FPGA Implementation Using Renoir Tools. Application For Bit Timing Logic (BTL) Synthesis of Controller Area Network With 100% Free Error.

Kamel ABOUDA, Jerôme DUCAUD, Herve HENRY, J. L. AUCOUTURIER

Laboratoire d'Etude de l'intégration des Composants et Systèmes Electroniques (IXL), UMR-CNRS 5818, Université Bordeaux I, 351 Cours de la Libération, F-33405 Talence Cedex, France.

abouda@ixl.u-bordeaux.fr

Abstract– **Logic Synthesis using VHDL simplifies considerably logic circuit design. However, when application requires few thousands of VHDL code, it is very benefit to use graphic software that can produce the VHDL code. But it is always necessary to master VHDL language. In this paper we have studied VHDL generated with three basic process of Renoir state machine. An application for designing, synthesizing, implementing a bit timing logic of an ISO normalized Controller Area Network (CAN) is given. It has been tested with 100% free error.**

I. Introduction

Renoir graphics capture tools includes graphical bloc diagram, state diagram, flow chart and truth table editors incorporating HDL generation and process management tools. The Renoir tools are closely integrated with HDL compilation, simulation and synthesis tools supporting both the VHDL and Verilog hardware description languages.
The bloc diagram which is the highest-level design description must be considered us a link between functional blocs. Inside a block diagram, we can perform state machines, one or more concurrent flow charts, truth tables and also bloc diagrams.
This paper interests particularly the VHDL generated code with both state machines and flowcharts. The synthesis validity of VHDL generated will be specially discussed. Finally, we will describe our contribution to implement a bit timing logic in Xlinx 4006 circuit using those software tools.

II. State diagrams philosophy

Renoir tools present the possibility to make Moore or Mealy state machines and also a combination of the two types. The VHDL generated by Renoir graphics tools is systematically composed by three concurrent process: clocked, next_state and output.

II.1 Clocked process [1]

The sensitivity list of the clocked process is the clock and reset when system is asynchronous. It contains waking conditions (active reset, clock edge, etc..), reset values, default assignment to internals, global internal actions, state actions for internal signals only and transition actions for internal signals only. We note that it is possible to assign internal at the beginning of the state with declaring it in transition actions. Therefore, internal can be assigned at the end of the state if it is declared in state actions. In the two cases, internal affectation is synchronous (figure.1).

II.2 Next_state process

The list of sensitivity of this process is the current_state, bloc inputs and internal signals. It contains only one CASE allowing the determination of the future state (next_state) depending on conditions declared in graph transitions. We note that the next_state process is completely asynchronous (figure.1).

II.3 The output process

This process has the same list of sensitivity as the next_state process. It contains default assignments, global actions, state actions and transition actions. We note that outputs are totally asynchronous (figure.1).

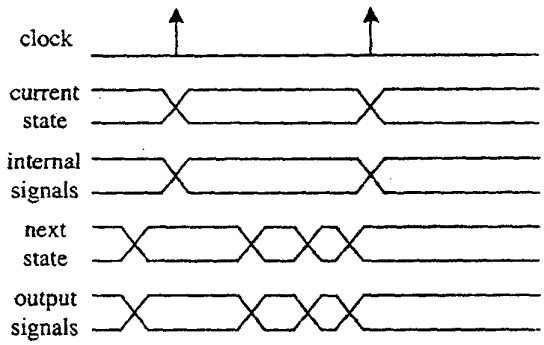

Fig.1. Concurrent process description

III. Contribution with Synthesizable VHDL generated Code Method with flowcharts.

The flow chart can create all types of process described above. It can also perform concurrent process in block architecture. Every flow chart has its proper sensitivity list and possibility to declare its own variables (signals and constants). However, it is indispensable to master very well VHDL language to design correct flow charts in aim to synthesize VHDL compilation. In this order, we present our contribution to avoid bad or no synthesizable VHDL code produced with flow charts even if it was done in aim to optimize logic gates of ASIC or FPGA associated. Testing Renoir tools with functions described with VHDL and rebuild with flow charts, has shown two essential properties that must be taken into account to generate correct and synthesizable VHDL code.

III.1 Cascaded decision boxes

The first rule we give, that where there is cascaded decision boxes (IF..THEN..ELSE), we must imperatively

closing the loop (END IF) of the flow chart diagram according to the decision level as it was shown in figure 2.

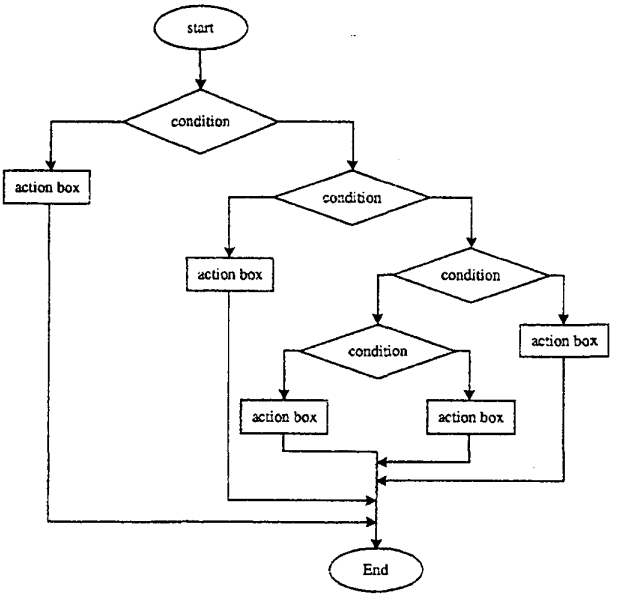

Fig.2. Cascaded decision boxes

III.2 Interaction between action boxes

The second rule we give that is isn't possible to try to optimize flow chart by merging branches of different decision boxes even if actions are similar. Figures 3 and 4 illustrate what we must do and what we must avoid to produce synthesizable VHDL using concurrent flow charts.

IV. Application for BTL Implementation on FPGA

IV.1 CAN Functional Description

The Controller Area Network Circuit is a large scale-integrated peripheral device [2, 3, 4] that executes the entire protocol of an automobile or industrial network. The block diagram of figure 5 describes it [4].

The Bit Stream Processor (BSP) controls the entire protocol, differentiates between the frame types and detects frame errors. The Error Management Logic (EML) receives error messages from the BSP and, in turn, sends back information about error state to the BSP and CPU Interface Logic CIL. The CIL controls the access of the host by way of a serial interface, interprets the commands and outputs status and interrupts information. The Bit timing Logic (BTL) determines the bits timing and synchronizes with the edges of the bit stream on the CAN bus. The Transceiver-control logic consists of the programmable output driver, input comparator and input multiplexer. Finally, the Clock Generator (CG) consists of an oscillator and a programmable frequency, dependent on the crystal clock, is available with the CLKOUT pin, e.g. for the clocking of a host controller.

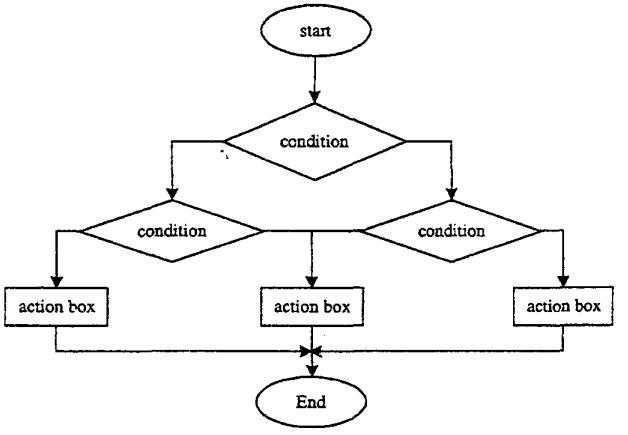

Fig.3. No synthesizable flow chart

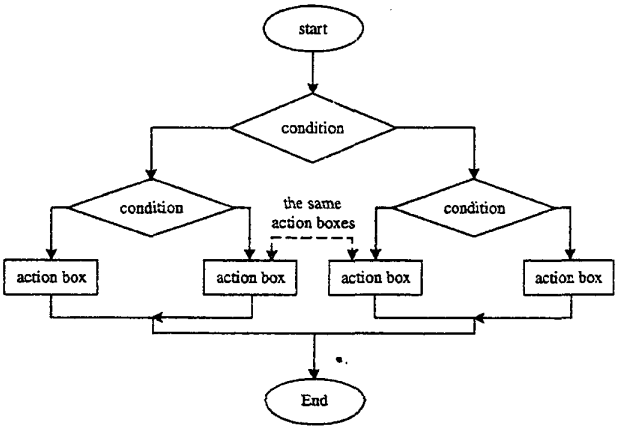

Fig.4. Synthesizable flow chart

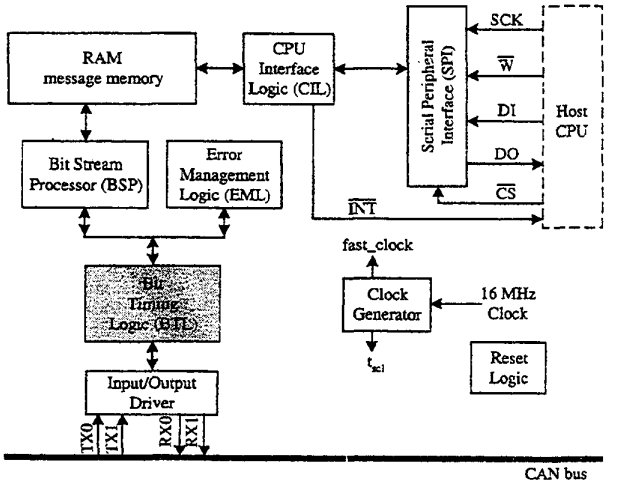

Fig.5. CAN block diagram.

IV.2 High level bloc diagram of The BTL

The bit timing logic samples logic level red on the CAN bus throughout the input interface. It reads the bus logic level at system clock period. The current byte value is taken at the sampling point corresponding to bus logic level at the end of segment 1. We have used the same BL1 and BL2 registers of the siemens CAN [4]. The high level description of the BTL is shown in figure 6.

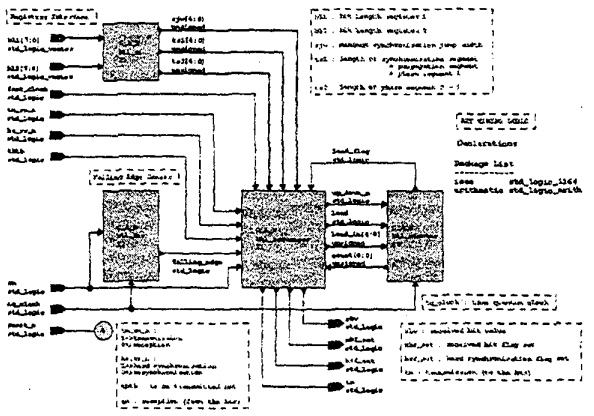

Fig. 6. Bit Timing Logic Bloc Diagram

IV.3 BTL Bloc functions

The btl registers interface, btl_ri bloc, shown in figure.7 computes the length of phase segment 1 (ts1), the length of phase segment 2 (ts2) and the synchronization jump width (SJW) which are used in BTL sequencer.

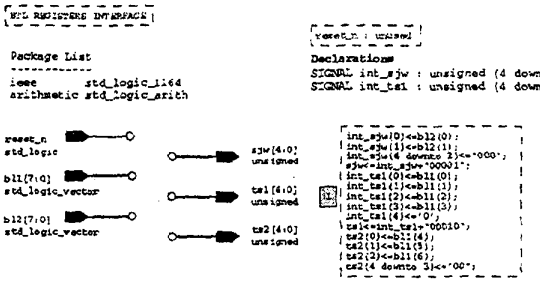

Fig.7. BTL registers interface

The btl_counter is illustrated by figure.8. It is a five bits counter / deductor with asynchronous reset and synchronous loading to be easily implemented in Xilinx circuit. The btl falling edge sensor: btl_fes bloc (figure.9), affects the falling edge byte to '1' when a falling edge is detected. The detection principal is described by figure 10. The btl-sequencer bloc, btl_sequencer shown in figure 11, is a state machine in which actions are simple and, sometimes, are directly described by VHDL language. All transitions between different states of synchronization are coded.

IV.4 Different BTL synchronizations

The edge of the input signal is expected during the synchronization segment, which is equal to one system clock cycle: tscl. The timing segment 1, determines the sampling point within a bit period. This point is always at the end of segment one. The segment is programmable from 1 to 16 tscl. Formula (1) gives the timing segment 1 length:

$$t_{Tseg1} = (TS1+1) * t_{scl}$$

Where TS1 is four bytes value in BL1 register [4] and the system clock period is calculated as follows:

$$t_{scl} = (BRP+1) * 2 * t_{osc}$$

The oscillator period t_{osc} is function of the crystal frequency:

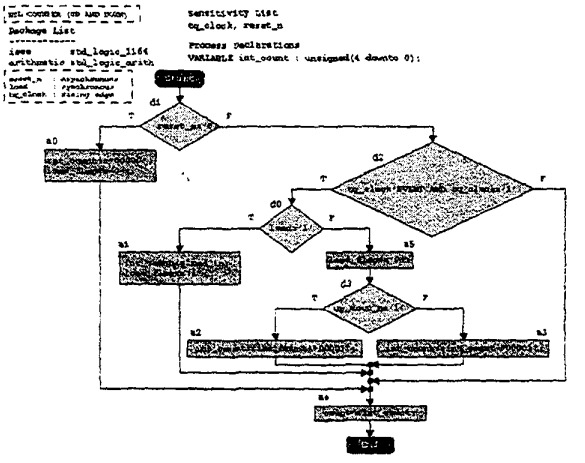

Fig.8. BTL counter.

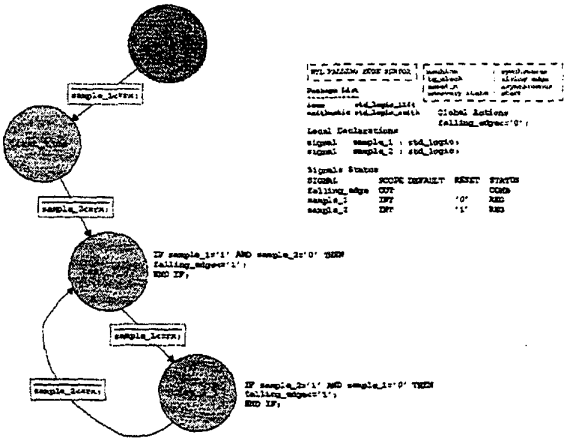

Fig.9. BTL falling edge sensor.

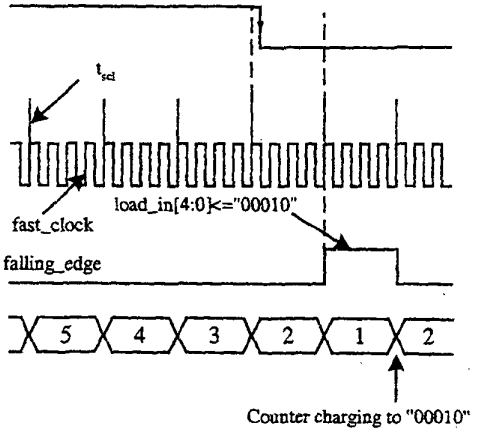

Fig.10. Falling edge detection principal.

$$t_{osc} = \frac{1}{f_{crystal}}$$

The timing segment 2, provides extra time for internal processing after the sampling point. The segment is programmable from 1 to 8 tscl. The length of this segment is given by the following formula:

$$t_{Tseg2}=(TS2+1)*t_{scl}$$

Where TS2 is three bytes value in BL1 register.

To compensate for phase shifts between the oscillator frequencies of the different bus stations, each CAN controller must be able to synchronize to the relevant signal edge of the incoming signal. The synchronization jump width, SJW, determines the maximum number of system clock pulses by which the bit period can be lengthened or shortened for resynchronization. The synchronization jump width is programmable from 1 to 4 tscl. The length of this segment is determined by this equation:

$$t_{SJW}=(SJW+1)*t_{scl}$$

Where SJW is two bytes value in BL2 register [4].

The different steps used in BTL synchronizations are shown in figure 11 and resumed as follows:

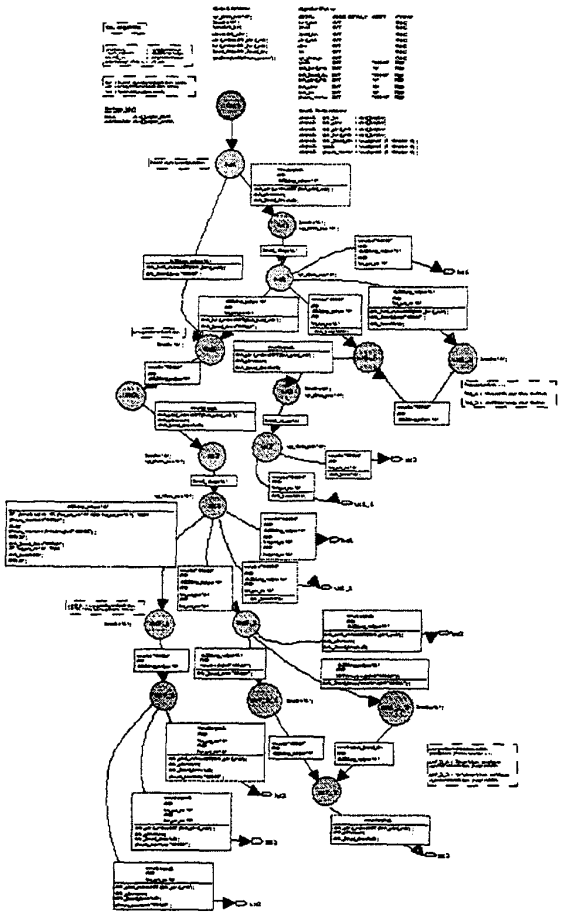

Fig.11. BTL sequencer

- Byte emission of starting frame: When a connected circuit to the CAN bus (node) is able to emit but haven't detect a falling_edge and the bit timing period is enhanced, it can emit starting frame byte without any hard synchronization.
- Passage from segment 1 to segment 2: When counter attain a value equal to the sum of ts1 and the phase-error, the end of segment 1 is attained and a sampling of bus level is effectuated on falling edge of the fast_clock. At this moment, the deductor is initiated with loading ts2 on the relevant signal edge of the following tscl.
- Passage from segment 2 to segment 1 without phase error: If there isn't any falling edge detected, segment 2 is ended when deductor attains '00000', counting is then started.
- Hard synchronization of receptor node: If a falling edge is detected before having finished deduction, the falling edge byte is forced to '1' during all tscl period, which follows the edge. Counter is loaded with '00010' only after resetting the falling edge. At this moment, the deductor is started.
- Hard synchronization of emitter node with the foremost node starting emission: When a node which is able to be emitter decide to emit dominant byte and a falling edge is detected before the end of the segment 2, it will becomes emitter but must be hard synchronized with the fastest node. Ididentifactors and RTR arbitrating will dissociates them.
- Resynchronization with SJW = 3 and positive phase error > SJW: When phase error is positive, segment 1 of the byte must be lengthened. If a falling edge is detected in segment 1 therefore counter is charged with value = count - SJW + '00010' and counting begins with this value. The phase variable error is equal to zero and segment 2 begins at ts2 value.
- Resynchronization with SJW = 3 and negative error phase > SJW: When phase error is negative and less than SJW, only segment 2 is shortened but segment 1 will be little lengthened. The end of the segment 1 is obtained when counter attain value = ts1 + phase error.
- Emission request during resynchronization phase: for example the emission of an acknowledgment slot byte by a receptor node which is, at list, slower than another node
- From resynchronization mode to hard resynchronization: for example reception of frame starting byte after bus idle or after intermission field.

V. Conclusion

For complicated applications, using graphic tools is very important and avoids writing a few thousands of VHDL code lines. In this application, the total VHDL code lines generated with BTL, BSP and EML are higher than 3000 lines. This don't prevent that subtleties like those discussed with flow chart description tools, increase logic gates number and consequently the size of the integrated circuit associated. We conclude that the interaction between flow charts and VHDL code optimization is a missing tool to complete performances of this designing graphic software. In this application, the solution was consisting to try to optimize VHDL code after its generation. Finally, we note that with Renoir, we have designed, synthesized and implemented in Xilinx 4006 a bit timing logic of controller area network with 100 % free error.

REFERENCES

[1] Mentor graphics, " Renoir Tutorial ", Software version 3.0, March, 1998.

[2] Dominique PARET, "Le CAN, Son Protocôle et ses particularités", le bus CAN Controller Area Network, Editions DUNOD 1996.

[3] Philips, " CAN Protocol Description ", 83C592, 80C592, 87C592 Target Device Specification, version 2.3, 1992.

[4] Siemens data sheet, " Standalone full-CAN Controller SAE 81C90/91 ", Microcomputer Components, January, 1997.