this state when exiting from DEFAULT_CONFIG state or when exiting from READY state. After unlocking CONFIG state and writing command [4:0] = “00010”’ the CC enters READY state. From this state the CC can transit to WAKEUP state and perform a cluster wakeup or to STARTUP state to perform a cold start or to integrate into a running cluster. The CC enters this state
- Ready State: When exiting from CONFIG, WAKEUP, STARTUP, NORMAL_ACTIVE, or NORMAL_PASSIVE state by writing command [4:0] = “00010”’ (READY command). The CC exits from this state To CONFIG state by writing command [4:0] = “00001” (CONFIG command) and To WAKEUP state by writing command [4:0] =

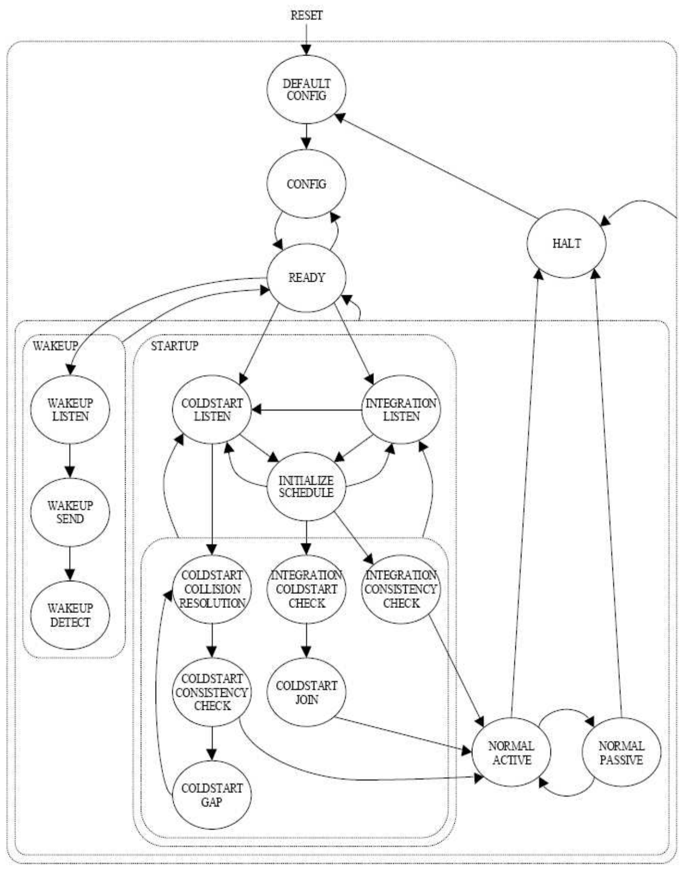
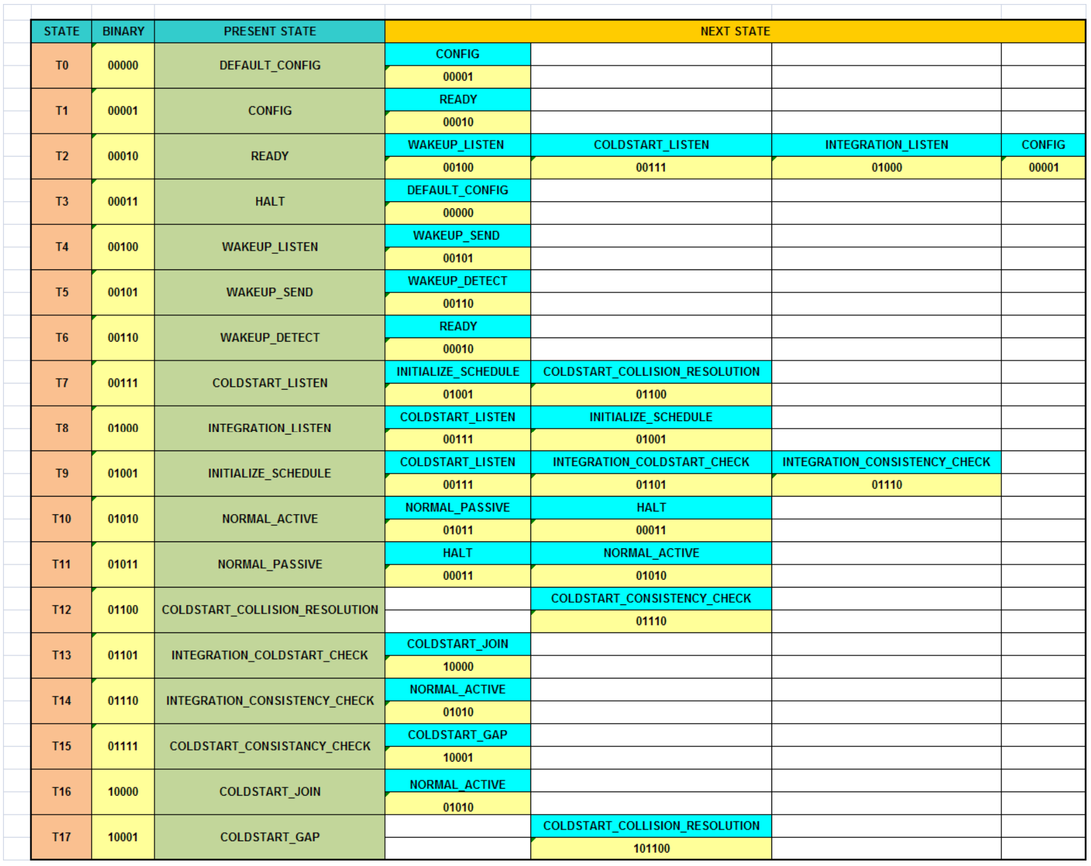


FIGURE 5. Overview of Protocol Operation Control.

"00100" (WAKEUP command) and To STARTUP state by writing command [4:0] = "00111" (STARTUP command).

- WAKEUP State: CC enters this state when exiting from READY state by writing command [4:0] = "00100" (WAKEUP command). The CC exits from this state to READY state after complete non-aborted transmission of wakeup pattern or after WUP reception or after detecting a WUP collision or after reception of a frame header or by writing command [4:0] = "00010" (READY command)
- The WAKEUP_LISTEN state is controlled by the wakeup timer and the wakeup noise timer. The two timers are controlled by the parameters listen timeout and listen timeout noise. Listen timeout enables a fast cluster wakeup in case of a noise free environment, while listen timeout noise enables wakeup under more difficult conditions regarding noise interference. In WAKEUP_SEND state the CC transmits the wakeup pattern on the configured channel and checks for collisions. After return from wakeup the Host has to bring the CC into STARTUP state by CHI command RUN. In WAKEUP_DETECT state the CC attempts to identify the reason for the wakeup collision detected in WAKEUP_SEND state.
- STARTUP State: Any node entering STARTUP state that has cold start capability should assure that both channels attached have been awakened before initiating cold start.

- NORMAL_ACTIVE state: Cold start path initiating the schedule synchronization or Cold start path joining other cold start nodes or Integration path integrating into an existing communication schedule (all other nodes). A cold start attempt begins with the transmission of a collision avoidance symbol (CAS). Only a cold start node



STATE	BINARY	PRESENT STATE	NEXT STATE			
T0	00000	DEFAULT_CONFIG	CONFIG			
			00001			
T1	00001	CONFIG	READY			
			00010			
T2	00010	READY	WAKEUP_LISTEN	COLDSTART_LISTEN	INTEGRATION_LISTEN	CONFIG
			00100	00111	01000	00001
T3	00011	HALT	DEFAULT_CONFIG			
			00000			
T4	00100	WAKEUP_LISTEN	WAKEUP_SEND			
			00101			
T5	00101	WAKEUP_SEND	WAKEUP_DETECT			
			00110			
T6	00110	WAKEUP_DETECT	READY			
			00010			
T7	00111	COLDSTART_LISTEN	INITIALIZE_SCHEDULE	COLDSTART_COLLISION_RESOLUTION		
			01001	01100		
T8	01000	INTEGRATION_LISTEN	COLDSTART_LISTEN	INITIALIZE_SCHEDULE		
			00111	01001		
T9	01001	INITIALIZE_SCHEDULE	COLDSTART_LISTEN	INTEGRATION_COLDSTART_CHECK	INTEGRATION_CONSISTENCY_CHECK	
			00111	01101	01110	
T10	01010	NORMAL_ACTIVE	NORMAL_PASSIVE	HALT		
			01011	00011		
T11	01011	NORMAL_PASSIVE	HALT	NORMAL_ACTIVE		
			00011	01010		
T12	01100	COLDSTART_COLLISION_RESOLUTION		COLDSTART_CONSISTENCY_CHECK		
				01110		
T13	01101	INTEGRATION_COLDSTART_CHECK	COLDSTART_JOIN			
			10000			
T14	01110	INTEGRATION_CONSISTENCY_CHECK	NORMAL_ACTIVE			
			01010			
T15	01111	COLDSTART_CONSISTANCY_CHECK	COLDSTART_GAP			
			10001			
T16	10000	COLDSTART_JOIN	NORMAL_ACTIVE			
			01010			
T17	10001	COLDSTART_GAP		COLDSTART_COLLISION_RESOLUTION		
				101100		

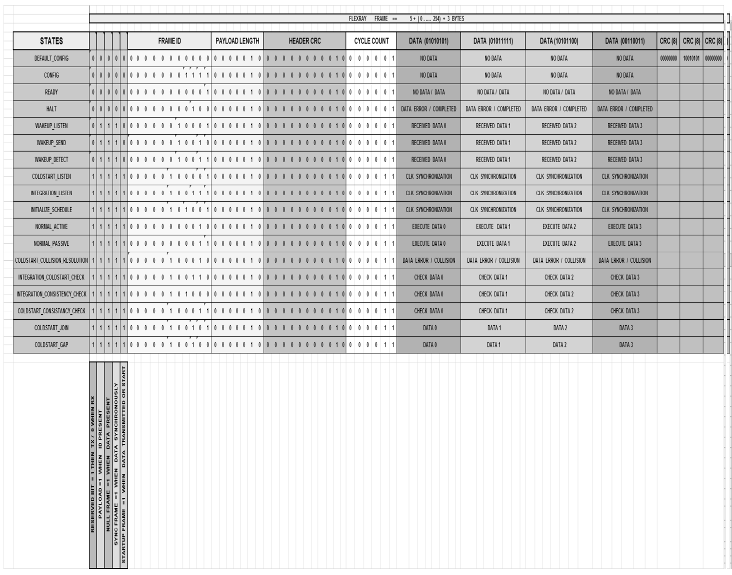
Figure 6. Overview of State Operation.

that had transmitted the CAS transmits frames in the first four cycles after the CAS, it is then joined firstly by the other cold start nodes and afterwards by all other nodes. In the frame header of the startup frame the startup frame indicator bit is set. In clusters consisting of three or more nodes, at least three nodes shall be configured to be cold start nodes. In clusters consisting of two nodes, both nodes must be cold start nodes. Each startup frame must also be a sync frame; therefore each cold start node will also be a sync node. A non-cold start node requires at least two startup frames from distinct nodes for integration. It may start integration before the cold start nodes have finished their startup. It will not finish its startup until at least two cold start nodes have finished their startup.

3. **VHDL Simulation and Synthesis Results.** The communication controller is designed by FSM methodology. The states coding and command sequence as per state table, frame structure and protocol operation using VHDL code on XILINX tool (simulation and synthesis) and Leonardo Spectrum (Synthesis) tool are shown below.

1. DefaultConfig to Wakeup State: In the default config to wakeup stage the POC awaits the explicit command from the host to enable configuration. The global clock pulse starts from initially and reset changes from HIGH to LOW. In default config stage command (4:0) value is “00000,” values of frame header, payload length, header CRC, data (3 to 0) all remains null. From config state to wakeup state data values changes from null values to state related values.

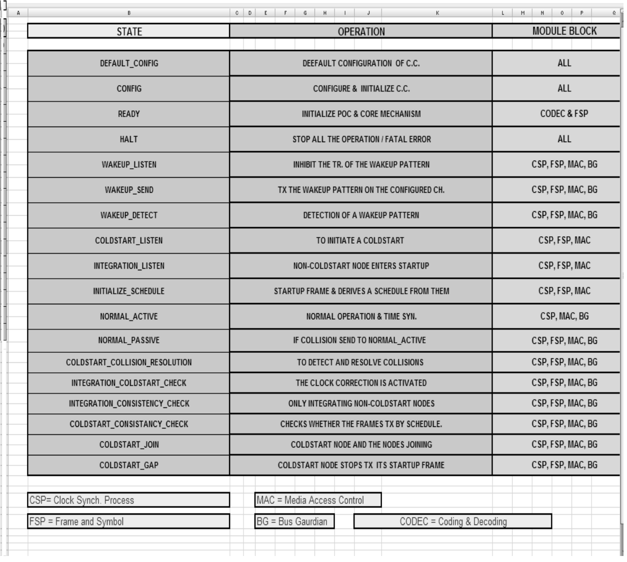
2. Startup State to Normal Active State: From this stage actual working of CC is start, in wakeup state CC listen the channel and gives proper indication to Host. Host gives ready command to POC, then after startup command to POC, these indicate the


FLEXRAY FRAME == 5 + (0 ..... 254) + 3 BYTES

STATES						FRAME ID	PAYLOAD LENGTH	HEADER CRC	CYCLE COUNT	DATA (01010101)	DATA (01011111)	DATA (10101100)	DATA (00110011)	CRC (8)	CRC (8)	CRC (8)
DEFAULT_CONFIG	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1	NO DATA	NO DATA	NO DATA	NO DATA	00000000	10010101	00000000
CONFIG	0	0	0	0	0	0 0 0 0 0 0 0 1 1 1 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1	NO DATA	NO DATA	NO DATA	NO DATA			
READY	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1	NO DATA / DATA	NO DATA / DATA	NO DATA / DATA	NO DATA / DATA			
HALT	0	0	0	0	0	0 0 0 0 0 0 0 0 1 0 0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1	DATA ERROR / COMPLETED	DATA ERROR / COMPLETED	DATA ERROR / COMPLETED	DATA ERROR / COMPLETED			
WAKEUP_LISTEN	0	1	1	1	0	0 0 0 0 0 0 1 0 0 0 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1	RECEIVED DATA 0	RECEIVED DATA 1	RECEIVED DATA 2	RECEIVED DATA 3			
WAKEUP_SEND	0	1	1	1	0	0 0 0 0 0 0 1 0 0 1 0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1	RECEIVED DATA 0	RECEIVED DATA 1	RECEIVED DATA 2	RECEIVED DATA 3			
WAKEUP_DETECT	0	1	1	1	0	0 0 0 0 0 0 1 0 0 1 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1	RECEIVED DATA 0	RECEIVED DATA 1	RECEIVED DATA 2	RECEIVED DATA 3			
COLDSTART_LISTEN	1	1	1	1	1	0 0 0 0 0 1 0 0 0 0 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION			
INTEGRATION_LISTEN	1	1	1	1	1	0 0 0 0 0 1 0 0 1 1 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION			
INITIALIZE_SCHEDULE	1	1	1	1	1	0 0 0 0 0 1 0 1 0 0 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION	CLK SYNCHRONIZATION			
NORMAL_ACTIVE	1	1	1	1	1	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	EXECUTE DATA 0	EXECUTE DATA 1	EXECUTE DATA 2	EXECUTE DATA 3			
NORMAL_PASSIVE	1	1	1	1	1	0 0 0 0 0 0 0 0 0 1 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	EXECUTE DATA 0	EXECUTE DATA 1	EXECUTE DATA 2	EXECUTE DATA 3			
COLDSTART_COLLISION_RESOLUTION	1	1	1	1	1	0 0 0 0 0 1 0 0 0 1 0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	DATA ERROR / COLLISION	DATA ERROR / COLLISION	DATA ERROR / COLLISION	DATA ERROR / COLLISION			
INTEGRATION_COLDSTART_CHECK	1	1	1	1	1	0 0 0 0 0 1 0 0 1 1 0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	CHECK DATA 0	CHECK DATA 1	CHECK DATA 2	CHECK DATA 3			
INTEGRATION_CONSISTENCY_CHECK	1	1	1	1	1	0 0 0 0 0 1 0 1 0 0 0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	CHECK DATA 0	CHECK DATA 1	CHECK DATA 2	CHECK DATA 3			
COLDSTART_CONSISTANCY_CHECK	1	1	1	1	1	0 0 0 0 0 1 0 0 0 1 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	CHECK DATA 0	CHECK DATA 1	CHECK DATA 2	CHECK DATA 3			
COLDSTART_JOIN	1	1	1	1	1	0 0 0 0 0 1 0 0 1 0 1	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	DATA 0	DATA 1	DATA 2	DATA 3			
COLDSTART_GAP	1	1	1	1	1	0 0 0 0 0 1 0 0 1 0 0	0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 1	DATA 0	DATA 1	DATA 2	DATA 3			

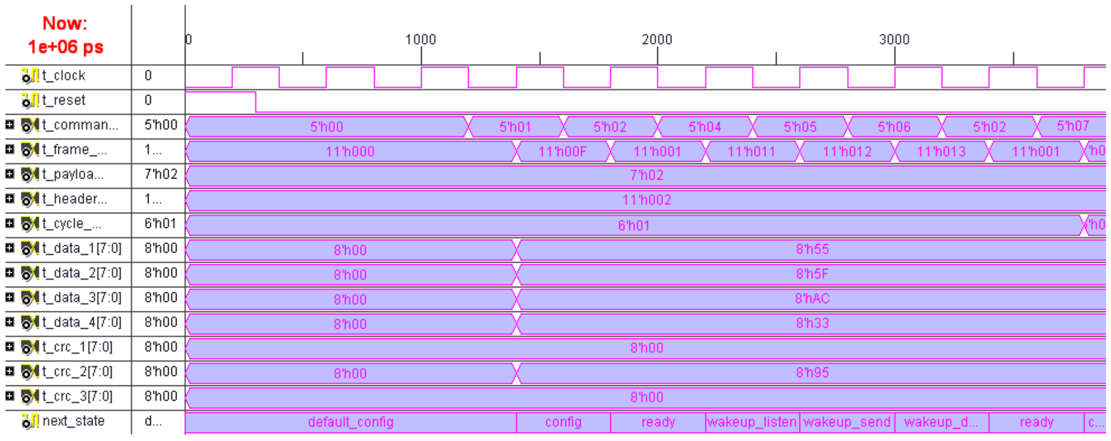
RESERVED BIT = 1 THEN TX / 0 WHEN RX
PAYLOAD =1 WHEN ID PRESENT
NULL FRAME =1 WHEN DATA PRESENT
SYNC FRAME =1 WHEN DATA SYNCHRONOUSLY
STARTUP FRAME =1 WHEN DATA TRANSMITTED OR START

Figure 7. Overview of Protocol Frame.


STATE	OPERATION	MODULE BLOCK
DEFAULT_CONFIG	DEEFAULT CONFIGURATION OF C.C.	ALL
CONFIG	CONFIGURE & INITIALIZE C.C.	ALL
READY	INITIALIZE POC & CORE MECHANISM	CODEC & FSP
HALT	STOP ALL THE OPERATION / FATAL ERROR	ALL
WAKEUP_LISTEN	INHIBIT THE TR. OF THE WAKEUP PATTERN	CSP, FSP, MAC, BG
WAKEUP_SEND	TX THE WAKEUP PATTERN ON THE CONFIGURED CH.	CSP, FSP, MAC, BG
WAKEUP_DETECT	DETECTION OF A WAKEUP PATTERN	CSP, FSP, MAC, BG
COLDSTART_LISTEN	TO INITIATE A COLDSTART	CSP, FSP, MAC
INTEGRATION_LISTEN	NON-COLDSTART NODE ENTERS STARTUP	CSP, FSP, MAC
INITIALIZE_SCHEDULE	STARTUP FRAME & DERIVES A SCHEDULE FROM THEM	CSP, FSP, MAC
NORMAL_ACTIVE	NORMAL OPERATION & TIME SYN.	CSP, MAC, BG
NORMAL_PASSIVE	IF COLLISION SEND TO NORMAL_ACTIVE	CSP, FSP, MAC, BG
COLDSTART_COLLISION_RESOLUTION	TO DETECT AND RESOLVE COLLISIONS	CSP, FSP, MAC, BG
INTEGRATION_COLDSTART_CHECK	THE CLOCK CORRECTION IS ACTIVATED	CSP, FSP, MAC, BG
INTEGRATION_CONSISTENCY_CHECK	ONLY INTEGRATING NON-COLDSTART NODES	CSP, FSP, MAC, BG
COLDSTART_CONSISTANCY_CHECK	CHECKS WHETHER THE FRAMES TX BY SCHEDULE.	CSP, FSP, MAC, BG
COLDSTART_JOIN	COLDSTART NODE AND THE NODES JOINING	CSP, FSP, MAC, BG
COLDSTART_GAP	COLDSTART NODE STOPS TX ITS STARTUP FRAME	CSP, FSP, MAC, BG

CSP= Clock Synch. Process
MAC = Media Access Control
FSP = Frame and Symbol
BG = Bus Gaurdian
CODEC = Coding & Decoding

Figure 8. Overview of Protocol Operation.



Now:
1e+06 ps
0
1000
2000
3000
t_clock 0
t_reset 0
t_comman... 5'h00 | 5'h00 | 5'h01 | 5'h02 | 5'h04 | 5'h05 | 5'h06 | 5'h02 | 5'h07
t_frame_... 1... | 11'h000 | 11'h00F | 11'h001 | 11'h011 | 11'h012 | 11'h013 | 11'h001 | 11'h0
t_payloa... 7'h02 | 7'h02
t_header... 1... | 11'h002
t_cycle_... 6'h01 | 6'h01 | 6'h0
t_data_1[7:0] 8'h00 | 8'h00 | 8'h55
t_data_2[7:0] 8'h00 | 8'h00 | 8'h5F
t_data_3[7:0] 8'h00 | 8'h00 | 8'hAC
t_data_4[7:0] 8'h00 | 8'h00 | 8'h33
t_crc_1[7:0] 8'h00 | 8'h00
t_crc_2[7:0] 8'h00 | 8'h00 | 8'h95
t_crc_3[7:0] 8'h00 | 8'h00
next_state d... | default_config | config | ready | wakeup_listen | wakeup_send | wakeup_d... | ready | c...

Figure 9. Default_Config To Wakeup State.

CC enters in cold start node. The cold start nodes can initiate the cluster start up. From startup stage actual communication done between channel and Host. Startup has stages: Cold start Listen State, Integration Listen State, Initialize Schedule State, Cold start Collision Resolution State, Integration Cold start Check State, Integration Consistency Check State, and Cold start Consistency Check.

3. Normal Active to Halt State: Normal Active state the POC performs a sequence of tasks at the end of each communication cycle for the purpose of determining whether the POC should change the modeling of the core mechanisms before the beginning of the next communication cycle. The POC performs a sequence of tasks at the end of each communication cycle if collision is done then POC enter in Normal Passive state. In between CC get any fatal error or hang the process due to any problem POC enter in Halt state, to stop all operation and reset the controller which provides security.

4. Frame Structure: The output of frame depends on input command (4-0), clock and reset signal. The FlexRay frame is divided into three segments. The segments are Header, Payload, and Trailer. All signals depends on the input data which is present in frame. In header section 5 bit indicator, frame ID (11 bits), payload length (7 bits),

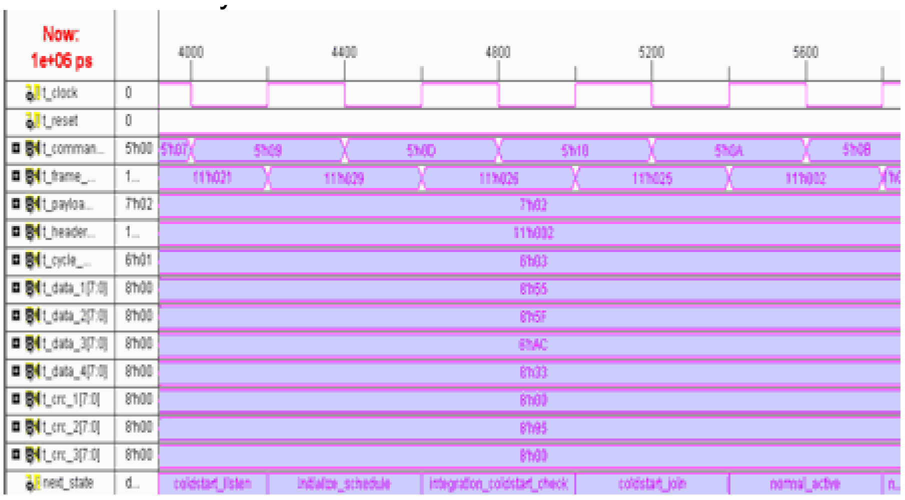


FIGURE 10. Startup State to Normal Active State.

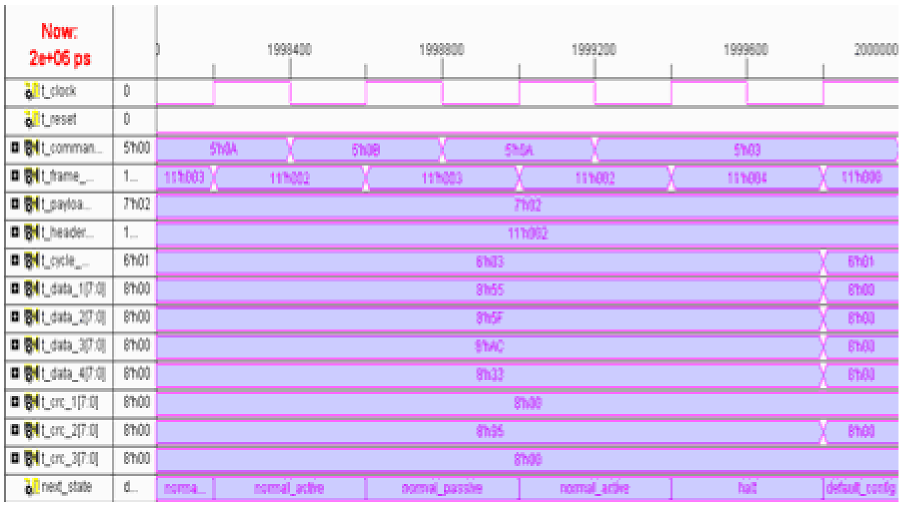


FIGURE 11. Normal Active to Halt State.

header CRC (11 bits) where as in payload data (0-254 bytes) 7 bits per data values and in trailer last tail part is present.

Commands apply by Host for respective state ("00000" to "10001"). Frame identity and frame has present data or null all these information present in Frame_ID.

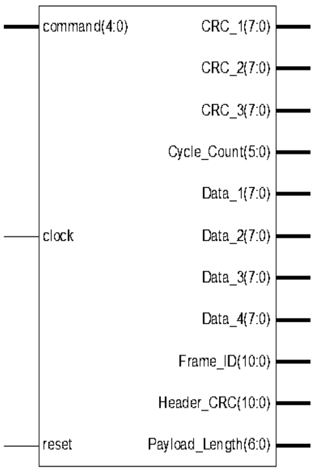
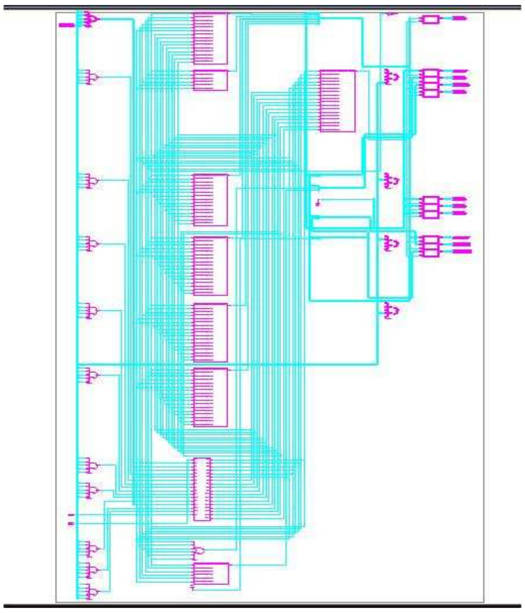


FIGURE 12. RTL View of Frame Structure.

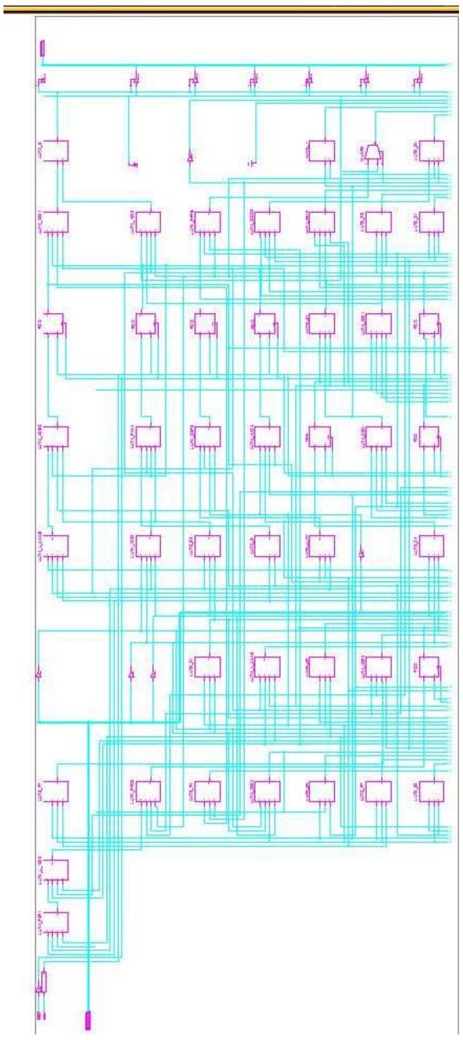
Figure 13a and figure 13b shows the xilinx RTL synthesis and schematic synthesis structure of communication controller of flexRay controller.

Figure 14a and figure 14b shows the Leonardo Spectrum RTL Synthesis along with synthesis report.

```
THE LEONARDO SPECTRUM SYNTHESIS REPORT:
-- LOADING ARCHITECTURE RTL OF CONTROLLER_T INTO LIBRARY WORK
-- COMPILING ROOT ENTITY CONTROLLER_T(RTL)
"C:/CONTROLLER/CONTROLLER.VHD",LINE 573: W ARNING, OTHERS CLAUSE IS NEVER SELECTED.
USING WIRE TABLE: XIS230-6_AVG
-- START TIMING OPTIMIZATION FOR DESIGN .WORK.CONTROLLER_T.RTL
NO CRITICAL PATHS TO OPTIMIZE AT THIS LEVEL
***********************************
CELL: CONTROLLER_T VIEW : RTL LIBRARY: WORK
**********************************
 NUMBER OF PORTS :                   98
 NUMBER OF NETS :                   218
 NUMBER OF INSTANCES :                 211
 NUMBER OF REFERENCES TO THIS VIEW : 0
 TOTAL ACCUMULATED AREA :
 NUMBER OF BUFGP :                      1
 NUMBER OF DFFS OR LATCHES :                5
 NUMBER OF FUNCTION GENERATORS :                84
 NUMBER OF IBUF :                    6
 NUMBER OF MUXF5 :                     20
 NUMBER OF MUXF6 :                     2
 NUMBER OF OBUF :                     91
 NUMBER OF ACCUMULATED INSTANCES : 211
***********************************
DEVICE UTILIZATION FOR 2S30PQ208
***********************************
RESOURCE                USED AVAIL UTILIZATION
-----------------------------------
IOS                98 132 74.24%
FUNCTION GENERATORS 84 864 9.72%
CLB SLICES              42 432          9.72%
DFFS OR LATCHES             5      1296 0.39%
-----------------------------------
                CLOCK FREQUENCY REPORT
            CLOCK              : FREQUENCY
-----------------------------------
            CLOCK             : 72.7 MHZ
                CRITICAL PATH REPORT
CRITICAL PATH #1, (PATH SLACK = 6.2):
NAME                                   GATE          ARRIVAL         LOAD
-----------------------------------COMMAND(0)/                            0.00 0.00 UP
COMMAND(0)_IBUF/O                              IBUF     2.55 2.55 UP       3.70
IX920_IX67_NX10/O                              LUT2      1.74 4.29 UP      2.50
IX920_IX51_NX16/O                              LUT4      1.91 6.21 UP      2.90
NX1042/O                                  LUT4      1.48 7.69 UP      1.90
```

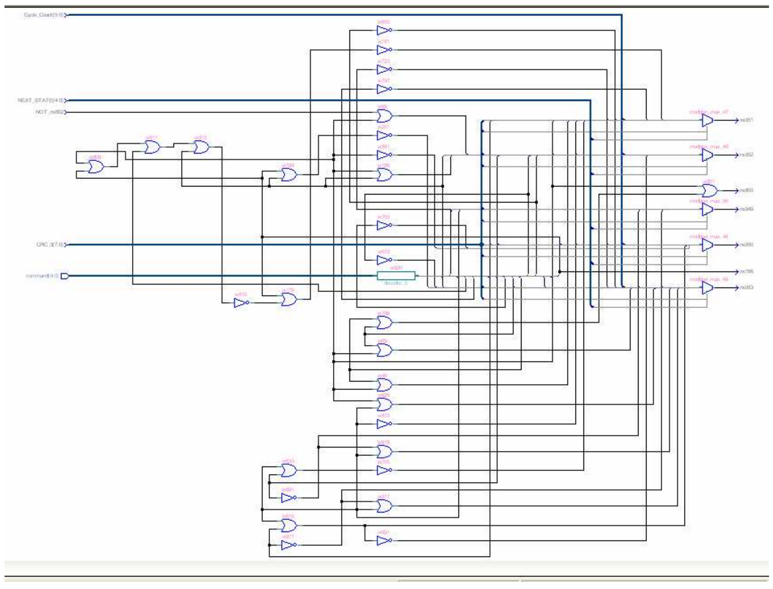


(A)

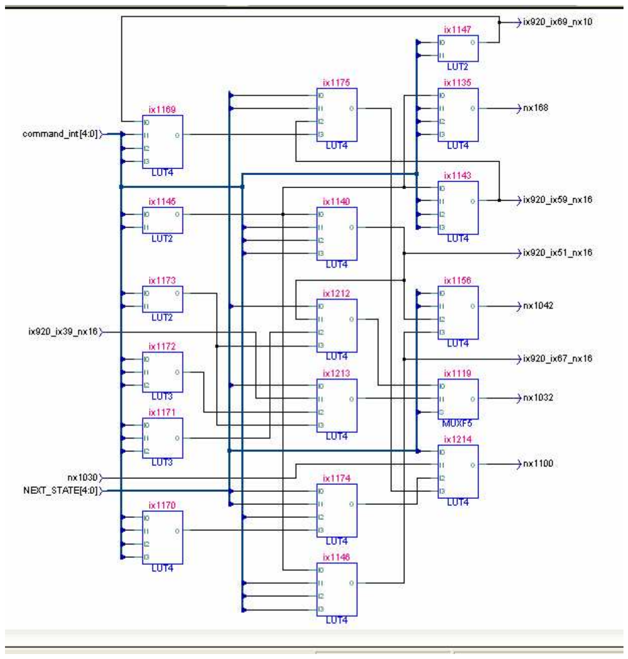


(B)

Figure 13. Xilinx Synthesis Report.



(A)



(B)

Figure 14. Leonardo RTL Synthesis.

```
NX1094/O                            LUT4      1.48 9.16 UP       1.90
IX1113/O                          MUXF5       1.18 10.34 UP         1.90
NX1041/O                            LUT4      1.48 11.82 UP        1.90
NX352/O                           LUT4      1.48 13.30 UP       1.90
REG_NEXT_STATE(1)/D                            FDC       0.00 13.30 UP        0.00
DATA ARRIVAL TIME                                  13.30
                           (DEFAULT SPECIFIED - SETUP TIME)         19.54
DATA REQUIRED TIME
-----------------------------------
```

```
DATA REQUIRED TIME                                  19.54
DATA ARRIVAL TIME                                  13.30
----------
SLACK                                      6.25
```

4. **Conclusions and Future Scope.** This paper has highlighted the concept of the FlexRay protocol. Authors have designed the communication Controller of flex ray node with FSM and the simulation and synthesis results on xilinx tool and synthesis results on Leonardo synthesis tool are presented.

The presented work demonstrated that verification of automotive software is feasible. However, feasibility alone is not sufficient for the method to become accepted by car manufacturers. It is also the question of the development cost that is vital.

The VHDL model Bus guardian module of flex ray node is already designed and verified, in future authors have planned to integrate communication controller and Bus Guardian with suitable host and use this integrated assembly for some intra vehicular communication application.

REFERENCES

[1] Awani Gaidhane, Milind Khanapurkar, and Preeti Bajaj , Design approach for FPGA implementation of flex-ray controller using VHDL for intra vehicular communication application, *Proc. of International Conference ICETET*, G. H. Raisoni College of Engineering, Nagpur, 2008.

[2] P. M. Szecowka, and M. A. Swiderski, On hardware implementation of FlexRay bus guardian module, *Proc. of the 12th MIXDES. FlexRay Communication System-Protocol Specification, v2.0, FlexRay Consortium*, 2007.

[3] *FlexRay Consortium*, Homepage, http://www.flexray-group.org

[4] *FlexRay Consortium, FlexRay Communications System: Protocol Specification Version 2.0*, 2004.

[5] *FlexRay Communications System: Protocol Specification Version 2.1 Revision*; FlexRay; Consortium, 2006.

[6] *FlexRay Communications System Data Link Layer Conformance Test Specification Version 2.1.1*

[7] *ISE 9.1i Release Notes and Installation Guide*, Xilinx Corporation.

[8] *TTP/C-C2 communication controller-preliminary data sheet*, Rev. 16, May 2002.

[9] *Message Handling Concept for a FlexRay Communication Controller-Special Edition FlexRay automotive*, 2004.

[10] *Multiplexed Networks for Embedded Systems CAN*, LIN, FlexRay, Safe-by-Wire, John Wiley & Sons Ltd, 2007.