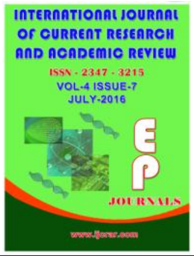
EXCELLENT PUBLISHERS

International Journal of Current Research and Academic Review

ISSN: 2347-3215 Volume 4 Number 7 (July-2016) pp. 76-88

Journal home page: http://www.ijcrar.com

doi: http://dx.doi.org/10.20546/ijcrar.2016.407.010



INTERNATIONAL JOURNAL OF CURRENT RESEARCH AND ACADEMIC REVIEW
ISSN - 2347 - 3215
VOL-4 ISSUE-7
JULY-2016
JOURNALS
www.ijcrar.com

Advanced Level Modelling and Simulation of CAN Controller for its Implementation in FPGA (SoC) with Synthesis and Timing Results for Integrated CAN Node

Duhita B. Paratane[1*], A.M. Patil[2] and Jitendra P. Chaudhari[3]

[1,2]M.E. Electronics and Telecommunication, J.T. Mahajan College of Engineering, Faizpur, India

[3]Charusat Space Rsearch and Technology Center, Charotor University of Science and Technology, Changa, Anand, Gujrat, India

Corresponding author

KEYWORDS

Synthesis, Placement and Routing, Timing constraints, RTL (Register Transfer logic), CAN (Controller area Network) controller, CAN node, FPGA, SoC, HPS, Verilog Modules.

A B S T R A C T

Conventional CAN controller is integrated in the SoC (FPGA + Hard Processor) Chip. The CAN protocol functionality is implemented in the FPGA fabric along with block memory utilisation as buffers and is controlled by ARM Processor (Hard Processor System) present in the same chip (Duhita *et al*., 2016). The CAN controller has been designed using Verilog so it can be targeted with different implementation technologies in custom designs. In this paper the synthesis and timing results of the implemented integrated CAN controller are discussed. The MAC (medium access Control) Sub-layer is designed separately using Verilog HDL and synthesized in FPGA programmable fabric of the SoC Chip. MAC layer i.e. CAN controller logic utilisation and timings along with timing constraints are discussed here. Control and application part of the CAN module is done in hard processor system (HPS) Cortex A9 microprocessor. The HSP system's specifications are also given in this paper. The CAN controller module is designed using Quartus Prime Lite software in Cyclone V SoC "5CSEMA5F31C6N" device.

Introduction

The MAC Sub-layer of data link layer is designed using register transfer logic with the help of Verilog HDL. Data link layer functionality of CAN controller is divided into smaller blocks like Data and Remote Frame generator, Serial frame transmitter, Bit synchroniser for receiver, Arbitration controller, Message process controller etc. These blocks are modelled into RTL logic with help of Verilog HDL. RTL implementation consists of Behavioural and combinational logic implementation of the digital design. All these RTL blocks are structurally combined in the top module to implement the CAN controller functionality completely.

Each design module of CAN controller is optimised with respect its logic utilisation.

Timing constraints are applied to this design to obtain the maximum working frequency and minimum register to register propagation time. Combinational logic between two registers is managed in such a way that register to register propagation time should be very less to achieve maximum possible operating frequency.

This CAN controller module from programmable logic of SoC is connected with the Cortex A9 ARM Hard processor system (HPS) with AXI interface. This connection is done via an AXI to Avalon Memory Mapped bridge. This Avalon MM bridge takes care of arbitration if multiple CAN controllers are connected to single 32 bit light weight HPS to FPGA master.

Controller Area Network (CAN) is a serial communications protocol that efficiently supports distributed real-time control with a very high level of data integrity (Bosch Controller Area Network (CAN) Version 2.0 PROTOCOL STANDARD). It is used in wide variety of embedded application like industrial automation due its advantages such as efficiency, high flexibility, low cost and simplicity [Obermaisser *et al*., 2010]. Home automation system, medical devices, industrial application are new control networks which will use this higher levels of integration into single chip to reduce the size, the power consumption and the price of the final system [de Lucas *et al*., 1999].

The ability of integrating the protocol handling in a single chip together with a microprocessor core, memory and other peripherals are the features of System on Chip concept that are utilised in this design to achieve flexibility of design. It also enhances the performance of overall system since HPS ASIC is combined with FPGA with latest semiconductor fabric design technology.

Higher operation speed due to parallel data processing and high bandwidth interconnect backbone, increase in reaction time of overall system, Real time data processing, configurable priority CAN controller are the special characteristics of this integrated CAN controller design.

Next section describes the detail design of the CAN controller which is based on the CAN protocol versions 2.0A and 2.0B. It explains the functionality of each module implemented in the design. It also gives explanation about design of HPS to CAN controller slave interface module which acts as the bridge between them. HPS functionality of controller data management is also explained below. Beside this several conclusions are drawn at the end.

CAN Controller design

CAN features Overview

CAN Networks have several features which make them well suited for the controller applications. Below mentioned characteristics highlights suitability of CAN networks better than other network protocols.

1. Real time applications with serial bus communication.
2. Easy configuration and modification
3. High reliability and better performance.
4. Cost efficient in designing as well as in implementation.
5. Ideally infinite network size but limited up to 100 running nodes due to protocol restrictions.
6. Higher transmission rate and could be achievable to 1 GHz by integration.
7. Multicast reception with time synchronisation.
8. Non Return to Zero (NRZ) coding with bit stuffing feature.

9. Data security using error detection codes.
10. Automation detection of transmission errors and retransmission feature to avoid data loss.
11. CSMA/CD mechanism implementation to determine the priority and avoid collisions.
12. Event driven communication.

Below given diagram Figure.1 shows the sample of the CAN frame format for the data transmission. Remaining all frames are somewhat similar to this frame.

Data frame of CAN controller could be of two types depending upon the version of protocol it supports. It uses 11 bit identifier for CAN 2.0A and 29 bit identifier for CAN 2.0B in arbitration field as shown in Figure.1.

RTR bit is used to indicate the Remote transmission Request for Remote frame. This frame does not have data in it. Data length code DLC is 4 bit field which denotes the number of bytes of data present in that particular frame. Number of data byte could be from 0 to 8.

Remote frame has same structure as that of the data frame only it does not carry any data in it. It request for the data from the particular node for which it is intended. A node acting as the receiver for certain data can stimulate the relevant source node to transmit the data by sending remote frame.

Detail design diagram of CAN controller is shown in Figure.2. Explanation of each Functional block and its functionality is given below.

Hard Processor System (HPS)

It is the interface application written in C for Host CPU which provides which provides the CAN controller with the data to be transmitted across the CAN bus and also reads the received messages from the controller. Processes the received data and take the actuator action if needed.

Avalon MM Slave interface

This block communicates with the HPS light weight master via 32 bit Avalon Memory Mapped interface. This block is synchronised with HPS clock. It receives the data to be transmitted on CAN bus. It also configures the parameters of CAN controller. And sends the CAN frame received data to Host processor. The interface between the host application and the CAN controller consists of an 8-bit data bus to transfer the message to the controller’s transmit buffer, an 8-bit data read bus which reads the messages received from the controller’s receive buffers and status signals to perform the requisite handshaking.

Parameter Register

The code parameter, mask parameter and the Re-Synchronous Jump Width (SJW) specified for the CAN node are stored in this register block.

Transmit Buffer

There are ten transmit buffers. Each buffer can hold one byte of data. The Avalon MM interface receives the message to be transmitted from the host CPU and stores the message in the transmit buffer before further message processing takes place.

Data and Remote Frame Generator

Data and Remote Frame Generator is responsible for generating the message frame as specified by the CAN protocol. It take the data from the transmit buffer block and generates the required basic CAN frame.

Parallel to serial convertor

This unit serialises the message generated by Frame generator to facilitate the CRC computation

Transmit CRC generator

This module computes the CRC of the serialised message before transmission of the data / remote Frame. The generated CRC frame is appended to the message being transmitted before bit-stuffing is performed.

Bit stuffing Module

This unit performs bit-stuffing as specified by the CAN protocol, making the message suitable for transmission across the CAN network. If five consecutive equal bits arrives in message frame then to create additional signal transitions, and to ensure reliable reception one opposite polarity bit is added during bit stuffing mechanism.

Error and Overload Frame generator

Generates Error or Overload frame whenever error or overload condition occurs. Error containment measures are also taken care of to ensure the accuracy of the controller’s performance and its further participation in the CAN network.

Serialised Frame Transmitter

This unit transmits the data/ remote frame or the error / overload frame or a dominant bit during the acknowledgment slot based on the prevalent conditions.

Message Processor

This is the central unit which provides all the control and the status signals to the various other blocks in the controller. This unit routes the different signals generated in various blocks to the necessary target blocks.

Arbitration Controller

The arbitration controller is responsible for indicating the arbitration status of the node.

Bit Synchroniser

This unit performs the bit timing logic necessary for synchronizing the CAN controller to the bit stream on the CAN bus. The recessive to dominant transition edges present on the received bit stream are used for synchronization and re-synchronization.

Bit De-stuffing Block

This unit performs the de-stuffing of the messages received from the CAN network. This unit also extracts the relevant information from the received message.

Receive CRC generator

CRC generator block also computes the CRC of the received message after message reception. It gives this computed CRC to CRC checker block for message error verification.

CRC Checker

This module compares the generated CRC for the received message with the CRC frame received by the node. An error is generated if the two CRC values do not match. And if there is CRC match then the message forwarded for further processing

Bit Stuff Monitor

This unit signals a stuff error when six consecutive bits of equal polarity are detected in the received message.

Form Checker

A form error is generated if any of the fixed-form fields in a received CAN message is violated. The fixed form fields include the CRC delimiter, ACK delimiter and the EOF field.

Bit Monitor

A CAN node acting as the transmitter of a message, samples back the bit from the CAN bus after putting out its own bit. If the bit transmitted and the bit sampled by the transmitter are not the same, a bit error is generated.

Acknowledgment Checker

During the transmission of the acknowledgement slot a transmitter transmits a recessive bit and expects to receive a dominant bit. If the node receives a recessive bit in the acknowledgement slot an ACK error is signalled.

Acceptance Checker

This unit checks the incoming message ID and determines if the received frame is valid.

Receive Buffer

There are two 10 byte buffers that are used alternatively to store the messages received from the CAN bus. This enables the host CPU to process a message while another message is being received by the controller.

Design and methodology

CAN controller described above is has been designed in verilog environment. The CAN controller and HPS to FPGA interface is converted to Qsys component and the whole system is interconnected in Qsys tool. The Qsys system integration tool saves significant time and effort in the FPGA design process by automatically generating interconnect logic to connect intellectual property (IP) functions and subsystems.

The main aim of this CAN controller is to be reused in different target technologies. Due to the nature of the application the design has been made at the Register Transfer Level, at the architectural design phase.

Figure.3 show the set of software tools used to design the CAN controller. RTL design is written in Notepad ++ editor where the versioning of the source code is done very easily.

For the simulation purpose we have use Altera Modelsim Starter Edition version 10.4b. During simulation we have instantiated four CAN controller and provided input data to all of them simultaneously with different 11 bit identifier. This is done to give the clear understanding of arbitration, overload condition etc.

RTL implementation of the CAN node is done in software Quartus Prime version 15.1.0. in step by step strategy.

In first step HPS-FPGA interface code “can_to_avalon_intf.v” is generated as Qsys Avalon MM slave. Similarly CAN Controller top module is taken into Qsys as the conduit Qsys component. In third step that Qsys components are instantiated in Qsys along with hard processor system (HPS). Then they are connected with each other to generate a CAN Node as shown in Figure 4.

After complete system generation comes the logic realisation and FPGA configuration of the system. It involves Analysis and Synthesis, Placement and Routing, creation of programmable file, System timing analysis and its configuration in FPGA. Each and every step of logic realisation is explained below.

Analysis and Synthesis

During this step compiler analyze the design's verilog files and syntheses the logic written in them. Analysis and synthesis first checks the design files and the overall design for errors, and builds a single design database that integrates all the design files in a design hierarchy. Analysis & Synthesis performs logic synthesis to minimize the logic usage of the design, and performs technology mapping to implement the design logic using device resources such as logic elements. Finally, Analysis & Synthesis generates a single project database integrating all the design files in a design.

Fitter- Placement and Route

Using fitter compiler places and routes the logic of a design into a device which is selected. The Fitter runs an Auto Fit compilation by default. In Auto Fit mode, the Fitter attempts to meet timing constraints and not to beat them; it automatically decreases compilation time by turning off some optimization steps if they are not required for the design to meet the timing constraints. One can also decrease compilation time by directing the Fitter to reduce Fitter effort after meeting a design's timing requirements, or to make only one fitting attempt.

Assembler

Assembler converts logic cell, and pin assignments into a programming image for the device. The Assembler is the Compiler module that completes project processing by converting the Fitter's device, logic cell, and pin assignments into a device programming image, in the form of one or more Programmer Object Files (.pof), SRAM Object Files (.sof), Hexadecimal (Intel-Format) Output Files (.hexout), Tabular Text Files (. ttf), and Raw Binary Files (.rbf), from a successful fit. In our case we use the .sof file to programme and configure our device.

Timing Analysis

The TimeQuest Timing Analyzer software tool analyzes, debugs, and validates the timing performance of all logic in a design. Timing analysis measures the delay of a signal reaching a destination in a synchronous system. The TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that uses industry standard constraint, analysis, and reporting methodologies. The Quartus II Fitter optimizes the placement of logic in the device in order to meet timing constraints. During timing analysis, the TimeQuest analyzer analyzes the timing paths in the design, calculates the propagation delay along each path, checks for timing constraint violations, and reports timing results as slack in the Report pane and in the Console.

Thus the verilog file gets synthesized into the digital logic and the binary file is generated which we can use to configure Cyclone V FPGA to work as the CAN node. After complete compilation of the Quartus prime project the tool generates Analysis and Synthesis report, Fitter report, Assembly report, and timing analysis report.

Synthesis and Analysis report gives information about project design ie top level entity name, device family, logic utilisation, total number of registers, total logic elements, total device pins the logic is using. It also gives the information about number of PLL used, memory blocks required to implement the logic etc.

Fitter report gives the information about logic elements placement and routing. It includes exact values of number resources utilised, percentage of routing resources, total block memory bits, logic partition report, number of preserved nodes, IO assignment report, also reports setting and values for the nominal core voltage, junction temperature report.

Table.1 Synthesis Result of CAN controller and CAN node

Module Name of CAN Controller	**Number of Combinational Logic**	**Number of Sequential registers**	**ALM Needed**	**Comments**
accp_checker:id_check	9	2	5.2	Acceptance Checker logic for identifier
ack_checker:ack_err_chk	2	1	1.3	Acknowledgement checking logic
arbtr_ctrl:arbtr_sts_ctrl	9	2	5.5	Arbitration control logic
bit_destuff:dyestuff	388	209	276	Bit de- stuffing module
bit_monitor:bit_err_chk	3	1	1.8	Bit error monitoring logic
bit_stuff:bt_stf	114	139	129.2	Bit stuffing logic module
bit_stuff_monitor:stf_error	2	1	1.3	Bit stuffing monitor module
crc_checker:crc_err_chk	8	1	4.1	CRC Checker
crc_generator:rx_crc	9	15	7.2	CRC calculator for received message
crc_generator:tx_crc	9	15	6.6	CRC generator for transmitted message
dt_rm_frame_gen:data_remote_frm	254	188	145.4	Data and Remote frame generator module
form_checker:frm_error	7	1	4.3	Form checker module
msg_processor:msg_prsr	25	13	13.4	Message processor logic
ovld_err_frm_gen:ovld_err	136	66	74.7	Overload and Error frame generator module
par_ser_conv:parser	101	97	91.7	Parallel to serial converter module
registers:reg_file	30	47	19.7	Configuration and

				priority setting data storing module
rx_buff:rx_buffer	88	183	76.8	Received message buffer logic
slzd_frm_tx:slzr	7	5	4.3	Serialized frame transmitter on CAN bus
synchronizer:synchro	41	26	25.7	Synchronisation at the receiving end
transmit_buf:tx_buffer	95	96	49.4	Transmit buffer module
CAN Controller total Logic Elements	**1337**	**1108**	**944.3**	CAN controller Top
can_to_avalon_intf	95	258	75.5	Avalon MM Slave logic
altera_reset_controller	0	3	0.0	Reset controller IP
altera_reset_controller	0	3	0.0	Reset controller IP
CAN_node_D_mm_interconnect	340	579		Avalon MM interface logic block IP
CAN Node HPS	1	36	245.7	HPS instantiation IP
Integrated CAN NODE total Logic Elements	**1773**	**1987**	**1266**	CAN node Top

Table.2 Maximum achievable frequency

Clock name	**Clock description**	**Maximum Achieved Frequency**	**Comments**
clk_clk	Input Crystal clock frequency.	170.62 MHz	We can raise maximum input frequency up-to 170.62 Mhz. But since our crystal is of 50 MHz we restrict the input clock constraints to 50 MHz
h2f_user0_clk	CAN controller working frequency	153.3 MHz	We can operate our CAN controller at 150 MHz giving the Bit rate of 480 Mbps
HPS internal clock ‘pll|afi_clk_write_clk’	HPS working frequency	717.36 MHz	Limited due to minimum period restriction

Fig.1 CAN Data frame Format

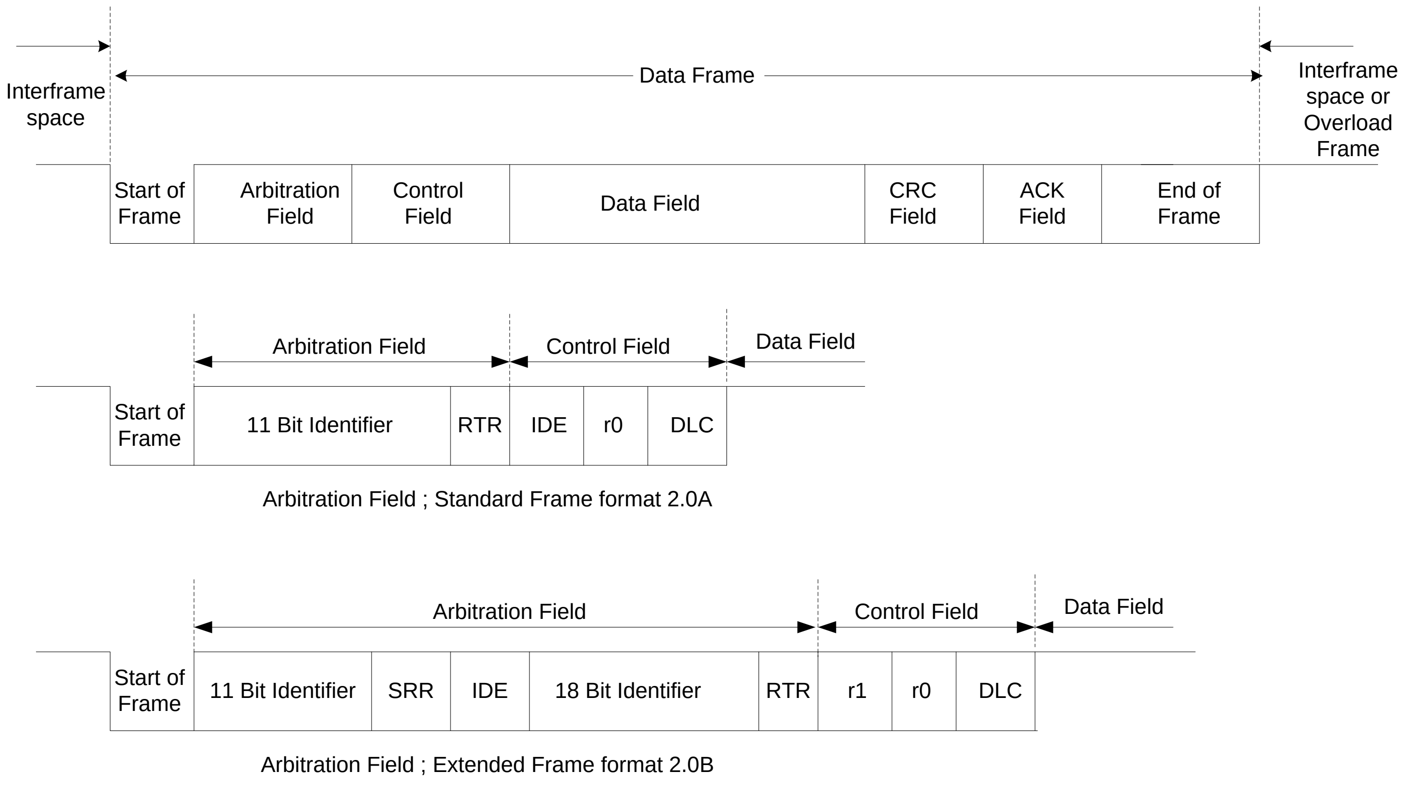


Fig.2 Functional Block diagram of the CAN protocol Controller

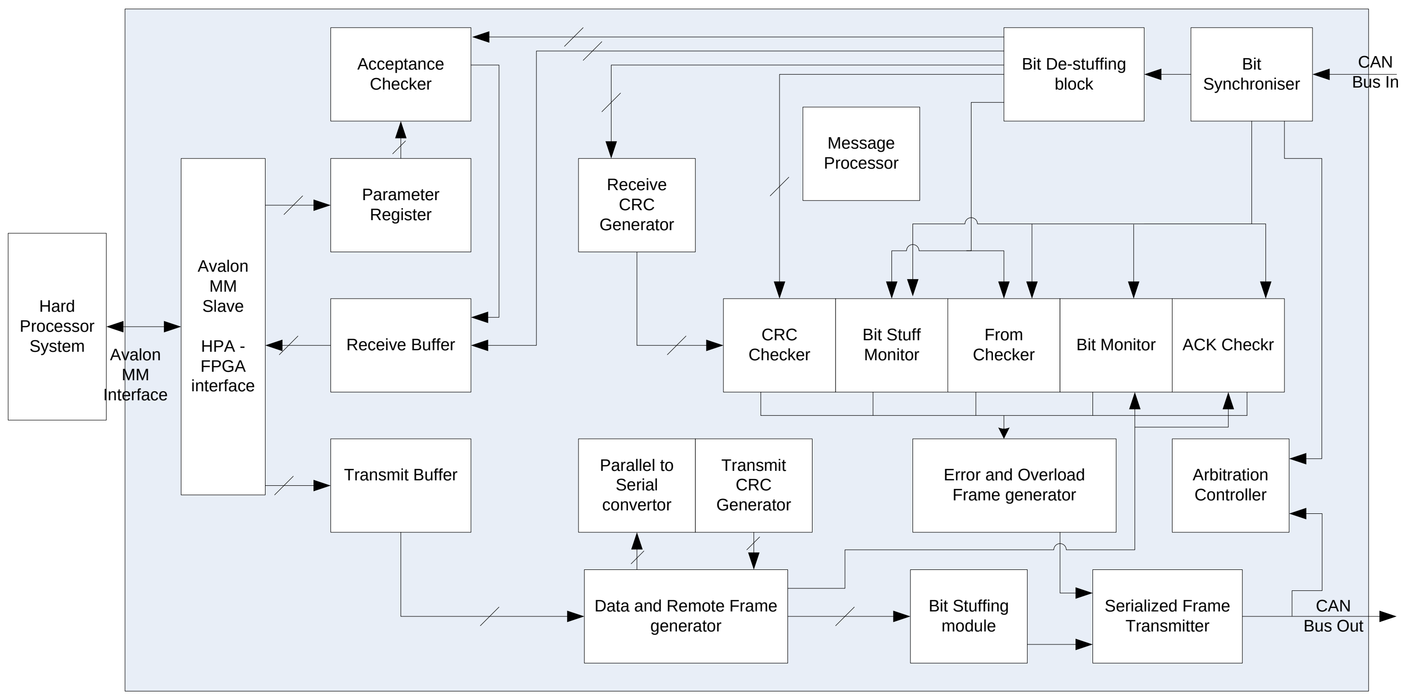


Fig.3 Software tools used in this design

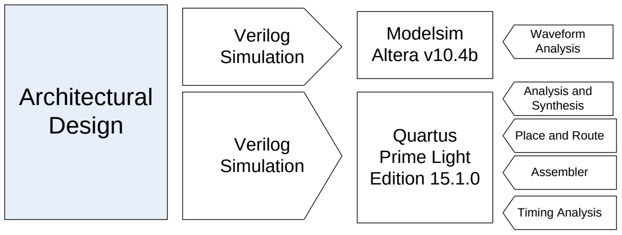
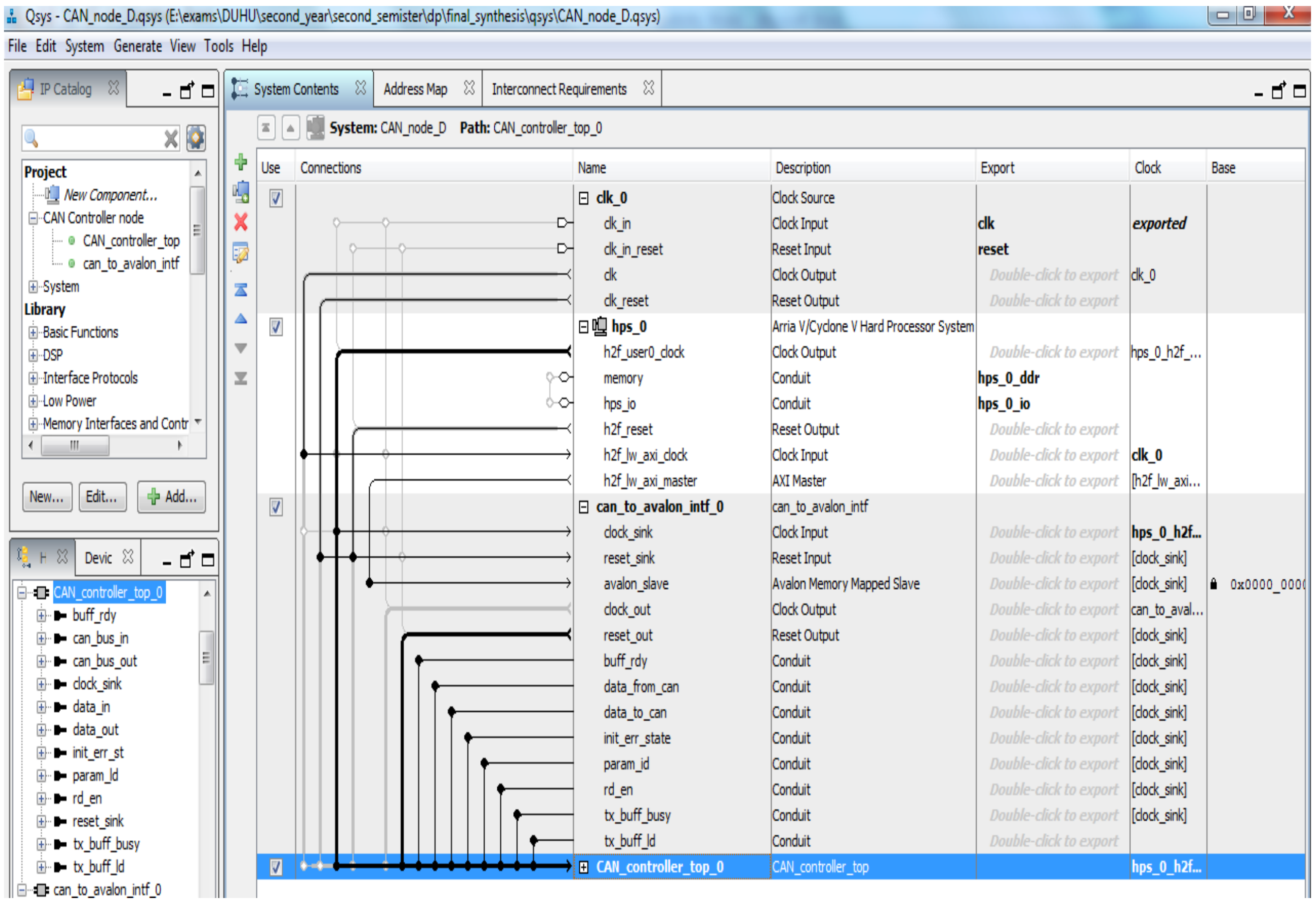


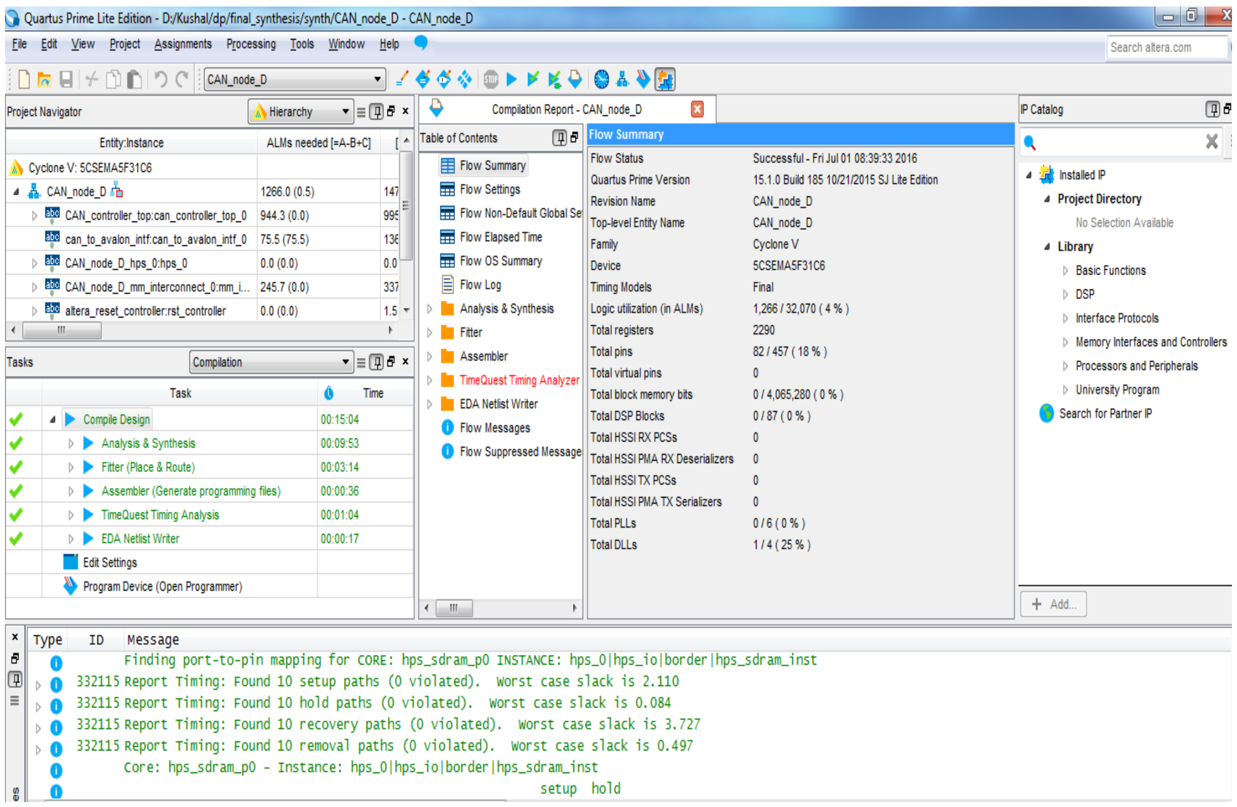
Fig.4 Qsys system project for CAN Node implementation.



System: CAN_node_D Path: CAN_controller_top_0

Use	Connections	Name	Description	Export	Clock	Base
✓		⊟ clk_0	Clock Source			
		clk_in	Clock Input	clk	exported	
		clk_in_reset	Reset Input	reset		
		clk	Clock Output	Double-click to export	clk_0	
		clk_reset	Reset Output	Double-click to export		
✓		⊟ hps_0	Arria V/Cyclone V Hard Processor System			
		h2f_user0_clock	Clock Output	Double-click to export	hps_0_h2f_...	
		memory	Conduit	hps_0_ddr		
		hps_io	Conduit	hps_0_io		
		h2f_reset	Reset Output	Double-click to export		
		h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0	
		h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi...	
✓		⊟ can_to_avalon_intf_0	can_to_avalon_intf			
		clock_sink	Clock Input	Double-click to export	hps_0_h2f...	
		reset_sink	Reset Input	Double-click to export	[clock_sink]	
		avalon_slave	Avalon Memory Mapped Slave	Double-click to export	[clock_sink]	0x0000_000
		clock_out	Clock Output	Double-click to export	can_to_aval...	
		reset_out	Reset Output	Double-click to export	[clock_sink]	
		buff_rdy	Conduit	Double-click to export	[clock_sink]	
		data_from_can	Conduit	Double-click to export	[clock_sink]	
		data_to_can	Conduit	Double-click to export	[clock_sink]	
		init_err_state	Conduit	Double-click to export	[clock_sink]	
		param_id	Conduit	Double-click to export	[clock_sink]	
		rd_en	Conduit	Double-click to export	[clock_sink]	
		tx_buff_busy	Conduit	Double-click to export	[clock_sink]	
		tx_buff_ld	Conduit	Double-click to export		
✓		⊞ CAN_controller_top_0	CAN_controller_top		hps_0_h2f...	

Fig.5 Quartus Project Pane with Synthesis and Timing reports



Quartus Prime Lite Edition - D:/Kushal/dp/final_synthesis/synth/CAN_node_D - CAN_node_D

Project Navigator — Hierarchy

Entity:Instance	ALMs needed [=A-B+C]	[
Cyclone V: 5CSEMA5F31C6		
CAN_node_D	1266.0 (0.5)	147
CAN_controller_top:can_controller_top_0	944.3 (0.0)	995
can_to_avalon_intf:can_to_avalon_intf_0	75.5 (75.5)	136
CAN_node_D_hps_0:hps_0	0.0 (0.0)	0.0
CAN_node_D_mm_interconnect_0:mm_i...	245.7 (0.0)	337
altera_reset_controller:rst_controller	0.0 (0.0)	1.5

Tasks — Compilation

	Task	Time
✓	Compile Design	00:15:04
✓	Analysis & Synthesis	00:09:53
✓	Fitter (Place & Route)	00:03:14
✓	Assembler (Generate programming files)	00:00:36
✓	TimeQuest Timing Analysis	00:01:04
✓	EDA Netlist Writer	00:00:17
	Edit Settings	
	Program Device (Open Programmer)	

Compilation Report - CAN_node_D

Table of Contents: Flow Summary; Flow Settings; Flow Non-Default Global Se; Flow Elapsed Time; Flow OS Summary; Flow Log; Analysis & Synthesis; Fitter; Assembler; TimeQuest Timing Analyzer; EDA Netlist Writer; Flow Messages; Flow Suppressed Message

Flow Summary

Flow Status	Successful - Fri Jul 01 08:39:33 2016
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	CAN_node_D
Top-level Entity Name	CAN_node_D
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	1,266 / 32,070 (4 %)
Total registers	2290
Total pins	82 / 457 (18 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	1 / 4 (25 %)

IP Catalog: Installed IP; Project Directory; No Selection Available; Library; Basic Functions; DSP; Interface Protocols; Memory Interfaces and Controllers; Processors and Peripherals; University Program; Search for Partner IP

```
Type   ID     Message
              Finding port-to-pin mapping for CORE: hps_sdram_p0 INSTANCE: hps_0|hps_io|border|hps_sdram_inst
       332115 Report Timing: Found 10 setup paths (0 violated).  Worst case slack is 2.110
       332115 Report Timing: Found 10 hold paths (0 violated).  Worst case slack is 0.084
       332115 Report Timing: Found 10 recovery paths (0 violated).  Worst case slack is 3.727
       332115 Report Timing: Found 10 removal paths (0 violated).  Worst case slack is 0.497
              Core: hps_sdram_p0 - Instance: hps_0|hps_io|border|hps_sdram_inst
                                                        setup  hold
```

Assembler report contains device options of FPGA configuration such as SRAM based device or EEPROM based device option. It also reports the file generated by the assembler during the current compilation. Assembler reports the error correction CRC results for the design when it is turned on in the Device and Pin Option Dialog box.

Timing analysis reports setup time values, hold time values, and timing violations if present, minimum clock to output time which corresponds to maximum frequency, longest propagation delay of combinational logic. And finally it gives the multi-corner timing analysis summary reports.

Figure.5 shows the CAN controller design project Quartus II diagram with all reports generated in the column Table of contains and the design is successfully compiled.

Synthesis and Timing Results

The design presented in this paper is implemented in verilog HDL. Described RTL logic is synthesized in Quartus Prime 15.1 software. As a result of this project FPGA configuration file "CAN_node_D.sof" is generated which is used for programming and configuring the Cyclone V SoC Chip.

Table.1 results are obtained from Quartus Prime HDL Compiler fitter report section. It shows logic utilisation of each block of CAN controller in terms of number of Adaptive Logic Module (ALM) after synthesis. ALM's are made up of Look up tables, multiplexer, adder i.e. combinational logic and Flip-Flops i.e. sequential logic. Hence this table gives both logic elements required in implementation of various logic blocks.

The Verilog codes are combined with Qsys design and its AHDL source code which have several advantages for design and for designer as well. First advantage is designer has to just graphically connect the blocks with custom design which reduces complexity and time required of instantiating these module in separate source code. Second advantage is it become easier to find out missing connections and design error while debugging the design.

Timing results for the design are given in Table.2. These results are obtained from Time Quest Timing analyzer which works on synopsys design constraints formats. These timing constraints to the FPGA design gives the maximum operating frequency, setup and hold violation reports for that particular design.

From the results we can raise our input frequency to 170 Mhz. But this frequency is obtained from the external crystal oscillator so we have to restrict it to 50 Mhz. But we can operate our CAN controller at 150 MHz which we are operating at 50 MHz. At 50MHz our bit rate is 160 Mbps but as the frequency is raised to 150 Mhz the bit rate becomes 480Mbps. But for this design bit rate is restricted to 160Mbps

In conclusion, design of CAN controller for integrated CAN node is successfully synthesize and fit into Cyclone V SoC FPGA. This node design is successfully implemented on DE1-SoC development and educational board and the can bus signal is observed in signal tap logic analyzer for its validity. All the results of this implementation are reported in this paper. The implementation result of placement and routing report shows that a complexity of logic utilisation is 1266 which is 4% of the actual logic presents in the FPGA. With this kind of implementation we can implement such 25 number of CAN controllers in one FPGA. Also the rate of bit transmission could be increased up to 480 Mbps which is very much higher than the conventional CAN controller.

References

Barranco, M., Proenza, J. 2006. “An Active Star Topology for Improving Fault Confinement in CAN Networks” *Industrial Informatics,* Pages 78-85. IEEE

Bosch Controller Area Network (CAN) Version 2.0 PROTOCOL STANDARD.

Carvalho, F.C., Jansch-Porto, I., Freitas, E.P. 2005. “The TinyCAN: an optimized CAN controller IP for FPGA based platforms” *Emerging Technologies and Factory Automation,* Pages 4 pp. – 374 IEEE.

de Lucas, J., M. Quintana “Design of CAN interface of custom circuits” Industrial Electronics Society, 1999. *IECON '99 Proceedings* Pages 662 - 667 vol.2 IEEE.

Duhita, B., Paratane, A.M. Patil. 2016. “CAN Controller Modelling and Simulation for its Implementation in FPGA (SoC) using Verilog for Integrated CAN Node” *Int. J. Curr. Res. Aca. Rev.*, 4(3): 76-85.

Dzhelekarski, P., Zerbe, V. 2004. “FPGA Implementation of Bit Timing Logic of CAN Controller” Electronics Technology: Meeting the Challenges of Electronics Technology Progress, 2004Pages 214 - 220 vol.2 IEEE.

Kyung Chang Lee, Hong-Hee Lee “Network-based Fire-Detection System via Controller Area Network for Smart Home Automation” Consumer Electronics Page 1093 - 1100 IEEE Consumer Electronics Society.

Obermaisser, R., R. Kammerer. 2010. A router for improved fault isolation, scalability and diagnosis in CAN. Industrial Informatics (INDIN), 8th IEEE International Conference Pages 123 – 129 IEEE.

Reges, J.E.O., Santos, E.J.P. 2008. "A VHDL CAN module for smart sensors" Programmable Logic, 4th Southern Conference Pages 179 – 182 IEEE.

Salem Hasnaoui, Oussema Kallel. 2003. "An Implementation of a Proposed Modification of CAN protocol on CAN Fieldbus Controller Component for Supporting a Dynamic Priority Policy" Industry Applications Conference, Pages 23-31 Vol.1 IEEE http://www.can-cia.de 2004. Homepage of the organization CAN in Automation (CiA).

SoC Product Brochure- Altera's user customizable ARM-Based SoC from www.altera.com/products/soc/overview.html.

How to cite this article:

Duhita B. Paratane, A.M. Patil, Jitendra P. Chaudhari. 2016. Advanced Level Modelling and Simulation of CAN Controller for its Implementation in FPGA (SoC) with Synthesis and Timing Results for Integrated CAN Node. *Int.J.Curr.Res.Aca.Rev.*4(7): 76-88.
doi: http://dx.doi.org/10.20546/ijcrar.2016.407.011