|  |  |
| --- | --- |
| **Title:** | **PSS ACU Using a CAN Network**  **SW Component < 1.0 >** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **History** | | | | |
| **Issue status**  (Index) | **Maturity/Date**  (draft/invalid/valid)  (dd-mmm-yyyy) | **Author**  Department | **Check/Release**  Department | **Description** |
| 1.0 | Draft  07-01-16 | Guillermo Ramirez | Oscar Miranda | Creation of the document |
| 18.0 | 11-01-16 | Guillermo Ramirez | Oscar Miranda | Purpose, Definitions And Abbreviations, Realization constraints and targets created. |

**Table of Contents**

[1 Purpose 3](#_Toc433796033)

[2 Definitions and abbreviations 3](#_Toc433796034)

[3 Realization constraints and targets 3](#_Toc433796035)

[4 SW Conceptual design 4](#_Toc433796036)

[5 SW Component internal breakdown 5](#_Toc433796037)

[5.1 Functional Decomposition 5](#_Toc433796038)

[*5.2* *Function void Close( )* 6](#_Toc433796039)

[*5.3* *Function void Open( )* 6](#_Toc433796040)

[5.4 *Function* void blueLed\_Close( ) 7](#_Toc433796041)

[5.5 *Function* void greenLed\_Open( ) 7](#_Toc433796042)

[5.6 *Void Antipinch()* 7](#_Toc433796043)

[5.7 *Function* void countPressingTime(unsigned int counterTime ) 7](#_Toc433796044)

# Purpose

The purpose is to explain and describes the functionality of the PSS ACU program, so the user/programmer will have an overview of the program to facilitate its understanding. This document describes the modules, functions and the functionality of the CAN network. Shows the diagrams of the structure and the classes inside it.

# Definitions and abbreviations

**Definitions**

|  |  |
| --- | --- |
| ACU  PSS  DIC | Airbag control unit  Passive Safety and Security  Drive Information Center |

**Abbreviations**

|  |  |
| --- | --- |
| GPIO  INTC  PIT | General purpose inputs and outputs  Interrupt Controller  Periodic Interrupt Timer |

**References**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N°** | | **Document name** | | **Reference** | |
| 1 | | PSS\_ACU\_Traceability\_Matrix v16.0 | | 1 | |
| 2 | | PSS\_ACU\_CAN database v16.0 | | 2 | |
| 3 | | PSS\_ACU Requirements v16.0 | | 3 | |
| 4 | | MPC5606B Reference Manual.pdf v7.1 | | 4 | |
|  | |  | |  | |
|  | |  | |  | |

# Realization constraints and targets

## Targets

This project involves the development of an ACU module that shall sense the state of the driver and passenger seatbelt and the state of the passenger seat (if there is or not a passenger). The module shall receive CAN message, this messages are used to determine the states of the telltales and other information for the user, this information is send as CAN messages. Here is an overview of the principal project’s features:

* The Seat Belt reminder is realized dividing its functionality in submodules:
  + Driver Reminder
  + Passenger Reminder
  + DIC (Driver information center)
  + Seat Belt Reminder handler
* The Seat Belt Reminder module must check the following data:
  + Operation Mode
  + Vehicle Speed, Distance traveled and Engine status, available as CAN messages.
  + Front Door switch status, available as CAN messages
  + Brake Pedal Pressed status, Transmission shift lever position and Transmission VDA validity status, available as CAN messages.
  + Buckle switch status and occupancy status for vehicle passengers read from Occupancy Handling module.
* The module shall read the sensors as a voltage input.
* The CAN messages, as transmitter or receiver, are described in Table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Table 1. CAN message used in the ACU | | | |
| Message | Transmitter | Receiver | ID |
| Eng\_RPM | ECM | ACU | 0x100 |
| Veh\_Speed | SIM/AVT | ID/ACU | 0x201 |
| Odometer | SIM/AVT | ID/ACU | 0x202 |
| Indication | BCM | ID/ACU | 0x204 |
| Seatbelt Indicator | ACU | ID | 0x601 |
| Chime Request | ACU | ID | 0x602 |

## Constraints

A possible constraint could be the PowerPC architecture of the hardware where the software will be implemented, which is a development board MPC5606B of Freescale. Here are some concerning specifications and a block diagram that could help when trying to export the project into another platform:

* MPC5606B MCU in a 144LQFP package.
* On-board JTAG connection via open source OSBDM circuit using the MPC9S08JM MCU
* Operating Frequency (Max): 64MHz
* Total DMA Channels 16.
* Internal Flash (KB): 512
* GPIOs: 149.
* EEPROM: 64 KB DataFlash®
* RAM: Up to 96 KB
* Timer: 16 bits up to 64 channels

The device has six Controller Area Network (FlexCAN) blocks.

* Each block supports 64 Message Buffers (MB).
* DMA support is not provided.
* It is possible to operate the FlexCAN bit timing logic with either system clock or 4–40 MHz fast external crystal oscillator clock (FXOSC).
* In the case of safe mode entry, the pad associated with CANTX can optionally be put into a high-impedance state (not recessive state)
* Modes of operation:
  + Four functional modes: Normal (User and Supervisor), Freeze, Listen-Only, and Loop-Back
  + One low-power mode (Disable mode)
* 1056 bytes (64 MBs) of RAM used for MB storage
* 256 bytes (64 MBs) of RAM used for individual Rx Mask registers
* Hardware cancellation on Tx message buffers
* Module Configuration Register (MCR): Bits 5, 9, 12, and 13 are reserved
* Error and Status Register (ESR): Bit 31 is reserved

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in ***Figure 1***, which describes the main sub-blocks implemented in the FlexCAN module, including two embedded memories, one for storing Message Buffers (MB) and another one for storing Rx Individual Mask registers. Support for as many as 64 Message Buffers is provided. The functions of the sub-modules are described in subsequent sections.

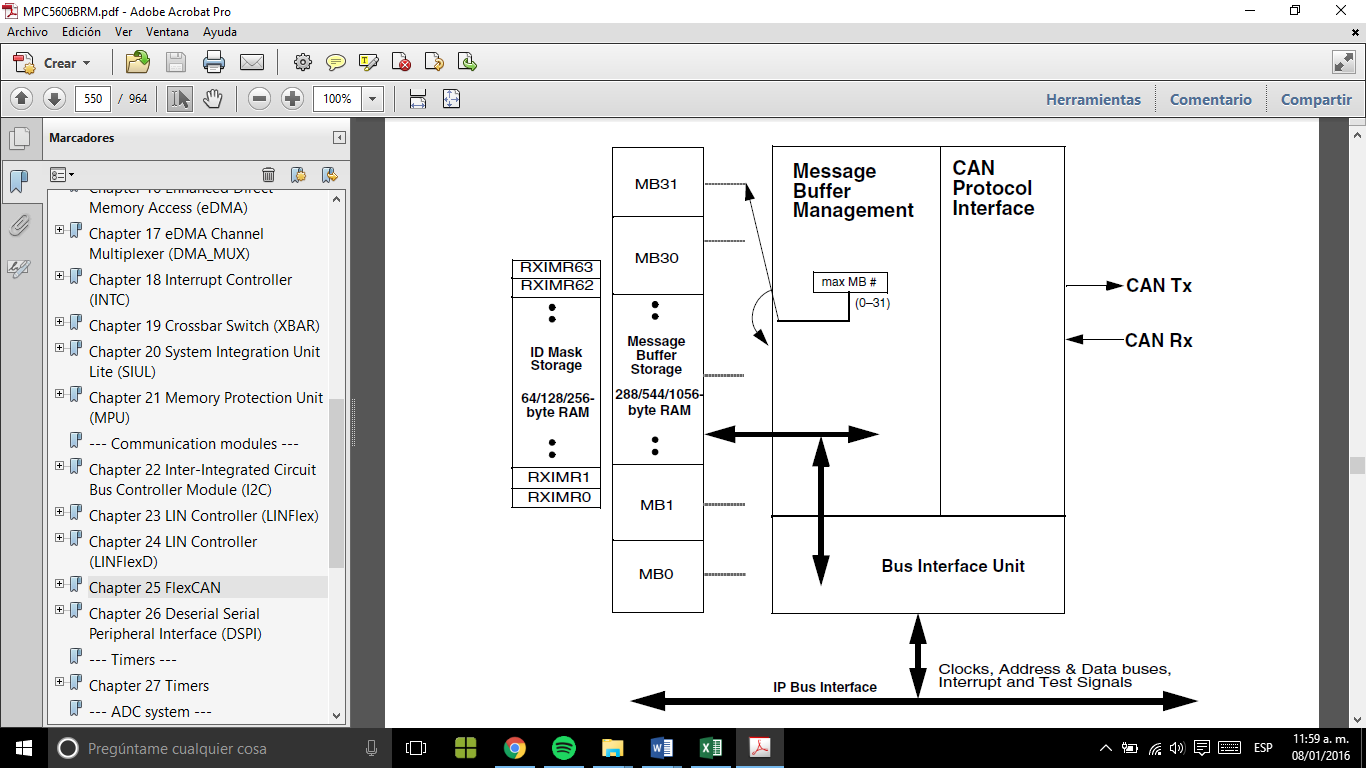


Figure 1. FlexCAN block diagram.

# SW Conceptual design

*To make a conceptual design identify the product functions and the programs parts needed to produce them.*

* *Identifie all relevant parts in a system or sub-system.*
* *Define which parts are inside & outside the system boundary*
* *Define de interfaces between these parts, but also interfaces with neighboring systems and environment.*

# SW Component internal breakdown

For complex SW Components, the designer may define SW Subcomponents. Please consider also the work step "Deal with complexity" provided by the method for Detailed SW Design.  
Note: SW Subcomponents are synonymous with the previous used term "Module"  
  
Mapping to the file structure:  
- Non complex SW Components should be represented by one object file.  
- For complex SW Components each SW Subcomponent should be represented by one object file.  
  
<Subcomponent decomposition if applicable>

## Functional Decomposition

Overview of functions and their dependencies shown by a Static Function Tree  
  
  


**Function Description and Dynamic Behavior**

Provide detailed static and dynamic description of all functions of the SW Component.   
Functions which are defined in other SW Components shall only be referenced in the external interface description!  
The signature description shall be done inside the function header in the source code.  
  
For each function, the following section should be copied

## ***Function void Close( )***

|  |  |
| --- | --- |
| **Description** | Lifts the window |
| **Return Value** | *There is no return value* |
| **Precondition** | Only can be called when the when up button is pressed |
| **Post condition** | *Leds’ transition down-to-up executes* |
| **Error Conditions** |  |

**Dynamic Behavior**

State Chart1, Flow Chart1

1 *Preferred*  
*In this document, the dynamic behavior shall be designed on an abstract level showing the principle workflow of a function. Do not show the detailed implementation to ensure that the design description can be maintained with a reasonable effort. The target is not to show the complete detailed implementation 1:1.  
  
The detailed design shall reflect in detail what a function is doing from a black box view. The internal details are useful on an abstract, but not very detailed level.****If the function is not complex a short textual description might be sufficient and a graphical description is not needed.***

*Symbol and function names shall be self explaining.   
The link to the implementation may be provided by using the same names as in the design or by a comment showing the full name followed by the declaration showing the implementation.*

## ***Function void Open( )***

|  |  |
| --- | --- |
| **Description** | Lowers the window |
| **Return Value** | *There is no return value* |
| **Precondition** | Only can be called when the down button is pressed |
| **Post condition** | *Leds’ transition up-to-down executes* |
| **Error Conditions** |  |

**Dynamic Behavior**

State Chart1, Flow Chart1, Nassi Shneiderman

1 *Preferred*  
  
*In this document, the dynamic behavior shall be designed on an abstract level showing the principle workflow of a function. Do not show the detailed implementation to ensure that the design description can be maintained with a reasonable effort. The target is not to show the complete detailed implementation 1:1.  
  
The detailed design shall reflect in detail what a function is doing from a black box view. The internal details are useful on an abstract, but not on a very detailed level.   
  
Symbol and function names shall be self explanatory.   
The link to the implementation may be provided by using the same name as in the design or by a comment showing the full name followed by the declaration showing the implementation.*

## ***Function***void blueLed\_Close( )

|  |  |
| --- | --- |
| **Description** | Turns on the blue Led |
| **Return Value** | *There is no return value* |
| **Precondition** | Antipinch functionality is disabled |
| **Post condition** | *Turns on/off Led* |
| **Error Conditions** |  |

## ***Function***void greenLed\_Open( )

|  |  |
| --- | --- |
| **Description** | Turns on the green Led |
| **Return Value** | *There is no return value* |
| **Precondition** | Antipinch functionality is disabled |
| **Post condition** | *Turns on/off green Led* |
| **Error Conditions** |  |

## ***Void Antipinch()***

|  |  |
| --- | --- |
| **Description** | Stops and lowers the window |
| **Return Value** | *There is no return value* |
| **Precondition** | Only up button was pressed |
| **Post condition** | *Block inputs during 5 seconds* |
| **Error Conditions** |  |

## ***Function*** void countPressingTime(unsigned int counterTime )

|  |  |
| --- | --- |
| **Description** | Counts |
| **Parameter 1** <input| output| inout> | *counterTime (input)* |
| **Return Value** | *There is no return value* |
| **Precondition** | Up, down or antipinch button must have been pressed |
| **Post condition** | Call up(), down, or antipinch functions |
| **Error Conditions** |  |