Design of a CAN interface for custom circuits

J. de Lucas, M. Quintana, T. Riesgo, Y. Torroja, J. Uceda

Universidad Politécnica de Madrid
E.T.S.I. Industriales. División de Ingeniería Electrónica
c/ José Gutiérrez Abascal, 2. 28006 Madrid (Spain)
Telephone: +34-91-3363191; Fax: +34-91-5645966
e-mail: {jlucas, maider, teresa, yago, uceda}@upmdie.upm.es

ABSTRACT. This paper presents the experience of the development of a CAN interface to be integrated in custom circuits (ASICs, FPGAs). The CAN controller has been designed in VHDL, so it can be targeted to different implementation technologies. The design method of the block is also presented in the paper as an experience of how to easily design digital systems to be reused in different applications, assuring its quality and reliability.

I. INTRODUCTION

CAN (Controller Area Network) is a standard protocol for control networks [1]. It was initially developed for control networks in automobiles, but now its is being used for other control applications as home systems, medical devices, industrial control, etc. [2]. The interest in CAN is increasing rapidly due to the different applications that are foreseen and the availability of devices integrating CAN in the market. Moreover, a higher demand could be addressed with the new emerging technologies related to totally networked environments in factories, at home, etc. where control network protocols will be required, rather than data networks.

Some of these applications will require higher levels of integration to reduce the size, the power consumption and the price of the final system. Besides, an additional requirement will be the ability of integrating the protocol handling in a single chip together with a microprocessor core, memory and other peripherals, what is called a *System-on Chip*.

The current trend to design Systems-on Chip implies the ability of having a library of "components", that can be called IPs (Intellectual Properties), macrocells or virtual components. For the design to be really reusable (for different applications, different implementation technologies, for different users), it is required that the design is made considering its reliability, controlling both the quality of the development phase and the final product quality [3]. On one hand, the use of hardware description languages (HDLs) is a must to develop both the model of the system and the test-benches to validate it. On the other hand, a real and practical reuse of hardware models requires additional features, such as:

- Independence of the implementation technology
- High quality HDL models with test benches
- Optimization in timing and area
- Good documentation
- Support for a test strategy

Therefore, two main goals drive the developments presented in this paper. On one hand, the design of a reusable hardware part for a CAN interface, developed in a hardware description language, VHDL in this case [4]. On the other hand, the demonstration of a method supported by a set of tools that allow the design-for reuse.

The reusability features will be analysed from the VHDL model, using the tools and the methods developed within a project called TOMI [5]. The particular aspects of these tools will be described in more detail in a later section.

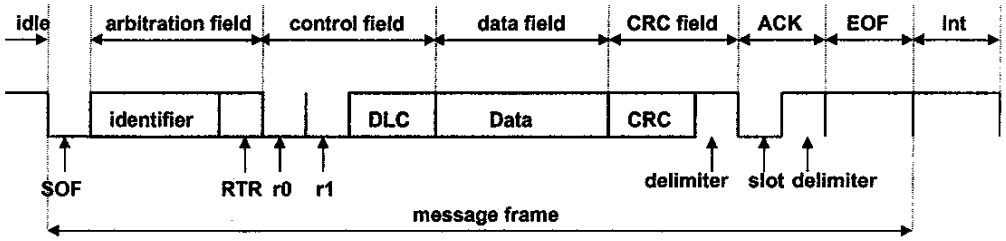
Next section briefly describes the features of the CAN protocol and the functionality of the component under design. The design methodology and the use of design tools for reuse will be described in section 3. Section 4 shows the results of the design. At the end several conclusions are drawn.

II. THE CAN CONTROLLER

CAN networks have several features that make them well suited for control applications. Among these features, the main ones are the following:

- Serial bus communication for real time applications
- Cost-efficient both in design and implementation
- Easy configuration and modification
- High reliability even in hostile environments
- Typical network size could be between 32 to 64 nodes, running up to 100.
- Transmission rate up to 1 MHz
- Multicast reception with time synchronization
- Multi-master node hierarchy, in a way that if a node fails the whole system does not collapse

0-7803-5735-3/99/$10.00 ©1999 IEEE



SOF:	Start Of Frame	RTR:	Remote Transmission Request (1 bit)
r0, r1:	two dominant bits	DLC:	Data Length Code (4 bits)
CRC:	Cyclic Redundancy Code (15 bits)	ACK:	Acknowledge (2 bits)
EOF:	End Of Frame (7 recessive bits)	INT:	Intermission period (3 bits)

Fig. 1. CAN message format

- Event driven communication
- NRZ (*Non-Return to Zero*) coding with bit-stuffing
- To determine the priority of messages a CSMA/CD mechanism is used (*Carrier Sense Multiple Access with Collision Detect*)
- Automatic detection of transmission errors
- Message name (*identifier*) designates the information, but not the address of the node

Fig. 1 shows the format of data message frames for the CAN protocol.

The message frames can have different lengths, depending on the type of identifier used (11-bit for CAN2.0A or 29-bit for CAN2.0B), and on the length of the transmitted/received data, specified in the DLC field. The number of data bytes in a transmission may vary from 0 to 8 bytes.

Another important advantage of CAN is the large number of available components in the market, normally integrated as a peripheral port of a standard microcontroller. However, there is also a need of having such a peripheral block in an ASIC (Application Specific Integrated Circuit) or a FPGA (Field Programmable Gate Array) and this is the main reason of the implementation of the CAN controller we are describing in this paper.

The block diagram of the CAN controller to be designed is shown in Fig. 2. The following subsections describe in more detail each block of the circuit.

II.A Bit Stuffing Block

The data blocks arrive in a serial bit stream and they are coded by the method of "bit stuffing". After five bits with the same polarity a stuff bit with opposite level is inserted to force the necessary edges to be resynchronized. The stuff bits are filtered from the received bit stream and the coded data is transmitted to the interface module.

II.BCyclic Redundancy Code Generator and Comparator

Each message is provided with a 15-bit-long CRC code. This code is generated from the preceding telegram sections (start of frame, arbitration field, control field, data field). When receiving a message frame, the code is generated analogously from the received data and compared with the CRC field in the message. This implies an error protection for the messages through the network.

II.CError Management Unit

The Error Management Unit is responsible for the fault confinement of the CAN protocol, which contains a mechanism for the automatic location and switching-off of defectives nodes. This mechanism is performed with two error counters: Receive Error Counter and Transmit Error Counter.

The Error Management unit is also responsible of changing the error counters, setting the appropriate error flag bits and interrupts and changing the error status (active, passive and bus off).

II.D Message Processor

This block is in charge of the identification of the different message fields (start of frame, arbitration field, control field, data field, etc.). The message processor cuts up the received message and inserts the bits in the corresponding registers. It is also responsible for building the messages to be transmitted: it creates the message from the different fields, which are taken out of the application.

The message processor takes care of the acceptance filtering, offering different possibilities. One or several identifiers can be indicated explicitly (*Full-CAN*) or it is operated with a mask register (*Basic-CAN*). The implementation we are developing will use the *Basic-CAN* strategy. It takes the received identifier to be compared with a register of acceptance filter. If they are not identical, the received message will be rejected.

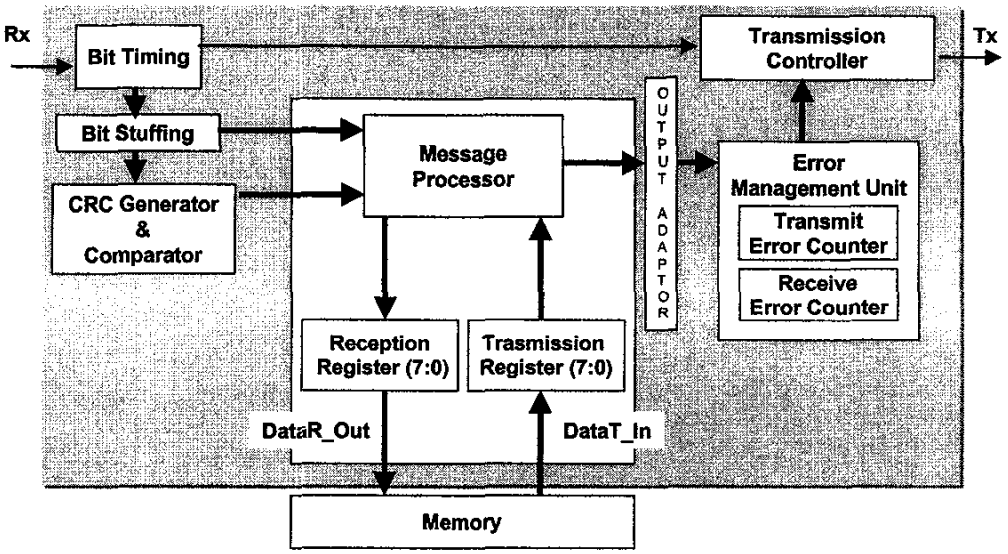


Fig. 2. Block diagram of the CAN controller

II.E Transmission Controller

The Transmission Controller block controls the output drivers and the synchronization of the bus with the CAN clock, generated by the *Bit Timing* block. It is the final responsible of the transmission: it controls the transmitted message, which arrives from the message processor, the error counters and the bit timing logic.

II.F Bit Timing

This block is in charge of the synchronization of all the CAN nodes. The length of a bit is determined by the following parameters, as shown in Fig. 3:

- *Synchronization Segment* (*SyncSeg*): This part of the bit time is used to synchronize the various nodes on the bus.
- *Propagation Time Segment* (*PropSeg*): It is used to compensate the physical delay times within the network.
- *Phase Buffer Segment1* (*PhaseSeg1*) and *Phase Buffer Segment2* (*PhaseSeg2*): These segments are used to compensate for edge phase errors. They can be lengthened or shortened by resynchronization.

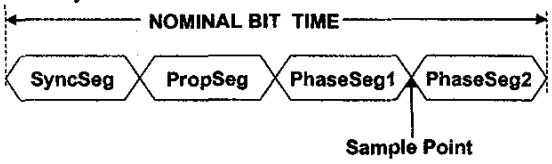


Fig. 3. CAN bit timing

All controllers on the CAN network must have the same data rate and bit length. At different clock frequencies of the individual controllers, the data rate has to be adjusted by the mentioned parameters. The resulting bit time can be computed by the following formula:

$$BitTime = m \cdot (SyncSeg + PropSeg + PhaseSeg1 + PhaseSeg2) \cdot Clk$$

being m the value of the clock prescaler.

II.G Interface Management

The data coming from the application are sent in parallel to the main controller through an 8-bit data bus. These data blocks are converted into a serial bit stream. The interface module is the link between the basic module and the microcontroller. The interface module makes available on the transmit data bus the data or control bits needed for the respective section of a transmission (send) or takes over already contained information when receiving a message.

III. Design Methodology

The CAN controller described above has been designed with VHDL-based design methodology. The main aim of this "virtual component" is to be reused in different applications and to be implemented 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.

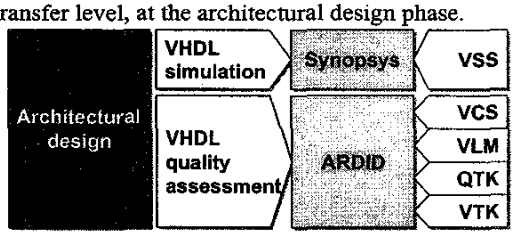


Fig. 4. Main tools used in the design

Fig. 4 shows the set of tools used in the design of the CAN controller. ARDID is a graphical front-end

environment specially designed to work with VHDL designs, and lies on the LVS system of LEDA [6]. ARDID has been developed by UPM/DIE within the TOMI project to assess designers and design managers in the development and use of virtual components.

The tools included in ARDID are the following. The first two are mainly related to design management, while the last ones are the key part of the quality assessment tools. All these tools are integrated under the same graphical interface, which allows an easy and simple use, by the design team.

- VCS (*Version Control System*), which allows an easy way to maintain the versions of the VHDL source code.
- VLM (*VHDL Library Manager*), which is a graphical library manager integrated in ARDID, including functions such as library creation and management, dependency analysis of VHDL files, interface with external tools, etc..
- QTK (*Design Quality Tool Kit*), which aims to detect design methods that are likely to produce difficulties in the reliability and reusability of the VHDL model under design. QTK checks the design compliance with a set of rules. This tool kit will be explained later in more detail.
- VTK (*Validation Quality Tool Kit*), which is charge of measuring the quality of the test-benches used to validate the functionality of the VHDL code. It consists of an "error simulator" that gives a quality measure based on error detection [7].

Fig. 5 shows the graphical user interface of the ARDID tool. The window of the right part of the figure shows the results of the QTK checkers for the design. The window on the left shows the library manager.

The use of ARDID in the design process has been very useful in the debugging of the VHDL codes, especially the QTK tools. The main checkers included in this tool that have been used in the design of the CAN controller are the following:

- *Sensitivity list analysis*, which detects omission of signals in the sensitivity lists of the VHDL process that may cause different behaviors in the pre and post synthesis circuit.
- *Architectural description style*, which informs of mixed structural-behavioral descriptions that may cause difficulties in the code maintenance and problems in the synthesis process.

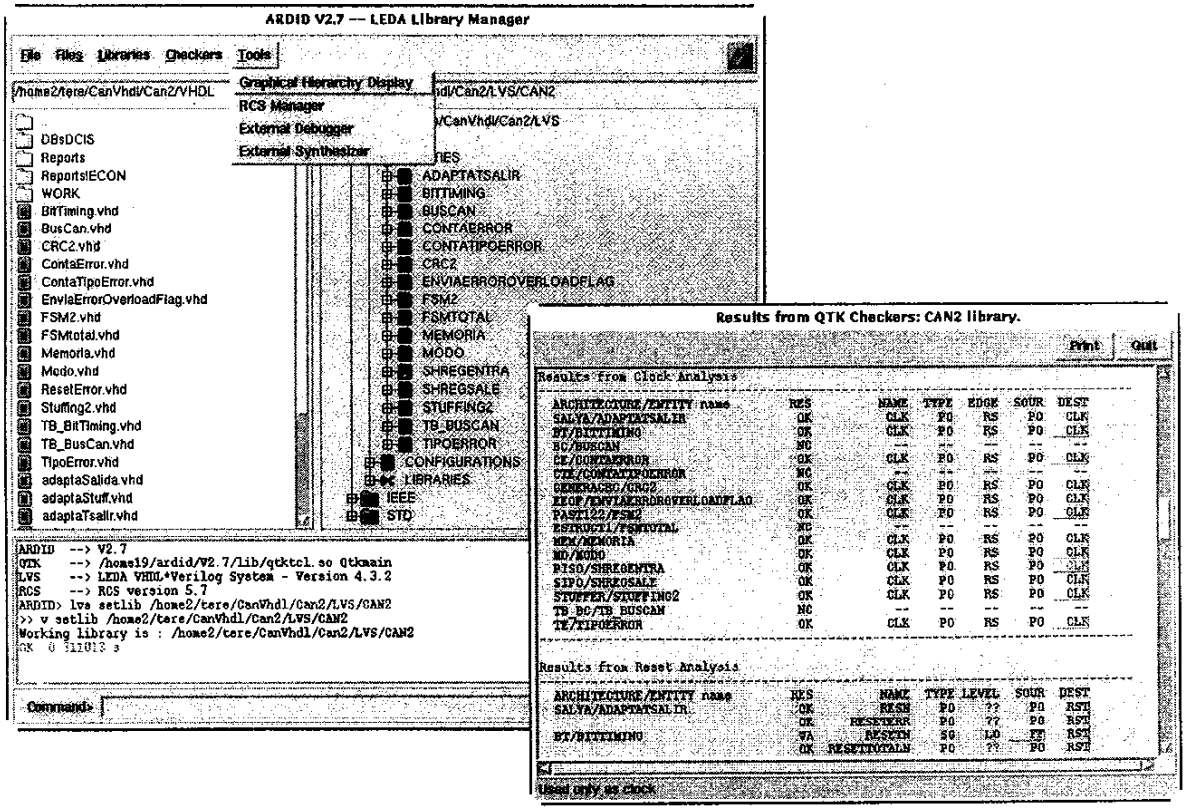


Fig. 5. Graphical user interface of ARDID

- *Object usage*, which detects if the all objects in the code are really used in the description. This checker has been very useful in the early stages of the code debugging by the designers and by the project manager.
- *Clock and reset analysis*, which is one of the most important issues in synchronous design. The tool gives a feedback of the active edges and the clock for each registered signal, previous to the synthesis process. With this tool it is possible to detect the clock source, if it is driven from combinational or sequential logic, a port, etc.
- *Registered objects analysis*, which detects whether a signal or variable of the code will be register with a flip-flop or a latch, before the synthesis process that normally takes too much time. The use of this checker with the one described above can guarantee that the registers are not duplicated in the design and that the VHDL adheres to the philosophy of synchronous design that enhances its quality and reusability.
- *Area estimation*, which provides an estimation of the complexity of the synthesized hardware, by an analysis of the VHDL code. The results of this area estimation do not try to be precise but to help the designer in the selection of two or more possible solutions. Therefore, the obtained results are useful for making comparisons and must be interpreted in a "relative" basis.

Fig. 6 shows the *graphical hierarchy display* of the CAN controller. This tool enables the possibility of analyzing the hierarchy of the circuit having information of the whole circuit or of any sub-block, such as inputs and outputs, number of code lines, etc. Besides, it is possible to trace the signals throughout the hierarchy, having information about their function as clock, initialization, data, etc. In this figure, it can also be observed the final block partition made in the design.

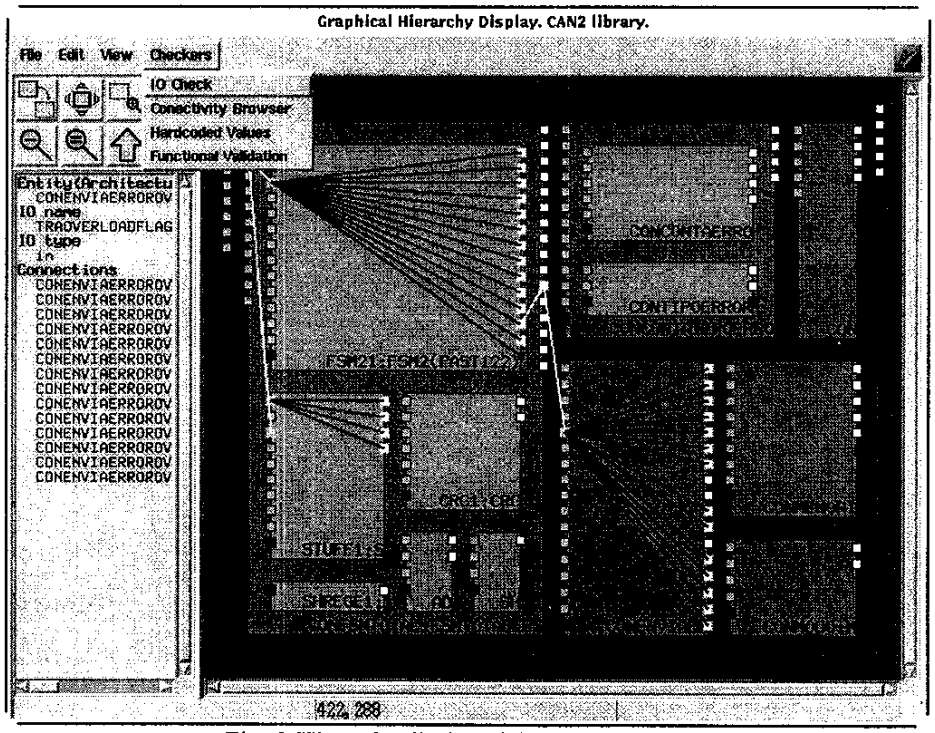


Fig. 6. Hierarchy display of the CAN controller

IV. EXPERIMENTAL RESULTS

Table 1 shows some data of the complexity of each block of the CAN controller, in terms of VHDL source code lines, number of inferred memory elements and equivalent gates after synthesis. The last column shows the complexity estimated by ARDID before synthesis. It has to be considered, that this tool works with non-elaborated VHDL, so all the information included in generics in the design is not considered in the area estimation tool. This is the main reason for the errors in the area estimation shown in the table. It is important to consider that ARDID area estimation only lies in a rough analysis of the VHDL code and no synthesis nor optimization process is run; this implies that the area estimation

made by ARDID is run in less than one minute, while the whole synthesis process takes hours.

The results shown in Table 1 have been obtained using *Synopsys Design Compiler* and the library *class*. The total number of VHDL code lines are not calculated as the addition of all the blocks but as a line count considering the structural descriptions, as well.

function	#VHDL lines	#gates (estimated)	#gates (real)
bit timing	165	718	302
bit stuffing	208	414	212
message proc.	724	5705	3368
transmission	412	1409	1098
error management	346	1144	1186
CRC	156	405	674
TOTAL CAN	–	9795	**6840**
Memory	220	1428	6087
Total with memory	–	11223	12927

Table 1. Synthesis results of the CAN controller

The resulting operating frequency is 30 MHz, for the main clock of the systems, which implies that the maximum transmission rate for this CAN controller is 5 Mbit/s, which is larger than required by the CAN standard. Besides, this operating frequency could be improved by using a more appropriate ASIC or FPGA technology.

The use of VHDL in combination with the method imposed by ARDID has demonstrated to have several advantages, both for the designers and the design manager.

First, during the design phase it has been possible to keep track of the "quality" of the VHDL code by using the version control system, where all the steps of the design have been stored. The version control system has been identified as an extremely useful tool not only in VHDL-based designs, but also in any project that involves source code development by a working team.

Second, the use of the quality checkers during the design phase has helped the designers in the code debugging in an easy and fast way. Any signal missing in a sensitivity list or any extra latch inferred in the code, for example, were easily detected during the coding process. The work of the manager has been also improved by the use of these tools as most of the issues of the design process were easily checked, reducing the need of visual inspection of the code, that is normally required in this type of projects.

Up to now, it is difficult to assert that ARDID-based methodology ensures an easy reusability of the developed VHDL code, but it has been demonstrated that reduces the risks of errors at the logic design stage, improving the re-targeting process on different implementation technologies.

V. CONCLUSIONS AND FUTURE WORK

The design of a virtual component for a CAN controller has been reported in this paper. The importance of CAN networks in current distributed control applications has driven the selection of this module and the possibility of its integration in different types of devices (ASICs, FPGAs). The implementation results on a generic library shows a complexity of less than 7000 equivalent gates, with a maximum transmission rate of 5 Mbit/sec, which complies with the CAN standard.

The importance of the design process to assure the quality of the design to be practically reused in different applications and by different design teams, has been demonstrated in this design experience. For this purpose, tools like ARDID are a need for designers, because just the use of HDLs and top-down methodologies can not guarantee the quality of designs.

We are now making the final tests to the CAN macrocell to integrate it in a FPGA that will be used in a CAN network controlled by a C167-CR CAN microcontroller.

VI. REFERENCES

[1] International Standard ISO 11898, *Road vehicles – Interchange of digital information – Controller area network (CAN) for high speed communication*, 1993.

[2] Wolfhard Lawrenz, *CAN System Engineering. From Theory to Practical Applications*, Springer-Verlag, 1997.

[3] M. Keating, P. Bricaud, *Reuse Methodology Manual*, Kluwer Academic Publishers, 1998.

[4] T. Riesgo, Y. Torroja, E. de la Torre, "Design Methodologies Based on Hardware Description Languages", *Trans. on Industrial Electronics*, vol. 46, pp. 3-12, February 1999.

[5] Y. Torroja, C. López, T. Riesgo, J. Uceda, "A Set of Tools to Help in the VHDL Design Flow of Complex Systems", *Proceedings of DCIS'97*, Sevilla, November 1997.

[6] LEDA, *VHDL Verilog System, Implementor's Guide*, v4.3.2, 1998.

[7] C. López, T. Riesgo, Y. Torroja, E. de la Torre, J. Uceda, "An Error Simulator to Estimate the Quality of Design Validation Experiments", *Proceedings of Forum on Design Languages (FDL'98)*, Lausanne (Switzerland), September 1998