Design and verification of a CAN controller for custom ASIC

Namsub Kim (speaker), Dawi Kim, Kyuhyung Cho, Jinsang Kim, and Wonkyung Cho

This paper presents a novel architecture and verification model of the CAN protocol controller for ASIC implementation. The key features of the proposed CAN controller are flexibility in terms of interfacing with host processors and smaller chip size. Also, the architecture is efficient for Intellectual Property (IP) reuse because of its flexibility and synthesis efficiency. For verification of the designed controller, we developed a verification model for fast verification during the design phase. The gate counts of the core logics in the proposed CAN controller are 3189 gates which are much smaller than other controllers. After successful verification, the CAN controller was fabricated by using 0.35 CMOS process.

1. Introduction

The Controller Area Network (CAN) is data communication protocol widely used in various application areas such as automobile, medical equipments, and many industrial applications [9]. Typically, one CAN system needs many CAN controllers and the CAN controller is indispensable for each CAN node, so that it is desirable to implement CAN controller as inexpensive as possible. There is an approach to make a low-cost communication controller called LIN (LIN: Local Interconnect Network) in conjunction with CAN. LIN operates at data rates up to 20 Kbit/s and is based on a common UART/SCI interface hardware [11]. The method for making low cost silicon in LIN is mainly focused on using simplified protocol and low cost physical layer device. Therefore, its usage is restricted only to low cost communication system and it could not be replaced to all CAN controllers.

There have been many CAN controllers widely used in commercial market and some of them are provided as an IP. From the viewpoint of IP, it is important for the design to be reusable, flexible, reliable, and easy to implement [13]. Since the design methodology and implementation of the CAN controller have been mainly depended on the designer’s experience, it is required to make efficient architecture which satisfies the aforementioned features.

Moreover, since the upper service boundary of CAN implementation is not standardized, the development of efficient verification method is required [8].

In this paper we propose a CAN controller which makes low cost silicon and efficient for IP reuse. Also, efficient verification process of the CAN controller for rapid prototyping is presented.

This paper is organized as follows:
Section 2 describes the proposed CAN controller architecture and its operation. Section 3 describes the verification method of the CAN controller. Section 4 describes ASIC implementation details and comparisons with previous CAN controllers. Finally, concluding remarks and future work are given in section 5.

2. Proposed CAN Controller Architecture

CAN controller can be classified by its mailbox structure [9]. Looking at the mailbox structure, basically there are two forms of structure type that are called BasicCAN and FullCAN. BasicCAN can be implemented with smaller size than FullCAN. However, the main disadvantage of BasicCAN is that it might give a host processor large load when the controller needs to gather large amount of data. FIFO Buffer CAN controller such as Philips SJA1000 could solve the problem of BasicCAN structure, but the chip size was large because of large amount of FIFO [10]. There had been an approach to use external memory which is operating similar
to FIFO [2]. This CAN controller could reduce the size of CAN core, but it has an additional burden of implementation of external memory and is not good for using as an IP.

Typically, the CAN protocol controller can be implemented as shown in Figure 1 [12].

![Figure 1. Basic architecture of CAN protocol controller]

This architecture describes well the data flow of signals and components needed for implementation. However, because Bit Timing Logic and Bit Stream Processor are heavily related, it is very hard to debug the hardware during the design phase. Moreover, because the error handling and control mechanism in CAN protocol is very complex, the complexity of Bit Stream Processor is very high. Therefore, the implementation depends heavily on the designer’s technique and the design is not suitable for IP reuse. Moreover, in order for the design to change the interface with host processor, the whole design must be re-designed.

For solving the foregoing problems, we separated bit timing operation as independent block. The role of Bit Stream Processor is divided into Field Manager, Error Checking, and Data Generation blocks according to the specification of CAN protocol. Because error management scheme in CAN protocol is complex, we separated each error management blocks and it could lead our architecture to be compact in size. Also, we used FIFO that can be variable in size in order to prevent problem known as “inner priority inversion” [6]. The proposed architecture is shown in Figure 2.

This architecture is a BasicCAN with FIFO buffer, someone called it as intermediate CAN controller, which is compatible to CAN version 2.0A. The functional description of each block is as follows

- Synchronizer synchronizes RX data with bit timing.
- ID Checker checks incoming ID and determines valid frame.
- Register Group stores bit timing parameters and host parameters.
- Interface Logic controls between host and CAN core.
- RX FIFO stores valid incoming data.
- Field Manager controls valid incoming data.
- Error Checkers consist of checking blocks for each type of errors specified in ISO 11898-1 and informs errors to Error Counter.
- Error Counter counts errors.
- Data & Remote Frame Generator generates frame.
- Error & Overload Frame Generator generates frame whenever error or overload condition occurs.
- Serializer serializes parallel data.

In this architecture, ID Checker is operating similar to acceptance filter of previous designs. Error checking blocks are separated. These separated blocks make the architecture easy to modify and test.

Our architecture can be easily upgraded to CAN version 2.0B and interface logic can be realized with simple structure as shown in Figure 3. Figure 3 shows synthesis result of Interface Logic targeting to 8051 microcontroller. It can be adapted to any systems with small modifications of Interface Logic and FIFO size.
3. Verification Methodology

Verification is an important topic in designing VLSI circuits. There has been a previous paper work for verifying CAN protocol by using a verification system called Murϕ [6]. The paper uses verification structure composed of one transmitter and many receivers, where the receiver can be varied according to the parameter N. Since Murϕ uses Murϕ compiler and Murϕ description language, it needs additional work during Hardware Description Language (HDL) simulation.

The proposed method we used consists of three steps. In the first step, we developed a simulation model for verifying the CAN controller as shown in Figure 4.

The Virtual Host Model is operating independently and handles transmission of data or remote frames. It also checks whether host interfacing is working correctly. The Virtual CAN Controller transmits error frames which can not be generated by the Design Under Test (DUT). Bus Model can force the CAN bus to be erroneous and monitor the verification results. In this simulation model, each CAN controller sends or receives messages. This means any of CAN controller can be a master. Also, this simulation model can not only test the controller’s behavior but also check the interface to the host processor.

Since the verification model shown in the Figure 4 was written in verilog HDL, we could test our CAN controller during any time of the design phase. As a result, the prototyping time has been greatly reduced. Figure 5 shows simulation example using the verification model.

The second step of our verification is the ISO conformance test specified in ISO 16845. Conformance test is necessary for testing interoperability between different implementations. In this test, we used the architecture of test plan (TP) described in ISO 16845 and tested our design step by step according to ISO 16845 document.

ISO 16845 is not perfect as mentioned in the previous paper [6]. Therefore, as a third step of our verification, we made a FPGA verification system to check the design feasibility for real time application as shown in Figure 6.

For the prototyping of the design, Altera’s Excalibur Development Board was used. We made two other test boards that can send temperature and motor speed data. 8051 microcontrollers were used as host processors. Temperature and motor speed data generation boards are designed by using commercial CAN controller.
The FPGA Verification System can change the CAN protocol controller from our design to commercial product. Therefore, we could test interoperability between our design and existing CAN controllers in board level. The host computer monitors and controls data communications on the CAN BUS using RS-232 serial port. The clock frequency used in this system was 24MHz and the CAN transmission rate was 1Mbit/s.

4. ASIC Implementation and Comparisons

The overall ASIC implementation flow of the designed CAN controller is depicted in Figure 7.

Implementation process consists of front-end design and back-end design. In the front-end design phase, proposed architecture was modeled using verilog HDL and simulated the CAN functions with our verification model.

In the back-end design phase, synthesized gate level netlist was translated into silicon layout after placement and routing process. Because of parasitic capacitance and resistance of interconnection of each gate, we extracted Standard Delay Format (SDF) file and simulated once again with this SDF information. After overall verification had been completed, we generated the final layout as shown in Figure 8 and the layout was fabricated by using Hynix 0.35 _ CMOS process.

For the comparison of our design result with various other designs, we investigated previous papers which are similar to our work [1, 2, 4]. For fair comparison, only the core gate counts are compared as shown in Table 1.

Table 1. Comparison of gate counts

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Gates</td>
<td>3199</td>
<td>11824</td>
<td>6840</td>
<td>5750</td>
</tr>
<tr>
<td>Type</td>
<td>BasicCAN</td>
<td>BasicCAN</td>
<td>BasicCAN</td>
<td>Not mentioned</td>
</tr>
<tr>
<td>Memory Structure</td>
<td>FIFC Registers</td>
<td>Dual-Port Memory</td>
<td>External Memory</td>
<td>Dual-Port RAM</td>
</tr>
</tbody>
</table>

Gate count of each block as shown in Table 2 has been obtained using Synopsys Design Compiler and Hynix library. Because we used timing
optimization options, the result of synthesis contains additional buffers and interconnection area. As shown in Table 2, each block of the proposed CAN controller could be implemented below 1000 gates, which means the design is efficient on synthesis and easy to modify. Because the memory implementation is not relevant to synthesis, we excluded memory comparisons on both sides.

Table 2. Block synthesis comparison

<table>
<thead>
<tr>
<th>Function Blocks of the Proposed CAN Controller</th>
<th>#gates</th>
<th>Functional Blocks of the paper [2]</th>
<th>#gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Logic</td>
<td>100</td>
<td>bit timing</td>
<td>302</td>
</tr>
<tr>
<td>ID Checker</td>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Manager</td>
<td>627</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronizer</td>
<td>594</td>
<td>bit stuffing</td>
<td>212</td>
</tr>
<tr>
<td>Serializer</td>
<td>176</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data &amp; Remote Frame Gen.</td>
<td>364</td>
<td>message proc.</td>
<td>3368</td>
</tr>
<tr>
<td>Error &amp; Overload Frame Gen.</td>
<td>207</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC Calculator (Including CRC Error Checker)</td>
<td>325</td>
<td>transmission</td>
<td>1098</td>
</tr>
<tr>
<td>Error Counter</td>
<td>552</td>
<td>error management</td>
<td>1186</td>
</tr>
<tr>
<td>Stuff Checker &amp; Stuff Generator</td>
<td>132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Checkers (Ack, Bit, Form)</td>
<td>50</td>
<td>CRC</td>
<td>674</td>
</tr>
<tr>
<td>TOTAL</td>
<td>3189</td>
<td>TOTAL</td>
<td>6840</td>
</tr>
</tbody>
</table>

5. Conclusions

In this paper, we designed a stand-alone CAN controller which is compatible to CAN Version 2.0A. The gate counts of core logic of the designed CAN controller was 3189 gates which is much smaller than other CAN controllers. Also, the implemented CAN controller can be easily upgraded to Version 2.0B and has the flexible interface logic in order to make it possible for interfacing with various host controllers. Because the proposed controller is flexible and easy to modify, it is suitable for IP. Furthermore, we tested the CAN controller with various methods and the results were successful. The proposed architecture was fabricated by using 0.35 _Hynix CMOS process and packaged with 100 Pin MQFP. The result chip has 38.3712mW dynamic power and maximum 25MHz operating frequency.

Since the need for System On a Chip (SoC) has been increased, the proposed CAN controller can be embedded into a SoC. Therefore, a further study and technology development is necessary in order to harness the potential of CAN protocol.

Acknowledgement

This work was supported by Kyunghee-Davan ASIC Center at Kyung Hee University.

References


About the author:
Namsub Kim received B.S and M.S degree in electronic engineering from the University of Kyung Hee, Korea, in 1990 and 1992. From 1994 to 1996, he was a professor in engineering department at Korea Naval Academy. From 1996 to 1998, he worked at Hynix Semiconductor Corporation. He is at present a senior researcher at Davan-Kyunghee ASIC Center, and part-time instructor in electronic engineering at the University of Kyung Hee.