CAN Error Injection, a Simple but Versatile Approach

Hauke Webermann, esd electronic system design gmbh
Andreas Block, esd electronic system design gmbh

Nowadays, CAN buses are standard building blocks, not only in automotive area and industrial automation, but to an increasing degree in safety sensitive areas, including medical environments, aircraft industry and even in space. With the elevated safety requirements there's a rising need for verification, simulation and testing. In general, CAN controllers available on the market are unable to generate CAN traffic containing errors or violating CAN ISO 11898 standard. This paper describes a simple and effective approach using flexible FPGA technology to inject errors into CAN buses. Adding rather small error injection units to a CAN controller within an FPGA provides means to not only generate all kinds of errors on CAN bus, but also to interact with and modify ongoing CAN traffic, at the cost of little more than standard CAN hardware. Error injection units feature several injection modes, such as CAN arbitration, time triggered or pattern matching, and can be combined to accommodate more complex scenarios.

Introduction

In this paper, the possibility for an easy to implement and resource saving solution for injecting reproducible errors into CAN buses will be discussed.
The error injection creates the potential to simulate, test and analyze the behavior of existing systems. Additionally, there is the possibility to expand the range of residual bus simulation with defective CAN messages.
As peculiarity, it uses the standard CAN hardware and no special CAN physics. So the error injection can be implemented on almost all FPGA based CAN boards. This has the advantage of testing existing systems with little effort by the error injection and without any special test equipment.
The error injection is a VHDL module and is an extension to a CAN IP core on an FPGA based CAN controller board. Via several trigger modes it is possible to generate all kinds of errors on CAN networks. This implies the possibility to test existing systems with reproducible CAN errors. Depending on the configuration, a CAN controller can have several units, so it is possible to create complex error scenarios by combining and cascading these units.
The error injection module is divided into several error injection units. These units can be assigned to the different CAN controllers in the FPGA. Figure 1 shows a simplified overview of the FPGA structure. The error injection is an additional module to a CAN IP core. It works parallel to the normal function of the CAN controller.

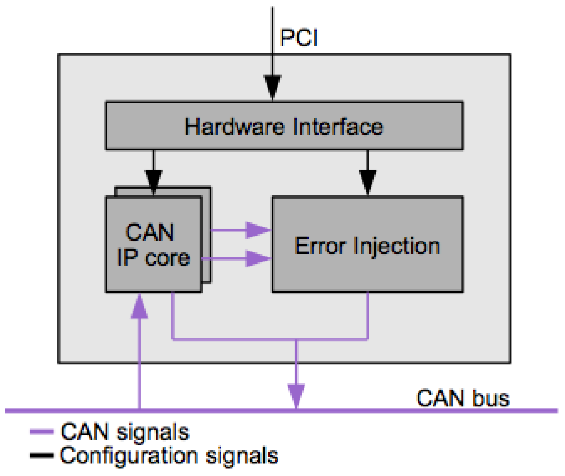

Figure 1: Overview CAN controller board with error injection

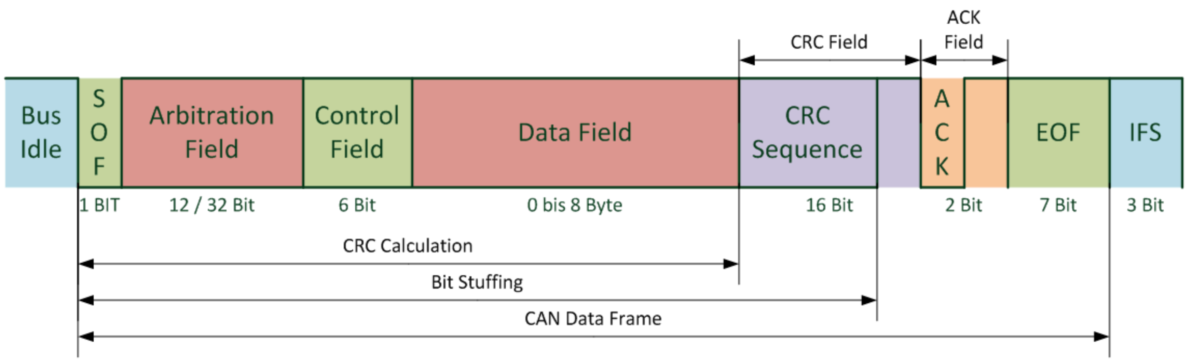

Figure 2: CAN frame overview (SOF: Start of Frame; ACK: Acknowledge; CRC: Cyclic Redundancy Check; EOF: End of Frame; IFS: Inter Frame Space)

CAN Protocol and Error Handling

At this point the relevant properties of the CAN protocol, which are necessary for error injection will be discussed shortly. The CAN protocol works on the wired AND principle. A recessive level only occurs on the CAN bus, if no tranceiver is actively driving the bus. The bus is dominant when at least one tranceiver is sending. A dominant level always overrides a recessive level.
This behavior is used among others for the propagation of errors. A detected error is indicated to all other nodes by an error frame, and the transmitted CAN frame will be destroyed. Figure 3 shows an example of an active error frame. An error frame is divided into two parts. The error flag consists of six bits. This sequence should not occur and violates the bit-stuffing rule, since only five identical bits are allowed consecutively. This way all the other nodes are notified of a detected error. The error flag can increase by up to six bits, because other nodes may notify an error by a received error flag and so they send an additionally error frame. The error frame is completed with an error delimitter of eight recessive bits. Depending on the controller state an active or a passive error frame is sent. A passive error flag consists of six recessive bits.

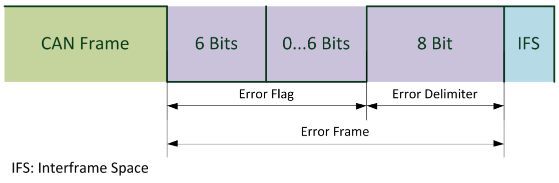

Figure 3: Active error frame

- Bit error
    - Form error
    - Stuff error
    - CRC error
- Acknowledge error

The bit error can only be detected by a sending node. Each node reads back the actual transmitted bit. A bit error occurs if a different bit is received from the CAN bus with the exception of the arbitration phase (see Figure 2). A form error occurs if one or several dominant bits in the fixed, predetermined segments are detected (such as the cyclic redundancy check (CRC) delimiter). The stuff error occurs when no inverse bit is received after a series of five equal bits on the bus. If the calculated checksum does not match the received CRC, the acknowledge (ACK) will not be set and beginning with the acknowledge delimiter an error frame propagates the CRC error. An acknowledge error occurs if no node has received the frame correctly and thus no

node overrides the recessive acknowledge bit.
The CAN bus is encoded in Non-Return-to-Zero (NRZ), so the information is saved in the level of the CAN signal. The CAN bit stream is sampled at a fixed time. This sample point depends on the baud rate, the clock rate of the used controller and the configuration of the "Baud Rate Timing Register" (BTR).

Concept

In ISO 11898 [1, 2] the CAN protocol is only defined in OSI layers 1 and 2. An error injection principally can be done in both layers. As the error injection is supposed to work on standard CAN hardware with standard CAN transceiver, it only works on the Data-Link-Layer
For the error injection the CAN bus is reduced to a bit stream with the signals:

- baud rate
- sample point
- sampled bit

In order to implement this as resource efficiently as possible, no signals should be generated twice. Many signals are preset by the CAN IP core. So these signals and internal states can be used by the error injection, too.
As already mentioned the error injection uses standard CAN hardware, thus only dominant bits can be injected, as dominant bits cannot be overridden by recessive bits. Through this characteristic all types of errors can be generated except for the acknowledge error.
In order to implement these characteristics, the preferred solution is a bit stream injection. For this purpose some more internal signals out of the CAN IP core will be needed. Amongst others, a signal "Point of transmission" is required for the correct beginning and length of a bit. Thus, created a small and simple transmit automat, which sends an user defined bit stream to the CAN bus via a shift register. The bit stream is sent without CRC calculation or bit stuffing. There is no CAN bus feedback, so the transmission will not be terminated, if CAN error frames are encountered. This unit is called "CAN TX".

To activate the CAN TX module there are some trigger modules defined:

- Trigger Pattern Match
- Arbitration
- Trigger Timestamp
- Trigger Field Position
- Trigger External Input

These modules are simple and flexible in handling. Only one trigger unit can be activated at a time.
A complete error injection unit is shown in Figure 4 and consists of the sending module described (CAN TX) and several Trigger Units.

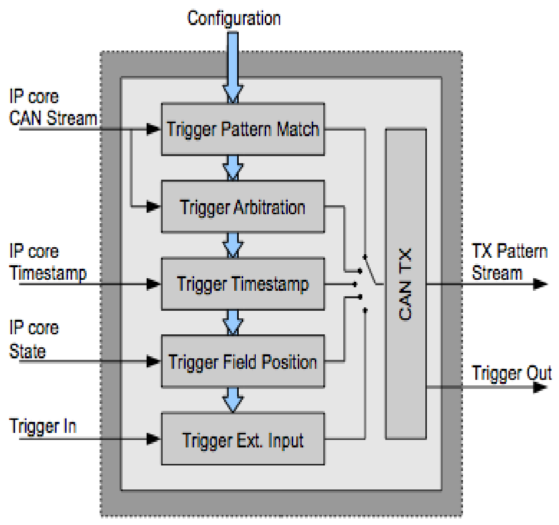

Figure 4: Error injection unit

The **Trigger Pattern Matching** module can search for a user defined bit stream. If the bit stream matches the sampled CAN bit stream, the CAN TX Module is triggered. Additionally, a bit mask defines the region which is to be compared. It can be chosen between the direct CAN bit stream and the destuffed bit stream.
The **Trigger Arbitration** sends a bit stream via the CAN TX module under the rules of arbitration. It is possible to send correct CAN frames or CAN frames with errors. This provides for example the means to send CAN frames with CRC errors otherwise impossible with standard CAN controllers.
With **Trigger Timestamp** the CAN TX module is triggered by expired timer.

The reference implementation CAN board has an internal 64 bit timestamp and this is compared with the user defined timestamp. When the time elapses, the CAN TX module is triggered.
The **Trigger Field Position** module triggers at a specific position in a CAN frame. For example, it can be set to trigger within data length code or on CRC delimiter. In the reference implementation an error code of a detected error can be used to configure this trigger module to repeat or simulate a CAN error
The module **Trigger External Input** triggers on an external signal. This may also be used to combine trigger sources or to trigger on an event provided by another error injection unit on another CAN bus. Each error injection unit has a trigger out signal which can be an external trigger source for another trigger module. Via a bit mask it can be chosen which trigger unit is the trigger source. Additionally, it is possible to trigger on an external IO pin.
Supplementary to these trigger modules there are some global options. It is possible to delay the enable signal for the CAN TX module. For this purpose, a delay time can be set in bit times. A second option is the repeat flag. With these, the trigger module is enabled again after the CAN TX module has sent the bit stream. There is an additional register (arm delay) to insert a delay in bit times between each repetition. For example this can be used to achieve an exactly defined bus load in combination with the Trigger Arbitration.
In the error injection module there are multiple injection units, which can be configured independently. Each unit can be assigned to one of the available CAN networks. The configuration and assignment of the error injection units are available from a global register file. The number of possible units depends on the size of the FPGA. In standard case there are four independent error injection units implemented which can be cascaded.
The output of the CAN IP core and the associated error injection unit(s) are combined via an AND logic.The error injection is connected with the CAN data stream and behaves passively in idle. It works in parallel to the CAN IP core and thus the normal CAN communication is completely unaffected.

The configuration of this error injection module can be done directly via the registers or by an operating system independent application programming interface (API).

Use Cases

The error injection can be help to improve the overall robustness in all areas where CAN is used. This may be in the aerospace, automotive, medical or general industrial automation. In the following three common examples are discussed in more detail. In all examples an FPGA based CAN board with error injection is connected to a system with four external sensors by a CAN bus.

Use Case I: Defective Sensor

The first scenario is an error frame injection on the sensor 2. An error injection unit is configured in Pattern Matching Mode and the Trigger Pattern is set to the CAN ID of data from sensor 2. The TX pattern (the bit stream which is sent by CAN TX module when triggered) is set to an Error Frame of six dominant bits.

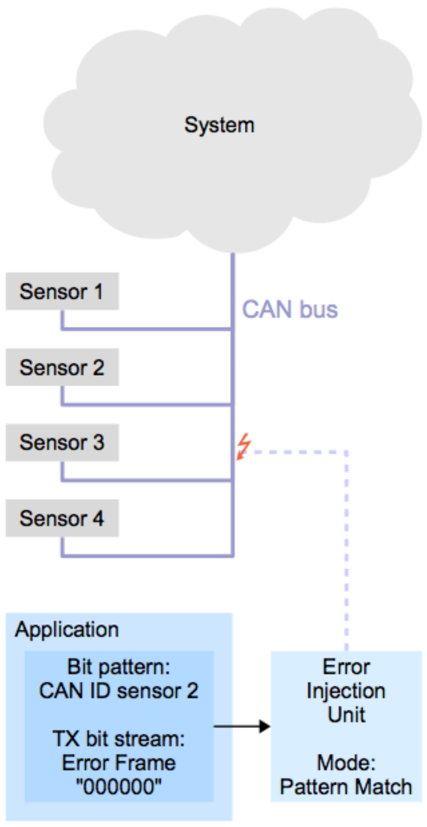

Figure 5: Use case scenario 1: defective sensor

Figure 5 shows the complete scenario. Sensor 2 maybe a temperature or an acceleration sensor which sends his value at any given rate. The activated error injection unit compares the current bit stream with the defined sensor CAN ID. As soon as the Trigger Pattern Match module detects the CAN ID from sensor 2 in the current CAN bit stream, the stored error frame is sent and the sent CAN frame from sensor 2 is destroyed.

Use Case II: Babbling Idiot

The second scenario is the so called “Babbling Idiot” or “ID Pollution” and simulates a defective sensor which sends continuously with a fixed CAN ID. An error injection unit is configured in Trigger Arbitration Mode and the repeat flag is set but without any delay time. The TX pattern is set to a complete CAN Frame with the CAN ID 0x100 for example.

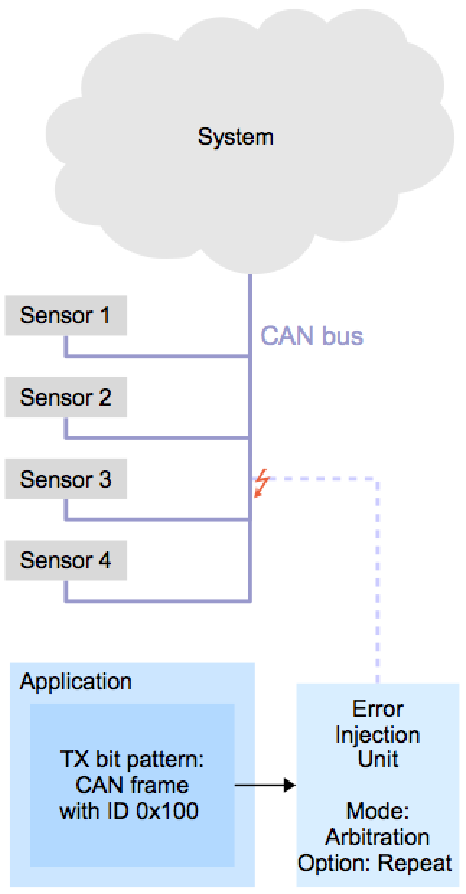

Figure 6: Use case scenario 2: "Babbling Idiot", "ID Pollution"

Figure 6 shows the complete scenario. The activated error injection unit sends continuously CAN frames with ID 0x100 back to back, so any CAN frame with an ID greater 0x100 will loose the arbitration. Thus the user can check whether vital CAN frames (e.g. emergency shut down commands like CANopen NMT messages) can be sent correctly and whether the system remains reactive.

Use Case III: Residual Bus Simulation

In the third scenario the system is tested by a residual bus simulation. The CAN frames of sensor 3 and sensor 4 are simulated by the CAN IP core. The data of sensor 2 is sent by an error injection unit. This unit is configured in the Trigger Arbitration mode and sends a CAN frame with a stuff error for example.

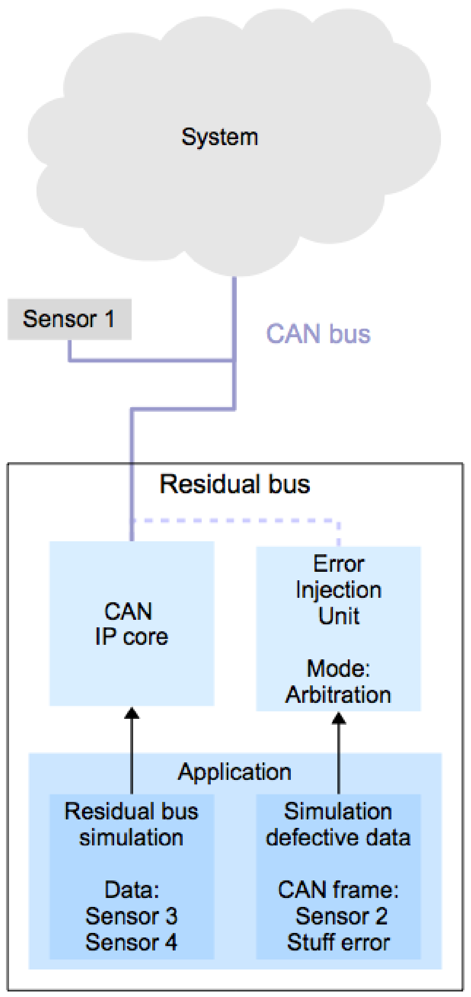

Figure 7: Use case scenario 3: residual bus simulation

Figure 7 shows the complete scenario. The activated error injection unit sends a corrupted CAN frame in Trigger Arbitration mode, so the user can check if all CAN nodes identify the error correctly.

Summary

The error injection provides the means to inject a user defined bit stream into a connected CAN bus. Due to the generally applicable design, the error injection is very versatile. It can be used on one hand for complex test scenarios and on the other hand for residual bus simulation.
The Error Injection is an extension of a CAN IP core and works with all CAN IP cores, which provide the aforementioned internal states and signals.
The outputs of error injection and CAN IP core are combined to one output signal within the FPGA. So, no change in the CAN network cable connection is necessary.

References:

[1] ISO 11898-1, December 2003: Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signaling.
[2] ISO 11898-2, December 2003: Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit.
[3] esd electronic system design gmbh (2007): CAN-API. Part 1: Function Description. (26. July 2007).
[4] esd electronic system design gmbh (2011): CPCI-CAN/400. 4x CAN with ARINC Protocol and IRIG-B. (27. April 2011).
[5] Etschberger, Konrad (2002): Controller-Area-Network. Grundlagen, Protokolle, Bausteine, Anwendungen. München: Hanser.
[6] Philips Semiconductors (2000): SJA1000. Stand-alone CAN controller. (4. February 2011).
[7] Voss, Wilfried (2005): A comprehensible guide to Controller area network: Copperhill Technologies Corporation.
[8] Webermann, Hauke (2011): Entwurf, Implementierung und Test einer Funktionseinheit zur Injektion von Fehlern in CAN-Busse.

Hauke Webermann
esd electronic system design gmbh
Vahrenwalder Str. 207
D-30165 Hannover
+49-511-37298-0
+49-511-37298-68
hauke.webermann@esd.eu
http://www.esd.eu

Andreas Block
esd electronic system design gmbh
Vahrenwalder Str. 207
D-30165 Hannover
+49-511-37298-0
+49-511-37298-68
andreas.block@esd.eu
http://www.esd.eu