AN1077
APPLICATION NOTE

OVERVIEW OF ENHANCED CAN CONTROLLERS FOR ST7 AND ST9 MCUS

by Microcontroller Division Applications

ABSTRACT

Automotive body network requirements have changed significantly in the last few years due to the introduction of OSEK and the increasing number of ECUs.

8-bit microcontrollers are and will stay widely used in a majority of automotive body applications. While a wide variety of powerful CAN controllers are available on the market for 16- and 32-bit microcontrollers, 8-bit microcontrollers still need too much CPU time for CAN communication management. This application note presents a new generation of CAN controllers optimized for 8-bit microcontrollers and designed to meet the needs of body applications.

1 INTRODUCTION

1.1 EVOLUTION OF AUTOMOTIVE NETWORKS

For several years CAN has been the world wide standard automotive communication protocol. Well established in power train applications in its high speed version, CAN is now used to connect a high number of ECUs in each car body. This body network - known as low speed CAN - brought major changes with its introduction.

Standardized Higher Layer

Today automotive manufacturers and their suppliers are going one step further towards standardization, integrating the OSEK software modules on each ECU.

Hierarchical architectures

To master the complexity caused by the increasing number of ECUs in car bodies, several networks and sub-networks are required. This limits the number of nodes connected to one network, but requires more nodes with a simple gateway function.

Low-end communication protocol

Local sub-networks - e.g. within one door module - does not need the full multi- master capabilities, speed and robustness of CAN. For this purpose the LIN communication protocol has been introduced as a low-cost solution.

1.2 IMPACT ON CAN TRAFFIC

These changes have an impact in terms of CAN message traffic.

Application Messages

Even if the information transmitted by each ECU has not increased significantly, the growing number of ECUs in body applications has led to a drastic increase in CAN traffic. Sometimes the same ECU may be used in several locations in the car - for instance for the right and for the left door module – so a node must be able to handle more message identifiers than the application itself would really require. A similar effect occurs when the same ECU is planned to be implemented in different car models or versions, this means the number of identifiers the node has to handle will increase.

Network Management Messages

In addition to the application messages, management messages like OSEK Network Management are needed to configure the network at start-up time, to control the bus when the nodes enter and exit sleep mode, to monitor the node while running and to handle bus-off conditions. This distributed NM requires the implementation of the OSEK-NM software on each ECU, which has to monitor the NM messages of all the other nodes. This NM is based on the token ring method and each node has its own address. The source and destination addresses are coded in the identifier of the CAN message. This means that there are as many identifiers needed for the NM as nodes in the network.

Diagnosis Messages For diagnosis purposes each ECU must be accessible to external diagnosis tools. According to ISO 15765-1 and 2 these diagnosis services are now implemented via CAN. These services require their own CAN messages.

1.3 IMPACT ON CAN CONTROLLERS

Just as the introduction of networks has modified automotive application implementations in a major way, modern network architectures have also changed the requirements imposed on CAN controllers. As mentioned earlier, the resource requirements have increased especially on the receiver side:

- A huge number of identifiers on the bus
- A high number of identifiers to be handled
- Various types of messages

All Quiet in the 8-bit CAN World?

While many CAN controllers for 16- and 32-bit microcontrollers have been recently developed in order to satisfy these new requirements, most of the CAN controllers implemented on 8-bit microcontrollers do not provide the required hardware to support these new functions effi-

ciently. Therefore software solutions are usually implemented, consuming CPU resources and making the application less independent from the CAN bus traffic.

The Overall Tasks of the CAN Interface

Considering the receiver part of the CAN interface the following tasks have to be performed:

- Message checking, error handling and acknowledgement. This task is fully handled by the CAN protocol implemented in all CAN controllers
- Message filtering indicates receptions addressed to the application and discards the other ones. This task is partially done by hardware but software filtering is often needed. The CPU resources consumed for software filtering depends on the CAN bus traffic. This means that the ECU performance is influenced by events not related to the application. This can make the modification of an entire system or the integration of the ECU in another network very critical.
- Identifier recognition. Once a message has passed through the filters its content must be identified in order to compute the destination address the data must be stored to.
- Signal extraction. To optimize the bandwidth usage of the CAN bus, the signals are concatenated to have more data transmitted in each message. Thus on message reception the application must extract the signals it is interested in.
- Signal storage in RAM

2 CAN CONTROLLER SOLUTIONS

2.1 TRADE-OFF BETWEEN COSTS AND EFFICIENCY

Nowadays CAN controllers can be classified in two main families:

- BasicCAN

This approach provides only few mailboxes for message transmission and reception. This solution allows very compact implementations on silicon.

Advantages:

– Very cost-effective and therefore well suited to 8-bit microcontrollers

Drawbacks:

– Requires CPU resources for filtering, identifier recognition and signal storage
– High software real-time constraints on message reception

- FullCAN This approach provides more mailboxes than messages to handle. In the past, 16 mailboxes were sufficient, today 32 or 64 are standard. Advantages:
- Autonomous message filtering
- One static identifier per mailbox

Drawbacks:

- 16 mailboxes are not sufficient for body applications
- 32 mailboxes to expensive for 8-bit microcontrollers
- Static usage of mailboxes not efficient in body application.

3 BXCAN AND BECAN: THE NEW CAN SOLUTIONS

As mentioned previously for cost efficiency reasons, an 8-bit microcontroller cannot afford to have a huge FullCAN controller with 32 mailboxes. Therefore ST's new solutions are based on a BasicCAN architecture, which has been extended to meet body application requirements.

3.1 BXCAN MAIN FEATURES

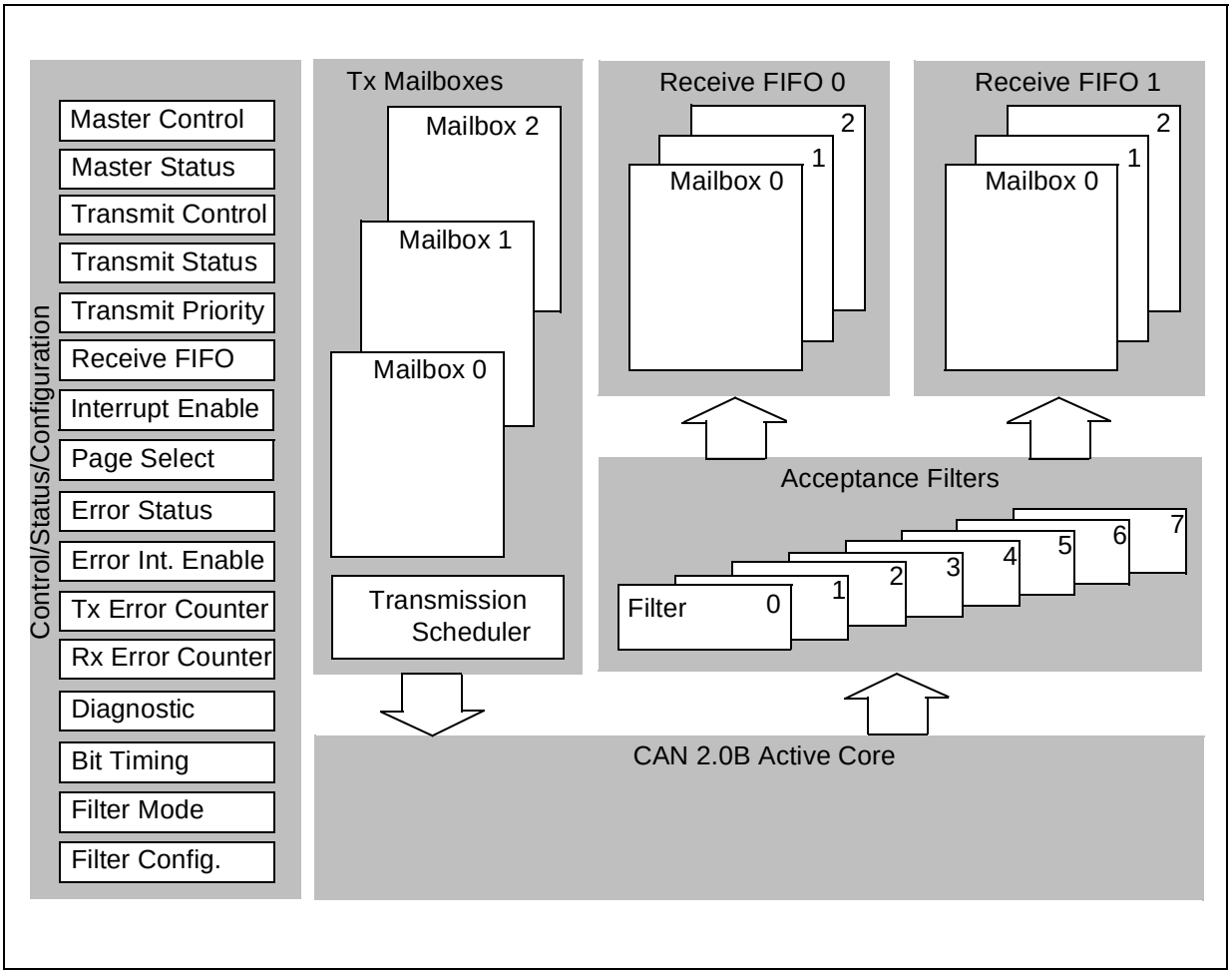
The basic extended CAN - called bxCAN - supports:

- CAN protocol version 2.0 A, B Active
- Bit rates up to 1Mbit/s at 8MHz
- Three transmit mailboxes
- Priority by identifier or FIFOs
- Two receive FIFO with three stages each
- Eight scalable filters
  - Can be associated with FIFO 0 or 1
  - Identifier list feature
  - Filter match index

3.2 CAN CORE

Although the CAN core is the heart of any CAN controller, this is also the most common part. This means that all CAN cores have to be compliant with the CAN standard and from an application point of view do not differ from each other. The processor interface providing the mailboxes, filters etc. must meet the application requirements and can evolve accordingly. For this reason the bxCAN is based on the BOSCH CAN core of the C_CAN product.

Figure 1. Figure 1: bxCAN Block Diagram



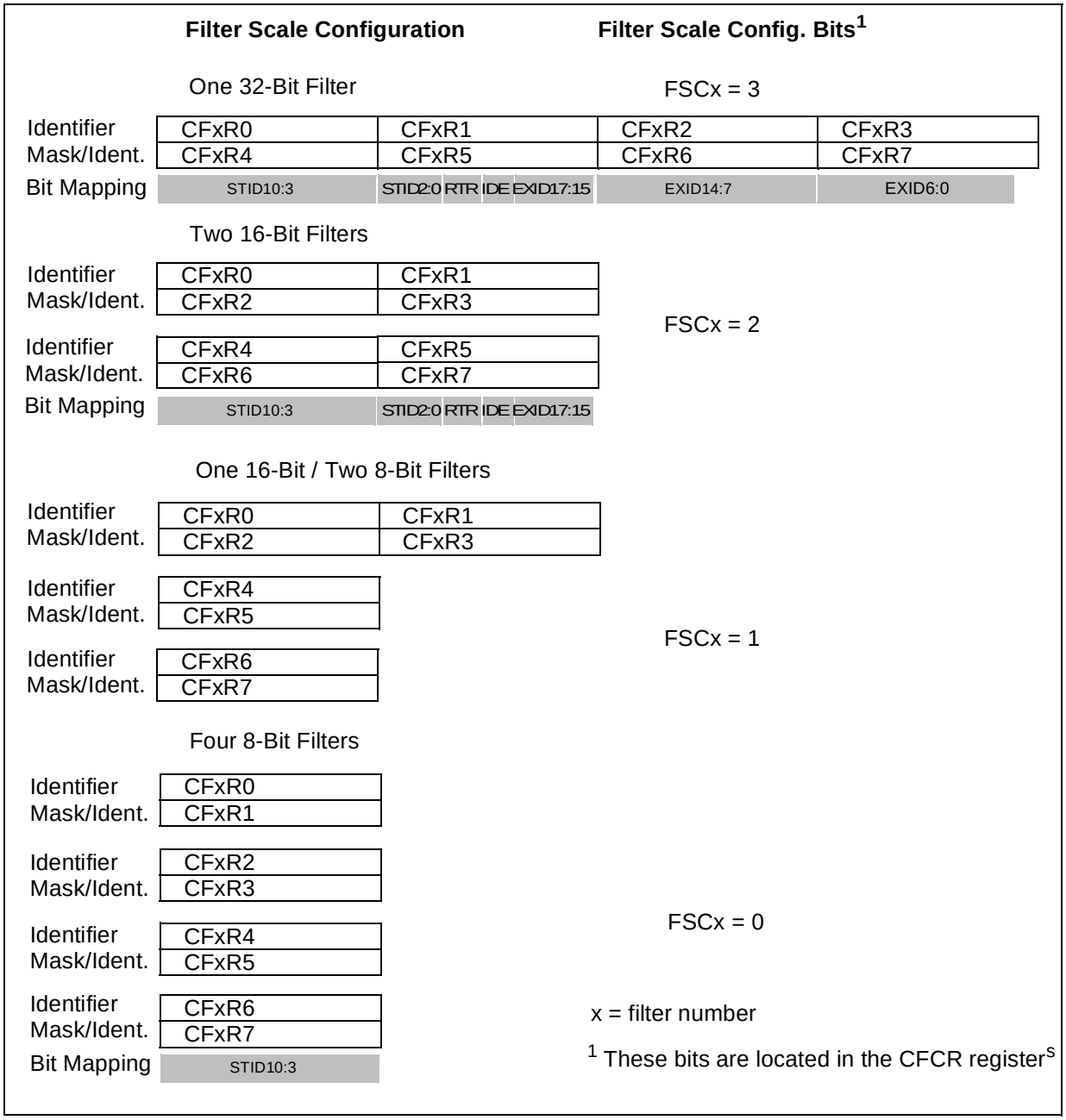
3.3 TRANSMISSION HANDLING

Three transmit mailboxes are provided to the application to set-up messages for transmission. Three mailboxes reduce software queuing and allow deterministic transmission. The Transmission Scheduler decides which mailbox has to be transmitted first according to the priority rules. Transmit Priority Rules BxCAN provides two modes for the transmit priority:

- In identifier mode when more than one transmit mailbox are pending, the identifier of the message stored in the mailbox gives the transmission order. The message with the lowest identifier value has the highest priority according to the arbitration of the CAN protocol.
- In FIFO mode the priority order is given by the transmit request order. This mode is well suited for segmented transmission.

3.4 MESSAGE FILTERING

Figure 2. Filter Scale and Configuration



One of the bxCAN's key improvements is the extended filter mechanism avoiding any message filtering by software. This pure hardware filtering makes the CPU performance independent from the CAN bus traffic.

Hardware filtering:

- Saves CPU resources required for software filtering

- Eases the software development evaluation, as even if the CAN bus traffic is not well defined at the beginning of the project, changes will not impact the CPU behaviour
- Eases the integration of this ECU in a new system To fulfil this requirement the bxCAN Controller provides eight configurable and scalable filters to the application, in order to receive only the messages the application needs.

Scalable Width

To optimise and adapt the filters to the application needs, each filter can be scaled independently. The following combinations are possible:

- One 32 bit filter for filtering the STDID[10:0], IDE, EXTID[17:0] and RTR bits.
- Two16 bit filters for filtering the STDID[10:0], RTR and IDE bits.
- Four 8 bit filters for filtering the STDID[10:3] bits. The other bits are considered as don't care.
- One 16 bit filter and two 8 bit filters.

Furthermore each filter can be configured in mask mode or in identifier list mode.

Mask mode

In mask mode the identifier registers are associated with mask registers specifying which bits of the identifier are handled as "must match" or as "don't care". This mode is well suited for selecting a group of identifiers, for instance network management messages.

Identifier List mode

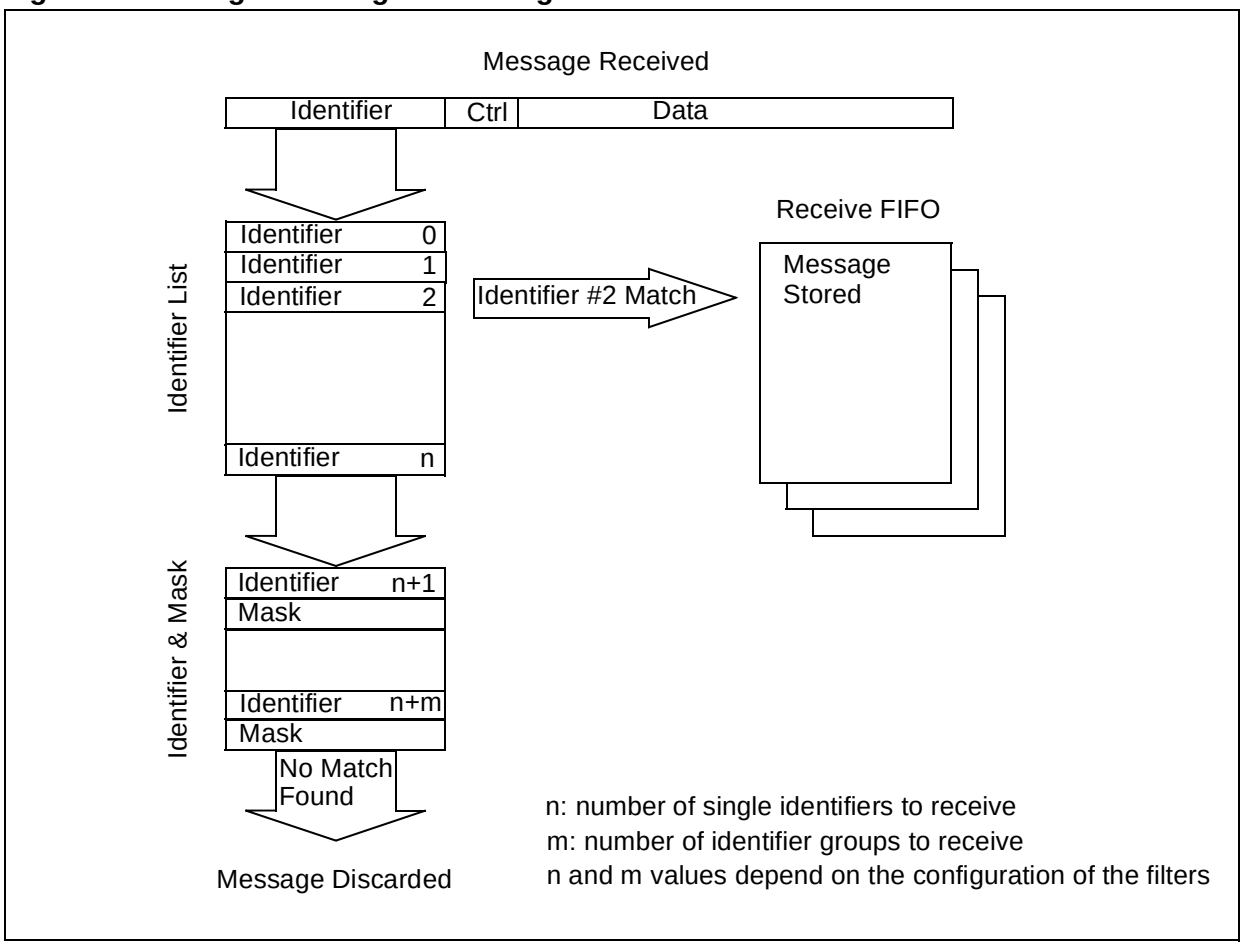
In identifier list mode the mask registers are used as identifier registers. Thus, instead of defining an identifier and a mask, two identifiers are specified, doubling the number of configurable single identifiers. All bits of the incoming identifier must match with the bits specified in the filter registers. This mode is well suited for application messages as their identifiers are quite "randomly" distributed.

Filter Configuration

While the scale and mode configuration must be done during the initialization of bxCAN, the value of the identifiers and masks registers can be modified on the fly.

3.5 RECEPTION HANDLING

Figure 3. Message Filtering and Storage



For the reception of CAN messages bxCAN provides two FIFOs. Each FIFO can store 3 complete CAN messages without CPU intervention. Each filter can be independently associated with FIFO 0 or FIFO 1.

This structure reduces the real-time constraint on message reception on the application side. In order to reduce CPU load, simplify the software and guarantee data consistency, the FIFO is managed completely by hardware. The application accesses the messages stored in the FIFO through the FIFO output mailbox.

Message Prioritization

A drawback of the FIFO architecture is that at reception time the application does not know the level of priority of these messages. Thus all messages have to be handled with the same “urgency”. To address this issue bxCAN provides two independent FIFOs. This allows differenti-

ated handling for high priority messages and non-critical messages. This feature helps to reduce the real-time constraint on the application.

Filter Match Index

Once a message has been stored in the FIFO, the application will transfer the data to the RAM. BxCAN provides a Filter Match Index, FMI, corresponding to the index of the filter the message passed through. The FMI is stored in the FIFO with the received message. Because of their wide spread distribution, CAN identifiers cannot be immediately used as an index to the destination location of the data. The FMI provides a means of accessing a receive message table directly.

Valid Message

A message is considered as valid when it has been received without error until the last but one bit of the EOF field. And it passes through the identifier filtering successfully.

FIFO Overrun

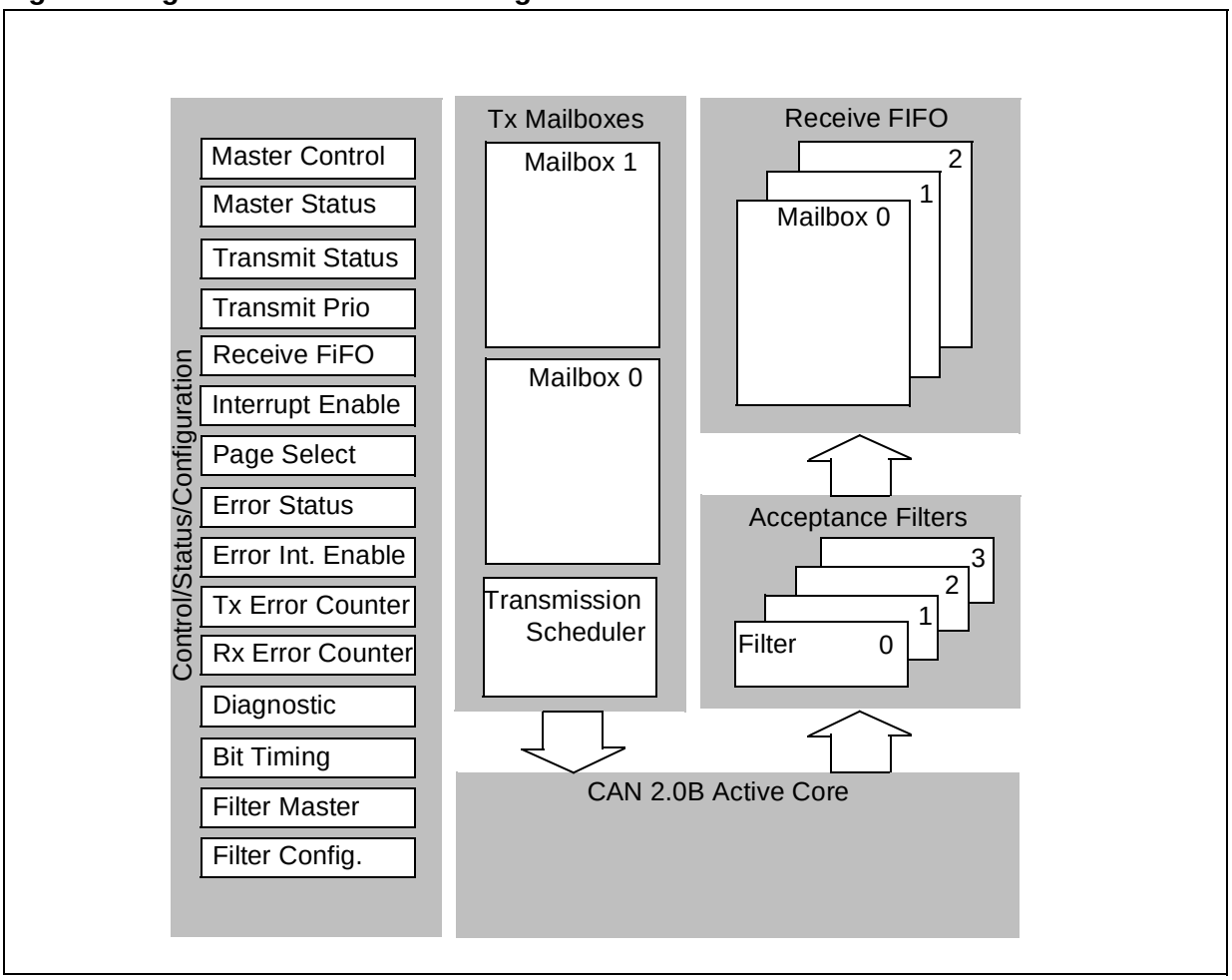
The CAN core has its own message buffer and starts transferring the message to the FIFO once the message is considered as valid. Thus an overrun condition will occur if four valid messages have been received in the same FIFO but not handled by the application. Furthermore this mailbox in the CAN core guarantees that no erroneous message will overwrite a correct one already stored in the FIFO.

3.6 beCAN MAIN FEATURES

The basic enhanced CAN - called beCAN - supports:

- CAN protocol version 2.0 A, B Active
- Bit rates up to 1Mbit/s at 8MHz
- Two transmit mailboxes
  - Priority by identifier or FIFO
- One receive FIFO with three stages
- Four scalable filters
  - Identifier list feature
  - Filter match index

Figure 4. Figure 4: beCAN Block Diagram



The beCAN is based on the same FIFO and filter concept as bxCAN but targets applications requiring less communication resources. This leads to a reduced processor interface functionality.

4 VALIDATION

BxCAN and beCAN are based on the CAN core of BOSCH's C_CAN controller. This approach limits the development risk, reusing the huge experience BOSCH has gained during more than one decade of CAN controller development. To guarantee the compliance of the BOSCH's CAN core with the CAN standard, the C_CAN has been validated according to the ISO standard 16845 „CAN Conformance Testing". The validation has been performed by the c&s group led by Prof. Dr. W. Lawrenz located in Wolfenbüttel, Germany.

For several years, c&s has been well accepted world wide as an organisation for CAN controller validation.

5 CONCLUSION

These two new CAN controllers bxCAN and beCAN have been designed to meet the requirements of today's and tomorrow's automotive body applications.

In particular CAN message filtering (this CPU resources consuming and difficult to evaluate task) has been optimized by the implementation of the „identifier list“ concept and the increased number of filters. Freed from the filtering task, the CPU can completely focus on the application tasks.

With eight filters and two independent receive FIFOs, bxCAN can easily handle a high number and different types of well selected messages. This is an ideal CAN interface for decentralized gateways and all applications with high capacity communications requirements.