APPLICATION NOTE

How to Handle Data Overrun Conditions of the 82C200, 8xC592 and 8xCE598

AN95092

Philips
Semiconductors

PHILIPS

How to Handle Data Overrun Conditions of the 82C200, 8xC592 and 8xCE598

Application Note AN95092

Abstract

Data overrun conditions may occur in CAN bus systems, if received messages cannot be read fast enough from the receive buffer by a CPU. Normally the controller software for a CAN bus system should be designed in such a way, that the receive buffer is serviced without getting an overrun condition.

For the seldom situation, where a data overrun occurs and must be serviced, this application note gives a recommendation on how to handle it.

APPLICATION NOTE

How to Handle Data Overrun Conditions of the 82C200, 8xC592 and 8xCE598

AN95092

Author(s):

Egon Jöhnk
Product Concept & Application Laboratory Hamburg, Germany

Keywords

Automotive
Industrial
CAN
Data Overrun
82C200
8xC592
8xCE598

Date: 11th August, 1995

Summary

This report is intended to provide application support for the seldom situation, where a data overrun occurs and must be serviced.

The report describes

- when a data overrun may occur
- how it is signalled to the CPU
- which problems arise from getting a data overrun
- how the controller software should react in order to recover from a data overrun.

The recommendation of how to handle a data overrun is given as structograms, which may be transferred easily to different implementations of a CAN controller software.

CONTENTS

(page has been left blank intentionally)

1. INTRODUCTION

The CAN-controller 82C200 and the on-board CAN-controllers of the 8xC592 and 8xCE598 contain twin receive buffers. They are each 10 byte memories and are used alternatively to store messages received via the CAN-bus. The CPU of the microcontroller can process one message while another is being received. One buffer is assigned to the CPU for reading it, while the other buffer is assigned to the CAN controller for storing a new message, provided that the buffer is empty. After a new message has been stored the buffer assignments are exchanged giving the CPU access to this message and allowing the storage of a new message in the empty buffer. After having transferred the new message the CPU has to release the buffer (‘Release Receive Buffer‘ command), which declares it as “empty”. If the second buffer contains a message at this time the assignments are exchanged again.

If both receive buffers are “full” and the CPU could not release one of them in time before a third message is being received, the CAN-controller cannot store the new message and signals a data overrun. This should be avoided by design in a safe system because messages are lost due to a data overrun.

2. DESCRIPTION OF ‘DATA OVERRUN‘

A data overrun is signalled by the overrun status flag and the overrun interrupt flag. They are set after the CAN-controller did receive the first byte (upper part of the identifier) of a message, if no empty receive buffer is available, which may happen, if the buffers were not released after having received two messages already. Also during a transmission the CAN-controller needs to store the first byte in a receive buffer, because it might lose the arbitration during the second byte and become a receiver (only valid, if the transmitted message passes the acceptance test).

Running into a data overrun situation states, that the CPU is extremely overloaded and that data is lost, possibly causing inconsistencies in a system. Normally a system should be designed in such a way, that the received data is transferred and processed so fast, that a data overrun will not occur. An exception handler should be implemented into the controller software for capturing unforeseen situations. This handler should be the only one, which has to consider a data overrun. Data available in the receive buffer after a data overrun has been signalled is no more reliable, as one buffer may contain scrambled data from two messages. This data must not be used for further processing.

3. RECOMMENDED REACTION ON A ‘DATA OVERRUN‘ EXCEPTION

In case a data overrun occurs the exception handler should re-establish a correct communication on the CAN-bus as fast as appropriate for the system in order not to lose more messages. The fastest and most appropriate method is to execute a reset of the CAN-controller. The following structograms illustrate this method for the software solutions, where received messages are fetched either by polling or during an interrupt initiated by the CAN controller.

3.1 Interrupt Solution

The solution for an Overrun Exception Handler as shown in Fig. 1 and Fig. 2 is recommended in a system, where the received CAN messages are transferred during a CAN interrupt.

ASSUMPTIONS:	Both data overrun and transferring received data is handled by interrupt, so at least the Data Overrun and Receive Interrupt must be enabled.
ATTENTION:	During the exception handling any active and requested transmission will be stopped immediately due to the reset of the CAN controller. It depends on the overall system strategy, if it is accepted that this message will not be transmitted, or if a re-transmission must be initiated after the reset of the CAN controller. Therefore no general handling can be given for this situation.

```
+-------------------------------------------------------------------------------+
|main()                                                                         |
|  +----------------------------------------------------------------------------+
|  |Intialise_CAN()                                                             |
|  |  +-------------------------------------------------------------------------+
|  |  |/* among other things: */                                                |
|  |  |.....                                                                    |
|  |  |Enable Overrun Interrupt (control register of the CAN controller).       |
|  |  |Set CAN Interrupt to HIGH Priority                                       |
|  |  |           (Interrupt priority registers of the micro controller).       |
|  |  |Enable CAN Interrupt and set Global Enable                               |
|  |  |             (Interrupt enable registers of the micro controller).       |
|  |  |.....                                                                    |
|  |  +-------------------------------------------------------------------------+
|  +----------------------------------------------------------------------------+
|  |/* Other parts of the main program */                                       |
|  |.....                                                                       |
|  +----------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
```

Fig. 1 Overrun Handling and Data Transfer Done by Interrupt (main program)

```
+-------------------------------------------------------------------------------+
|CAN_Interrupt()                                                                |
|  +----------------------------------------------------------------------------+
|  |Save registers (if necessary)                                               |
|  +----------------------------------------------------------------------------+
|  |Read / Clear  the Interrupt register of the CAN controller                  |
|  +----------------------------------------------------------------------------+
|  |if (‘Overrun Interrupt’ = “set”)                                            |
|  +--------------then---------------+------------------else--------------------+
|  |”Overrun Exception Handling”     |if (‘Receive Interrupt’ = “set”)          |
|  |  +------------------------------+--------------then---------------+--else--+
|  |  |Set control bit               |”Read Receive Buffer”            |        |
|  |  |  ‘RESET REQUEST’ = “present” |  +------------------------------+        |
|  |  +------------------------------+  |Transfer Data by DMA *)      |        |
|  |  |Wait 2 clock cycles of the    |  |or other methods             |        |
|  |  |CAN controller  **)           |  +------------------------------+        |
|  |  +------------------------------+---------------------------------+        |
|  |  |Set control bit           **) |Set command bit                  |        |
|  |  |  ‘RESET REQUEST’ = “absent”  |’RELEASE RECEIVE BUFFER’ =       |        |
|  |  +------------------------------+”released”    **)                |        |
|  +---------------------------------+---------------------------------+--------+
|  |”Other Interrupts”                                                          |
|  |  +-------------------------------------------------------------------------+
|  |  |/* Serve other CAN Interrupts (if enabled).  */                          |
|  |  +-------------------------------------------------------------------------+
|  +----------------------------------------------------------------------------+
|  |Restore registers (if necessary)                                            |
|  +----------------------------------------------------------------------------+
|  |Return from Interrupt                                                       |
|  +----------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
```

*) DMA: Direct Memory Access, available on the 8xC592 and 8xCE598

**) Command requests may be executed with a max. delay of one clock cycle (1*t_{SCL}) of the CAN controller

Fig. 2 Overrun Handling and Data Transfer Done by Interrupt (Interrupt Handling)

3.2 Polling Solution

The solution for an Overrun Exception Handler as shown in Fig. 3 is recommended in a system, where the received CAN messages are transferred by polling the ‘Receive Buffer Status‘ flag of the CAN Status Register. Before actually checking for a new message the overrun status flag is checked. In case of a data overrun condition the CAN controller is reset in order to re-establish a bus communication in a short time.

```
+------------------------------------------------------------------------------+
|main()                                                                        |
|  +---------------------------------------------------------------------------+
|  |Intialise()                                                                |
|  |  +------------------------------------------------------------------------+
|  |  |/* Initialise the CPU and the CAN controller */                         |
|  |  +------------------------------------------------------------------------+
|  +---------------------------------------------------------------------------+
|  |while (FOREVER)                                                            |
|  |  +------------------------------------------------------------------------+
|  |  |Get current content of the CAN Status Register                          |
|  |  +------------------------------------------------------------------------+
|  |  |if (‘Data Overrun’ = “overrun”)                                         |
|  |  +-------------then--------------+------------------else------------------+
|  |  |”Overrun Exception Handling”   |if (‘Receive Buffer Status’ = “full”)   |
|  |  |  +----------------------------+-------------then--------------+--else--+
|  |  |  |Set control bit             | “Read Receive Buffer”         |        |
|  |  |  |’RESET REQUEST’ = “present” |  +----------------------------+        |
|  |  |  +----------------------------+  | Transfer Data by DMA *)    |        |
|  |  |  |Wait 2 clock cycles of the  |  | or other methods           |        |
|  |  |  |CAN controller  **)         |  +----------------------------+        |
|  |  |  +----------------------------+-------------------------------+        |
|  |  |  |Set control bit         **) | Set command bit               |        |
|  |  |  |’RESET REQUEST’ = “absent”  | ‘RELEASE RECEIVE BUFFER’ =    |        |
|  |  |  +----------------------------+ “released”    **)             |        |
|  |  +-------------------------------+-------------------------------+--------+
|  |  |/* Further reactions on the content of the Status Register              |
|  |  |   (if necessary) and other parts of the main program */                |
|  |  |.....                                                                   |
|  +--+------------------------------------------------------------------------+
+------------------------------------------------------------------------------+
*)  DMA: Direct Memory Access, available on the 8xC592 and 8xCE598.
**) Command requests may be executed with a max. delay of one clock cycle
    (1*tSCL) of the CAN controller
```

Fig. 3 Data Transfer Done by Polling (main program)

ASSUMPTIONS: All CAN interrupts are disabled. The communication between the CPU and the CAN controller is handled by polling only.

ATTENTION: During the exception handling any active and requested transmission will be stopped immediately due to the reset of the CAN controller. It depends on the overall system strategy, if it is accepted that this message will not be transmitted, or if a re-transmission must be initiated after the reset of the CAN controller. Therefore no general handling can be given for this situation.

4. CONCLUSION

Normally the controller software of a CAN bus system is designed in such a way that no data overrun occurs, because this would mean losing messages. Nevertheless it is recommended to consider the case of data overrun by implementing software, handling the data overrun exception.

During the recommended handling of the data overrun exception for the CAN controllers 82C200, 8xC592 and 8xCE598 messages available in the receive buffers are discarded. These are lost in excess of the data, which are lost due to the data overrun condition anyhow. This is done in favour of spending a minimum of time on detecting of and on reacting on the data overrun exception and on re-establishing a proper CAN communication.