SLIO-CAN CAN-Linked I/O Based on COP884BC

National Semiconductor
Application Note 1073
Tobias Wenzel
June 1997

ABSTRACT

Main applications of the CAN Link I/O lies in automotive sensor/actuator systems, but also other applications in industrial systems are of interest. The specification is based on the Microcontroller COP884BC, which has an integrated CAN interface as a peripheral block, but the conversion of the software to COP888EB, National Semiconductor's second CAN Microcontroller, can be done quite easily. The features of the SLIO are restricted the hardware limits like Pinout, periphery blocks, core, and interrupt possibilities of the COP884BC. The features of the SLIO specification were divided in Processes, which gave an overview the communication between CAN, internal processing, and I/O pins of the SLIO.

1.0 INTRODUCTION

1.1 SLIO-CAN — Module Concept

CAN integration variants reach from stand-alone solutions to Microcontroller and SLIO solutions. To reduce the costs the Microcontroller- and especially the SLIO variant are very important, because external components in these solutions are avoided and this reduces the costs as well. Many nodes in vehicle network systems are used to manage Input/Output transfers to sensors and actuators only. These nodes have consisted usually of a Microcontroller with integrated microprocessor, CAN interface, RAM, ROM and several I/O-function blocks. The customer develops his special solution of the transfer with a software program, which is programmed into the controller. Microcontroller solutions are very universal and the customer has many possibilities to implement his application. On the other hand, the customer needs a little time to develop and test the software, learning the processor architecture and to handle the several development tools. Contrary to the microcontroller solution the idea of a Serial Link I/O/-chip solution is to reduce the cost of developing and testing. This CAN integration variant has preprogrammed logic inside, which predefines the I/O-pin CAN data transfer. The function variety is limited so the SLIO is used for more or less standard I/O-Functions. The data transfer to/from SLIO devices is limited to two databyte messages and this messages are used to configure or change the registers within the SLIO memory (see *Figure 1*). In spite of the limited variety of function capabilities, Serial Link I/O's can replace microcontroller solutions in order to spare development time and costs.

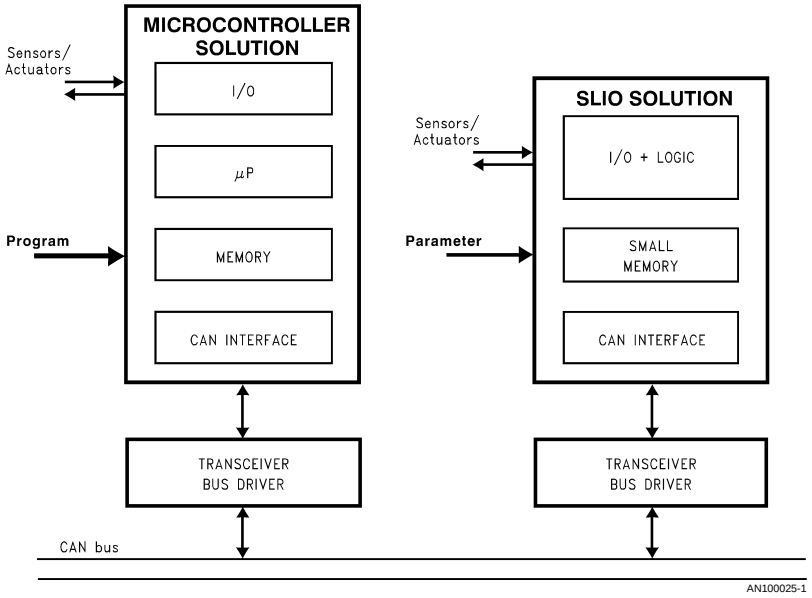


FIGURE 1. CAN Integration Variants Microcontroller/SLIO

The following documentation describes the internal construction and the important internal processes of the implementation of the SLIO.

AN-1073

1.2 SLIO Specification

The SLIO is presented as an application (see *Figure 2*), which is based on the SLIO-CAN Principle with advanced input/output functions and other interesting features like reading identifier from the external EEPROM and a CAN bus rate up to 1 Mbaud. The highlight of this new specification is the external EEPROM mode. In this mode all 11 identifier bits from a standard frame can be influenced with 4 I/O-pins.

In the Module Concept the SLIO has a small memory, called SLIO register block, which contains several registers (parameters). These parameters can be configured by CAN messages. But in addition to that, these registers can also be configured by the external EEPROM during the initialization phase.

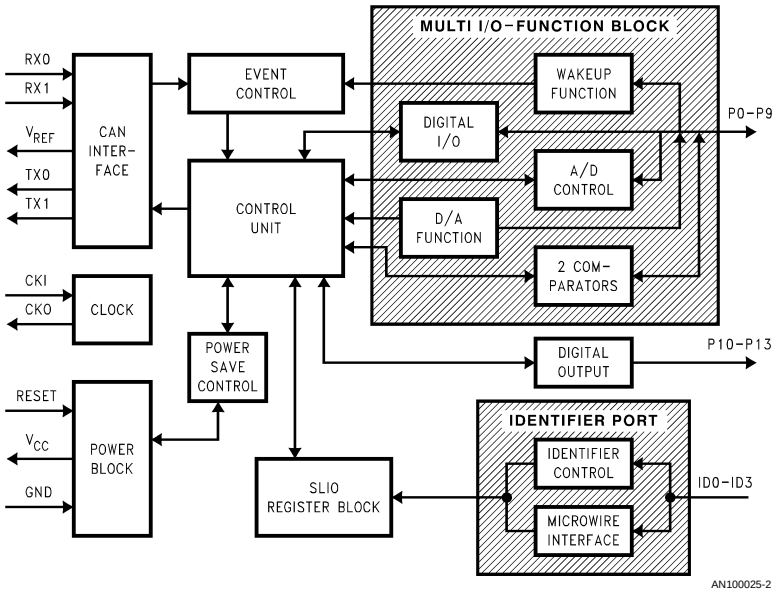


FIGURE 2. SLIO Block Diagram

In *Table 1* the collection of all SLIO registers is shown. By changing these parameters the configuration of Multi I/O-Function Block, power save condition and CAN bus mode condition within the CAN interface will be changed. Furthermore, some parameters are defined for the CAN identifier and the CAN prescaler. But this parameter can only be configured by CAN identifier port.

TABLE 1. SLIO Register Block

Name	Function	config. over CAN	config. over EEPROM
IN1	read status P0 to P7	read only	no
PE	config P0 to P7 positive edge	yes	yes
NE	config P0 to P7 negative edge	yes	yes
OD1	write data to P0 to P7	yes	yes
DD1	config P0 to P7 data direction	yes	yes
IN2	read status P8 to P13	read only	no
OD2	write data to P8 to P13	yes	yes
DD2	config P8 to P13 data direction	yes	yes
ADC	read specified analog input	read only	no
DAC1	config analog output	yes	yes
DAC2	config analog output	yes	yes
ACT	analog output control	yes	yes
CCT	comparator control	yes	yes
CTR	configuration register	yes	yes

The following important features of the SLIO-CAN are summarized.

Advanced Functions

- three identifier programming modes
  - — pin mode
  - — EEPROM mode
  - — pin_e2 mode
- automatic setup of SLIO functions
- 7 pins support A/D-conversion
- bus speed up to 1 Mbit/s
- local oscillator
- local wakeup

Standard Functions

- configuration of different bus modes
- two power save modes: SLEEP/NAP
- PWM D/A-output 8/10 bit
- comparator logic
- external event recognition
- I/O pins individually configured
- WATCHDOG™ mode

1.3 COP884BC Microcontroller

The solution consists on the one hand of the COP884BC Microcontroller, which has an integrated CAN interface and on the other hand on the software, which controls the data transfer process (see *Figure 3*).

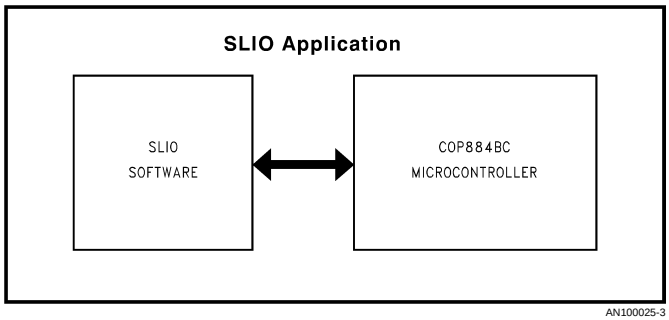


FIGURE 3. SLIO Application

In *Figure 4* Block Diagram the Microcontroller COP884BC with its peripheral blocks is shown. The COP884BC belongs to the family of 8-bit Microcontroller from National Semiconductor. It contains all program and data memory internally and additionally it contains peripheral blocks like configurable I/O-ports, one programmable 16-bit timer, idle timer, serial interface MICROWIRE/PLUS™, CAN interface, two comparators and Multi Input Wakeup function in order to generate external interrupts. The COP8 core supports two power save modes HALT/IDLE and can operate on interrupts from internal and external sources.

The memory organization is based on “Harvard” architecture, in which the program memory (ROM) is distinct from the data memory. One exception from the conventional Harvard architecture is the instruction called Load Accumulator Indirect (LAID). This instruction allows access to data table stored in program memory, which is also very useful for optimizing code. The packages are 28-pin SO with general 18 I/O pins and 20-pins SO with 10 I/O-pins. The instruction cycle time is CKI/10, because all internal COP8 operations are serial. The highest allowed value of CKI is 10 MHz (1 µs instruction cycle time tc).

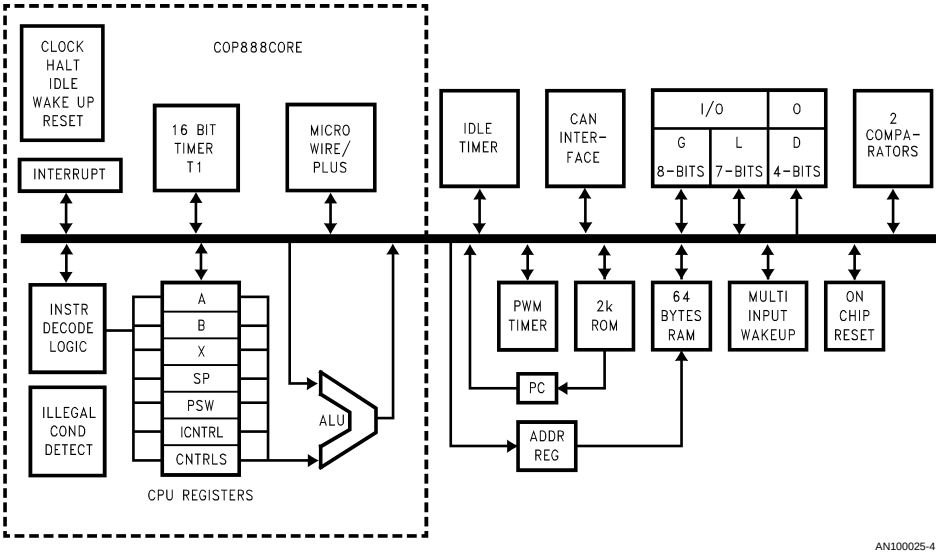


FIGURE 4. COP884BC Block Diagram

CAN Interface

The task of the physical CAN interface is to manage the protocol layer of the CAN communication. That means at first to transmit and to receive frames in correct CAN format (SOF, identifier, Control Field, CRC, ACK, EOF), secondly an error management logic insures the security of the data transfer and execute an error frame if an error has occurred. The interface works independent from the COP8-CORE and can be configured over Control Registers in RAM (see *Figure 5*). The local CAN bit rate will be executed in the bit time logic (up to 1 MBit/s) depending on the frequency CKI.

Important for the SLIO implementation is that 2 data Byte messages can be automatically retransmitted. Thereby the software expense for the implementation can be reduced because the SLIO concept demands only communications of two data bytes. Additionally the SLIO has the possibility to change the bus mode from single wire mode to dual wire mode. This will be managed with the CAN input module and the CAN output module, which can be configured over a CAN interface control register.

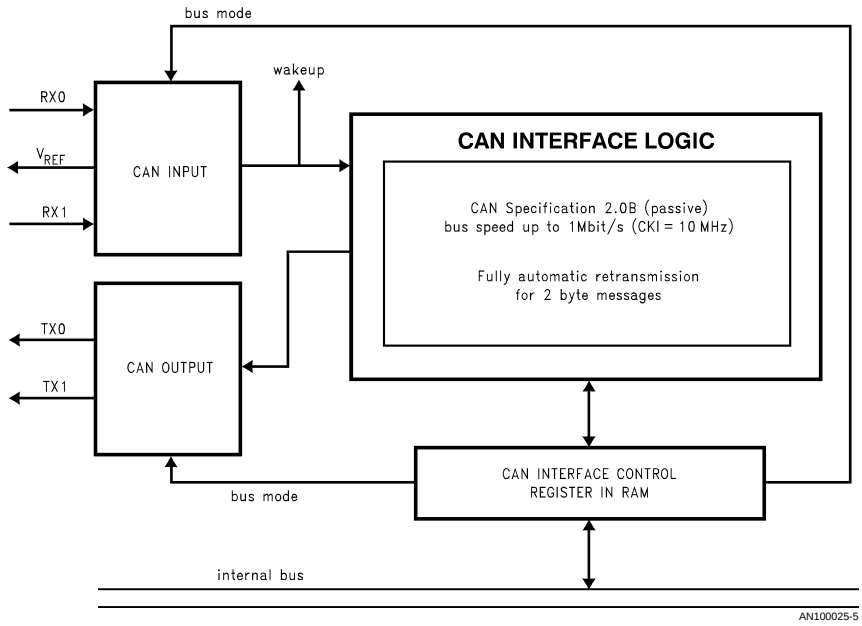


FIGURE 5. CAN Interface Block Diagram

Referring to the network nodes, it is possible to use extended nodes and COP888BC µC together in one CAN network, because the CAN interface is compatible with CAN Specification 2.0 part B. However, extended frames can only be accepted for errors, but are not processed.

More details referring to the configuration of the CAN Interface can be found in the COP884BC datasheet.

MICROWIRE™ Interface

This serial interface of COP884BC is important in order to read data from an external EEPROM. Additionally, other peripherals which can be connected to MICROWIRE, should have the capability to read from the EEPROM with the same chip select. In order to manage that, all users of the MICROWIRE bus have to switch between logical high and high impedance state. When the chip select is high, one of the users reads from the EEPROM at this time and the other nodes can not do this at the same time. In this context the pull down resistor of the chip select wire is necessary so that in high impedance condition the state is defined.

The Master (in this case COP884BC) creates the shift clock (SK), which can be programmed between 2*tc and 8*tc. For the data communication with the EEPROM the highest shift clock rate of CKI/20(2*tc) is possible. This clock time was selected for the SLIO implementation.

In *Figure 6* hardware connection of the EEPROM and COP884BC is shown without any peripherals connected to the MICROWIRE interface.

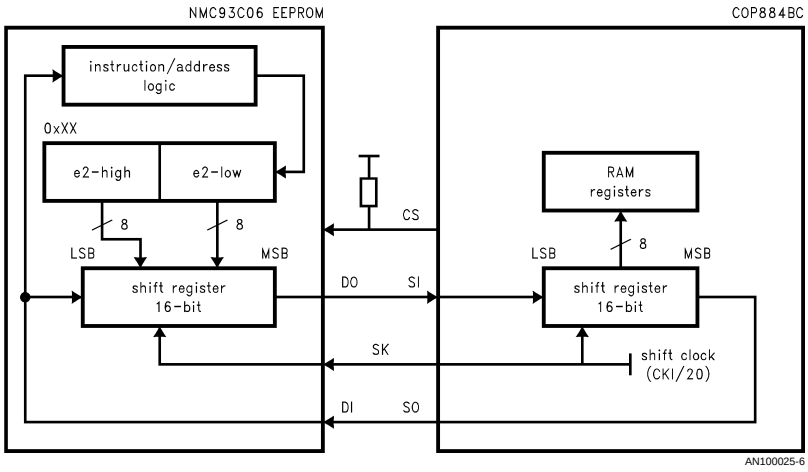


FIGURE 6. Data Transfer EEPROM → COP884BC

Multi Input Wakeup

Two functions are important for SLIO implementation:

- create an interrupt from external positive/negative edges
- wake up the Microcontroller from power save (HALT/IDLE mode)

Figure 7 shows the wakeup sources of COP884BC. One of the external wakeup sources is the CAN input module (see also *Figure 5*), which is internally connected with the MULTI INPUT WAKEUP block and can not be influenced by software.

The other external wakeup sources are the COP884BC L-Port pins L0 to L6. These pins can be controlled by software with the control register located in RAM. This means that the enable or disable can wakeup separately or select the edge to execute wakeup functions. When one wakeup is executed, the Microcontroller will wake up from HALT/IDLE condition if it is in power save condition and after that it will jump into the wakeup interrupt routine. When the Microcontroller is active, the process goes directly into the interrupt process.

More detailed information about MULTI INPUT WAKEUP and Power Save modes are described in the COP884BC datasheet.

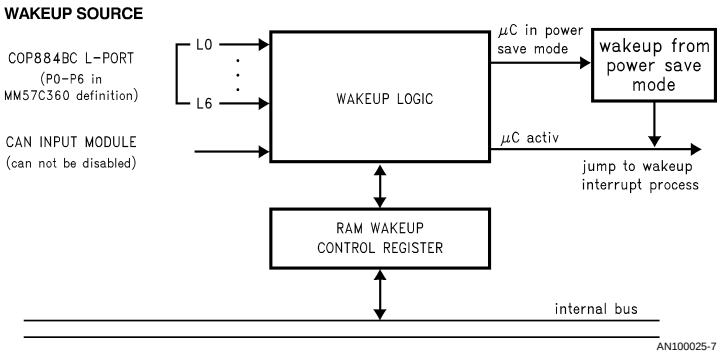


FIGURE 7. Structure of MULTI INPUT WAKEUP

2.0 SYSTEM DESIGN

2.1 SLIO Processes

The functionality of the SLIO can be separated in different processes, which are described in the Section “System Design”.

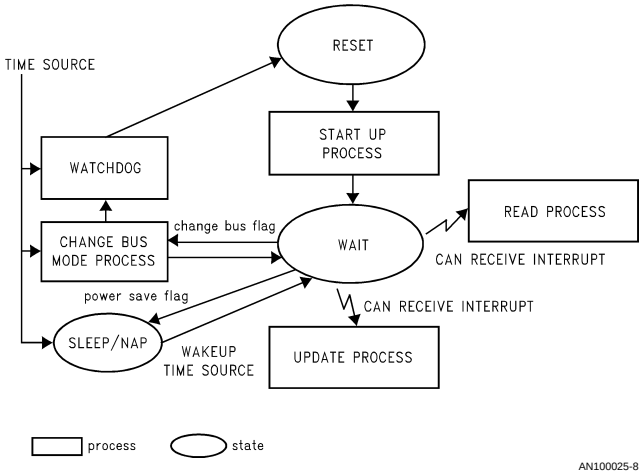


FIGURE 8. SLIO Processes

Figure 8 shows processes that are described in rectanglular objects and on the other hand states which are described in round objects. In order to get the system into an active state, a start up process has to be executed. In this state all important configurations will be done, and after that, the CAN system master is informed with a status message that this SLIO node is ready to communicate via CAN. That means that the device is waiting for messages or actions from the CAN or the I/O pins.

CAN messages can be classified in two different kinds. On the one hand there are “Read messages”, which should send information about the status of I/O pins to the system master. This will be done in the Read Process. Examples of this classification of messages are “read analog input” or “read digital status” of pins P0 to P7 (P8 to P13). On the other hand there are “Update Messages”, which change the configuration of the SLIO. This means that the SLIO Registers are updated and because of that the configuration of the peripheral controller blocks is changed. This procedure is described as the “Update Process”.

Two other processes are shown in *Figure 8*: WATCHDOG process and change bus mode process. These are linked to an internal SLIO time source. The basic unit of this time source is called chip base time (Bt) and is defined to 40960/CKI. WATCHDOG and Change Bus Mode can be executed depending on multiples of Bt and these multiples can be configured via the SLIO Register individually. Additionally, the NAP condition also has a relationship to the internal time source, because the wakeup time from NAP mode is configurable in multiples of Bt too.

2.2 Start Up Process

The start up process describes the process after RESET until the SLIO is ready to communicate with CAN or with sensors/actuators, which are connected on the I/O-pins. A RESET can be executed firstly by toggling the RESET pin or secondly by an internal software WATCHDOG logic, when it is enabled. The Flow of the process is shown in *Figure 9*.

The ID Configuration and SLIO Registers Configuration subprocesses read the identifier port and will be explained in later section. Next, all peripheral Controller blocks are initiated by the status of the SLIO Register. When this process is finished the status message will be created, which informs the system master of the CAN network about the status of the pins P0 to P7. After transmitting this message all interrupts are enabled. Then the device is in condition “Wait”, which means that the SLIO is ready to communicate via CAN and with connected devices on I/O pins.

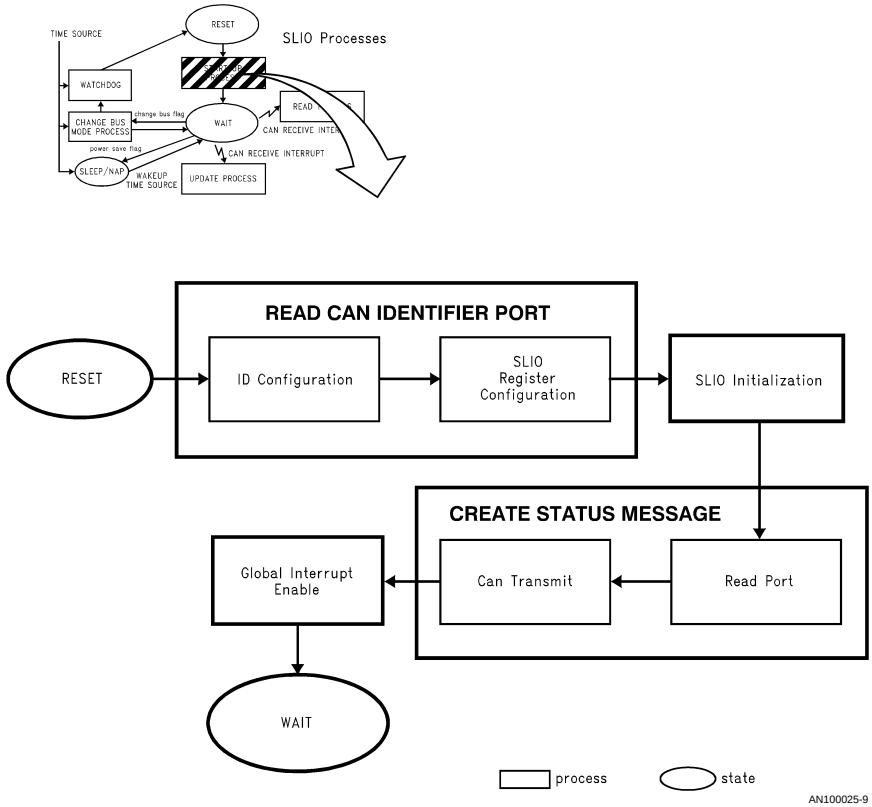


FIGURE 9. Start Up Process

2.2.1 Read CAN Identifier Port

Every Serial Link I/O in the network needs a transmit ID and a receive ID. It is defined globally that after every transmit ID follows the receive ID of the according note. The ID will be programmed through the CAN Identifier Port. This identifier can only be changed by reading this port during initialization and not via the CAN bus.

Basically, there are two different modes to read the CAN identifier. In pin mode only four identifiers from the 11 identifiers in the standard CAN format can be read with pull up/down resistors on pins ID0 to ID3. This mode is the simplest way to program the identifier, but it has the disadvantage that only 16 different SLIO nodes can be connected on one CAN network system. In order to avoid this disadvantage the user has the alternative to use the EEPROM mode. In this mode, all 11 identifiers of standard CAN format can be read from an external EEPROM connected on the CAN identifier port. This port consists of pins from the G-Port (see COP884BC datasheet) and has the capability to connect the MICROWIRE interface under software control (see *Figure 10*). A further advantage of an external EEPROM consists of the possibility to read standard data to the SLIO Registers, which controls the SLIO condition. This process will be done during the initialization phase of the chip and avoids a lot of CAN communication after the initialization in every SLIO node.

In this pin_e2 mode, the identifiers are read in pin mode and the configuration of the SLIO registers can be read from the external EEPROM. In this case the information to go in pin_e2 mode is given by the EEPROM.

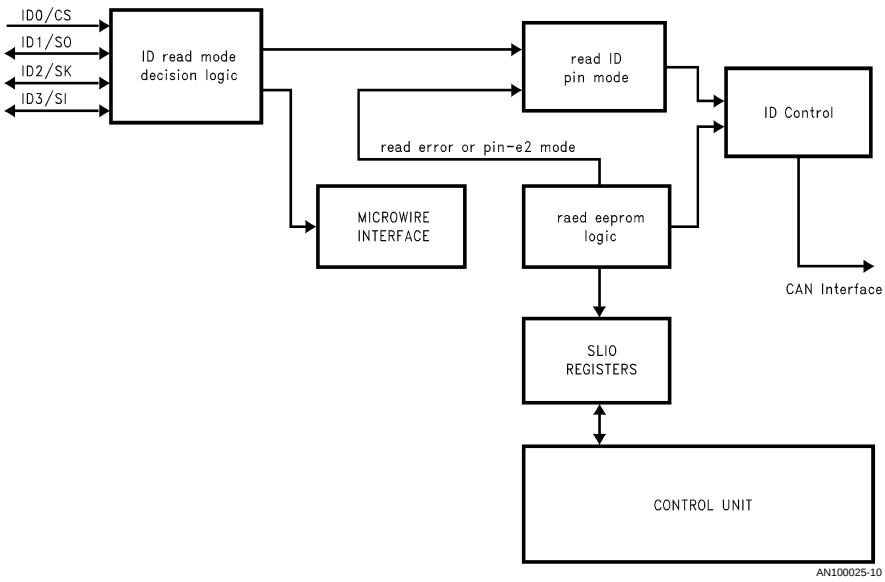


FIGURE 10. Read CAN Identifier Port

2.3 CAN Communication Processes

2.3.1 General

Due to the specification, the identifiers are used to distinguish CAN messages from the different SLIO Nodes and in order to find the direction of the CAN message (to/from the SLIO). This ID can be understood as addresses of CAN messages in a global address space, which is managed from a system master. If the Master Controller in a network wants to change the configuration of the SLIO, it has to send a message to the SLIO with the correct receive iIdentifier (seen from SLIO side). In order to know, which register should be changed through this message, 5 Bits of the first date byte are received to mark the specified SLIO Register. These 5 bits get the name Register Marker. The relationship of the Register Marker and the specified SLIO Registers are shown in *Table 2*, which was defined in specification. The new data to this Register is placed in the second date byte of the CAN frame.

TABLE 2. SLIO CAN Registers

RM (hex)	Register Marker Bit					Name	Function	Dir
	4	3	2	1	0			
00	0	0	0	0	0	IN1	P0 . . . P7 input data	r
01	0	0	0	0	1	PE	P0 . . . P7 positive edge	r/w
02	0	0	0	1	0	NE	P0 . . . P7 negative edge	r/w
03	0	0	0	1	1	OD1	P0 . . . P7 output data	r/w
04	0	0	1	0	0	DD1	P0 . . . P7 data direction	r/w
05	0	0	1	0	1	IN2	P8 . . . P13 input data	r
06	0	0	1	1	0	OD2	P8 . . . P13 output data	r/w
07	0	0	1	1	1	DD2	P8 . . . P13 data direction	r/w
08 . . . 0E	0 . . . 0	1 . . . 1	0 . . . 1	0 . . . 1	0 . . . 0	ADC	A/D read	r
0F	0	1	1	1	1	IN1	P0 . . . P7 input data reset register marker	r
10 . . . 13	1	0	0	x	x	DAC	D/A write	r/w
14 . . . 1B	1 . . . 1	0 . . . 1	1 . . . 0	0 . . . 1	0 . . . 1	n/a	reserved	n/a

TABLE 2. SLIO CAN Registers (Continued)

RM (hex)	Register Marker Bit					Name	Function	Dir
	4	3	2	1	0			
1C	1	1	1	0	0	ACT	analog control	r/w
1D	1	1	1	0	1	CCT	comparator control	r/w
1E	1	1	1	1	0	REF	revision	r
1F	1	1	1	1	1	CTR	configuration register	r

Due to the global address space the SLIO device can communicate with the CAN bus. The next section shows how the device manages this communication internally.

2.3.2 Read Process and Update Process

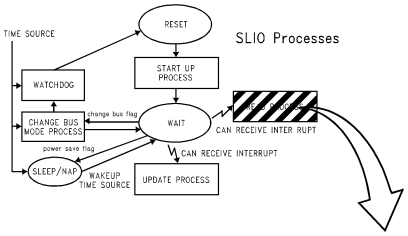
Basically the communication via CAN by these two processes is follows by the same rules:

- disable global interrupt
- CAN receive process
- individual process
- CAN transmit process
- enable global interrupt

The reason to suspend all interrupts during CAN communication lies in the fact that this process should not be disturbed by other actions caused by interrupts like CAN receive/error messages or external events. CAN messages are ignored during this time but the reception of external events will only be delayed until CAN communication is finished.

And what is the difference between Read process and Update process?

For one read process (*Figure 11*) the specified Register Marker defines which read message should be executed. On the one hand, in read process the digital status is read how it was managed by creating a Status Message during the start up process. On the other hand the analog status of one analog input pin can be read. In SLIO Registers the result of reading is saved. After finishing these two different reading processes the status of the result will transmit to the System Master.



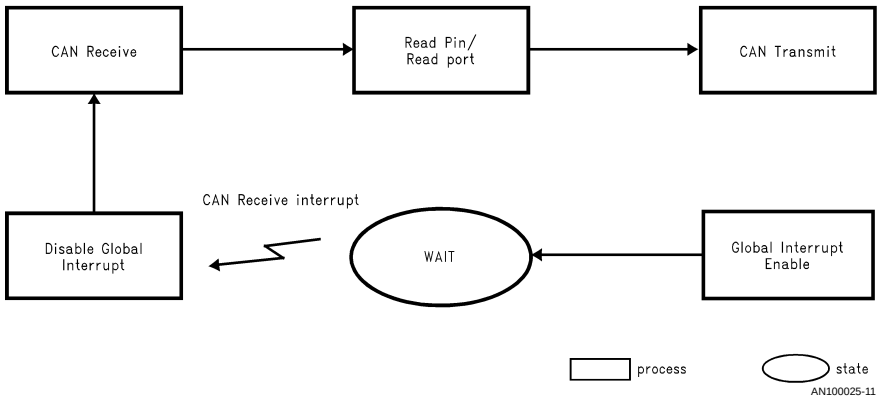


FIGURE 11. Read Process

In contrast to the Read Process, during the Update Process (*Figure 12*) SLIO Control Registers have to be changed in order to reconfigure the condition of the device. After the selection, which is defined through the Register Marker, the SLIO Register will be updated. This updating causes a new initialization depending on the new data. After finishing this process the condition of SLIO will be changed. In order to

show the Master of the CAN system the valid reception, the SLIO will respond with the new data of the specified SLIO Register.

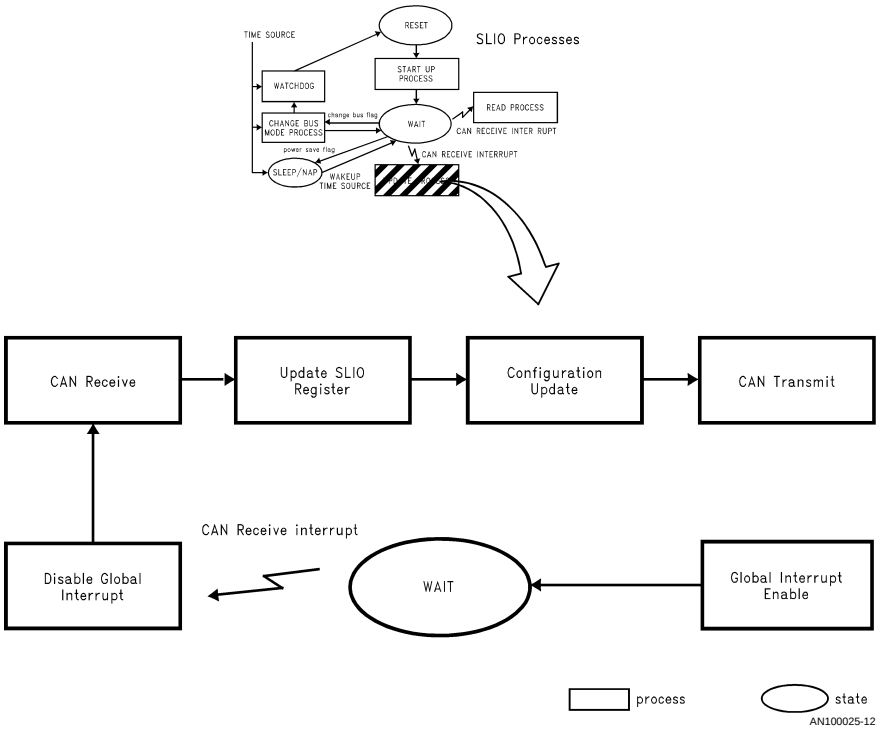


FIGURE 12. Update Process

2.3.2.1 Analog Input Module

Besides reading the digital status of the SLIO pins the analog input function is also a part of the reading process. The idea of this analog input lies not in a high performance A/D-conversion, but in the possibility to read up to 16 different voltage levels by one I/O-pin. *Figure 14* shows an example of reading different voltage levels of a resistor array. This will be done by measuring the charge or discharge time of an external capacitor. The internal construction of an I/O pin (see *Figure 13*) will be supported by the software implementation of the analog input. At first the level of the Schmitt Trigger Input is measured. Depending on the result, low or high, the internal driver, which is controlled by the software charge/discharge logic, charges or discharges the external capacitor.

Schmitt Trigger level = low → charge capacity

Schmitt Trigger level = high → discharge capacity

Afterwards, the time to get the original digital (after Schmitt Trigger) state is measured by a counter register. This counter values consider different input voltages. The problem with this A/D software implementation is shown in *Figure 14*, because the charge or discharge time of the capacitor is dependent on the current and this current is not linear. With voltages near the Schmitt Trigger level, the 8-bit counter value is overflowed and no measurement is possible. Additionaly, this measurement is dependent on CKI and so the choice of R/C values has to be adjusted to the different application conditions.

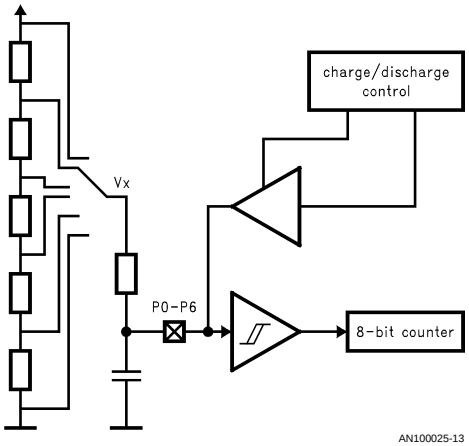


FIGURE 13. Analog Input

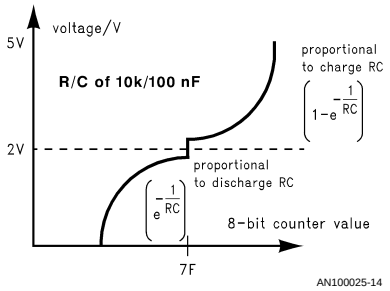


FIGURE 14. Input Voltage Depending on Counter Value

2.4 Time Source Processes

With the IDLE timer of COP884BC, an internal time source can be implemented. This time is dependent on CKI. The toggle of the thirteenth bit of this 16-bit timer can generate an interrupt, furthermore the toggle is latched in the pending flag T0PND. By a CKI of 10 MHz this flag is always set after 4.096 ms. A unit is defined, which is called chip base time (40960/CKI).

Next, two processes will be described, which can be configured depending on the chip base time.

2.4.1 Change Bus Mode Process

The CAN input and CAN output modules support three different bus modes:

- dual wire
- single wire RX0/TX0
- single wire RX1/TX1

In dual wire mode the information is transmitted and received in a differential mode. This means the difference of Signal1 on RX0 pin and the Signal2 on RX1 pin is considered as the message signal. In single wire modes only one of the pins is used for communication. In application it is sometimes useful to switch between the three bus modes. Therefore, three different types to change the mode are defined.

- **automatic mode** — In automatic mode the bus will be checked and the SLIO is configured automatically when it receives a message. When a message is received, the current bus mode will be fixed. This change bus mode, for example, is used when the device is initialized in pin mode (read ID over pin status). Then the bus mode is executed automatically when any message is received. The switch time in this automatic mode is fixed to 8*Bt.
- **WATCHDOG mode** — If in any bus mode no message is received, the WATCHDOG mode causes a RESET. After RESET, the power save mode SLEEP is enabled and this mode is nearly “switched off” from CAN bus. This is useful in order to avoid disturbing the CAN bus with permanent error frames transmitting from the SLIO.
- **cycle mode** — In cycle mode the switching is programmable by the SLIO CTR Register. This change bus mode is specified, for example, in cases when one CAN wire is broken. If cycle mode is enabled the data transfer can be continued by the other valid CAN wire.

2.4.2 Power Save Control Process

The NAP mode in the power save process can also be configured depending on multiples of the chip base time. Contrary to the SLEEP mode, in NAP mode the WAKEUP block and the IDLE timer are still active. This IDLE timer will wake up from NAP mode every Bt.

3.0 SOFTWARE IMPLEMENTATION

3.1 Software Control Structure

In *Figure 15* the structure of the control software for the SLIO is shown. The software is controlled by the control/status register. The status of these control flags influence the flow of the structure.

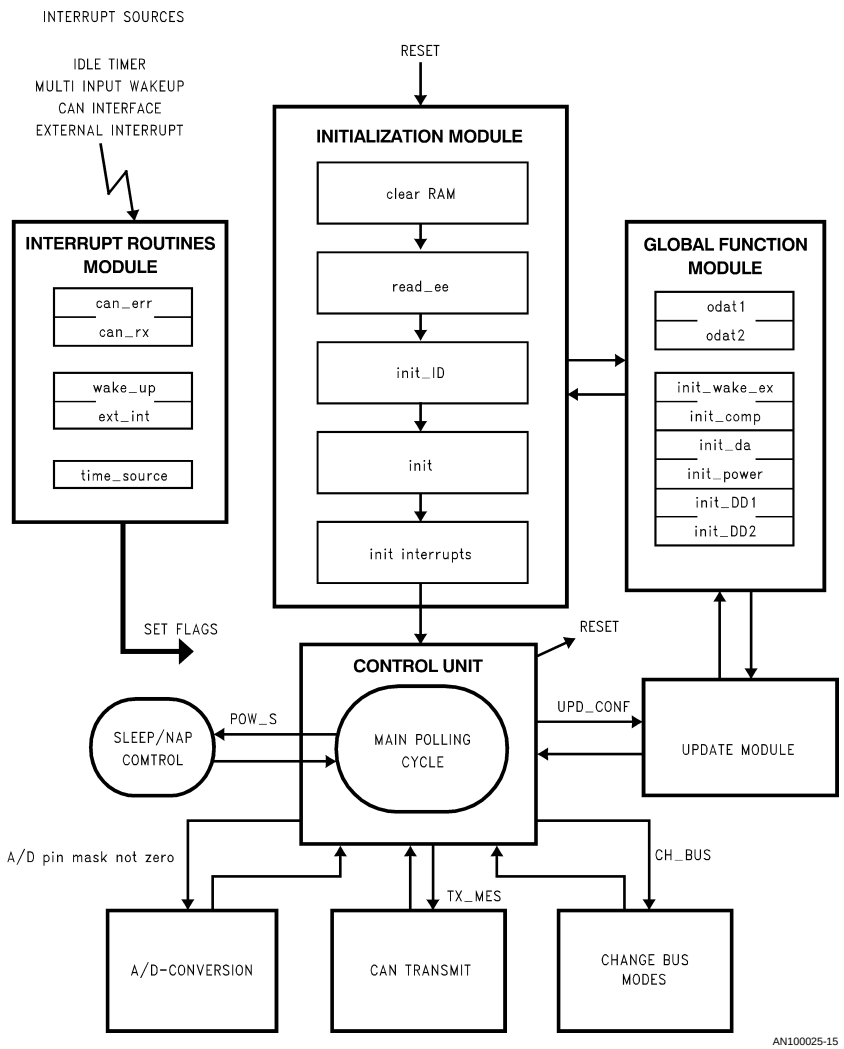


FIGURE 15. Software Control Structure

The initialization module is responsible for the correct configuration of the peripheral blocks of the Controller. This contains trying to read from the external EEPROM (read_ee) and within init_ID analyzing the identifier depending on reading mode (E2/pin mode). After that, the init routines follow the configuration of the peripheral blocks depending on the status of the SLIO Registers, which are configured over an external EEPROM. In pin mode all SLIO Registers are set to zero with the exception of the CAN prescaler register (CAN_PSC), this is set to 03h. This means that the default bus rate of the SLIO is 250 kbit/s.

After that the interrupts (CAN error, CAN receive, Idle timer interrupt, wakeup interrupt, external interrupt) are initiated. These maskable interrupts are still disabled by the global interrupt flag and this flag will be set in the end of the start up process. Then the process is prepared for transmitting a status message to the bus. This means on the one hand to set the UPD_REG flag in order to execute the update process in the control unit and on the other hand to set the IN1_UP flag (UPDAT_ST1), which cause a reading of P0 to P7 in the update module.

It was mentioned in the first draft description that this solution contains a collection of functions in the global function module, which are used firstly from the initialization module and secondly from the configuration update module.

In the left side of *Figure 15*, all interrupt routines are collected and will be executed, when one of the interrupt source creates an interrupt. Two different types of interrupt sources are implemented. On the one hand CAN interface and Multi Input Wakeup, which creates interrupts through actions from outside and on the other hand the Idle timer, which creates

an internal interrupt sequentially after every Bt in order to make an internal time source. This time is needed for NAP control, WATCHDOG and change bus modules.

3.1.1 Interrupt Handling

The interrupt handling is managed from an interrupt vector table, which is placed after the ROM address 00FFh. This interrupt unit leads the process to jump to the start label of the specified interrupt routine. Basically, during the execution of an interrupt routine, all interrupts are disabled. This means that the interrupt process can not be broken, but after finishing the first interrupt routine, all other received interrupts are executed one after another. When all received interrupts are executed, the process jumps back to the previous program address and continues the program flow.

It was described in System Design during CAN communication processes, all interrupts are suspended. These interrupts are wakeup/external interrupts and the time source interrupt. The routines, which are synchronized by this time source interrupt (see System Design) are not time critical. But what is done by external events, which were recognized during CAN communication processes. They are not lost. Although wakeup interrupts and external interrupts are disabled, they are latched in the pending flag registers. Therefore after the CAN process, this register is decoded and for every recognized event, a status message is created. In this way, after the start up process, no external events are lost.

3.2 Software Modules of the SLIO

3.2.1 Initialization Module

Read from the external EEPROM

Within the initialization module (see *Figure 15* Software Control Structure) the subroutine read_ee has the function to read the external EEPROM register in the RAM of COP884BC. This will be processed if the ID0 pin is low. This means that no other peripheral reads the EEPROM at this time. In order to check the ID0 pin, it has to be set as input. The check, if the E2 lies on the ID-pins, will be checked in the subroutine test_ee_mask with the higher byte of the first 16-bit register of E2. These mask identifiers have to be the bit setting 0xAA. Then the process accepts the reading register values and the E2_CON flag in CONTROL1 register is set. Now the flow of read_ee is started, which is shown in *Figure 16*.

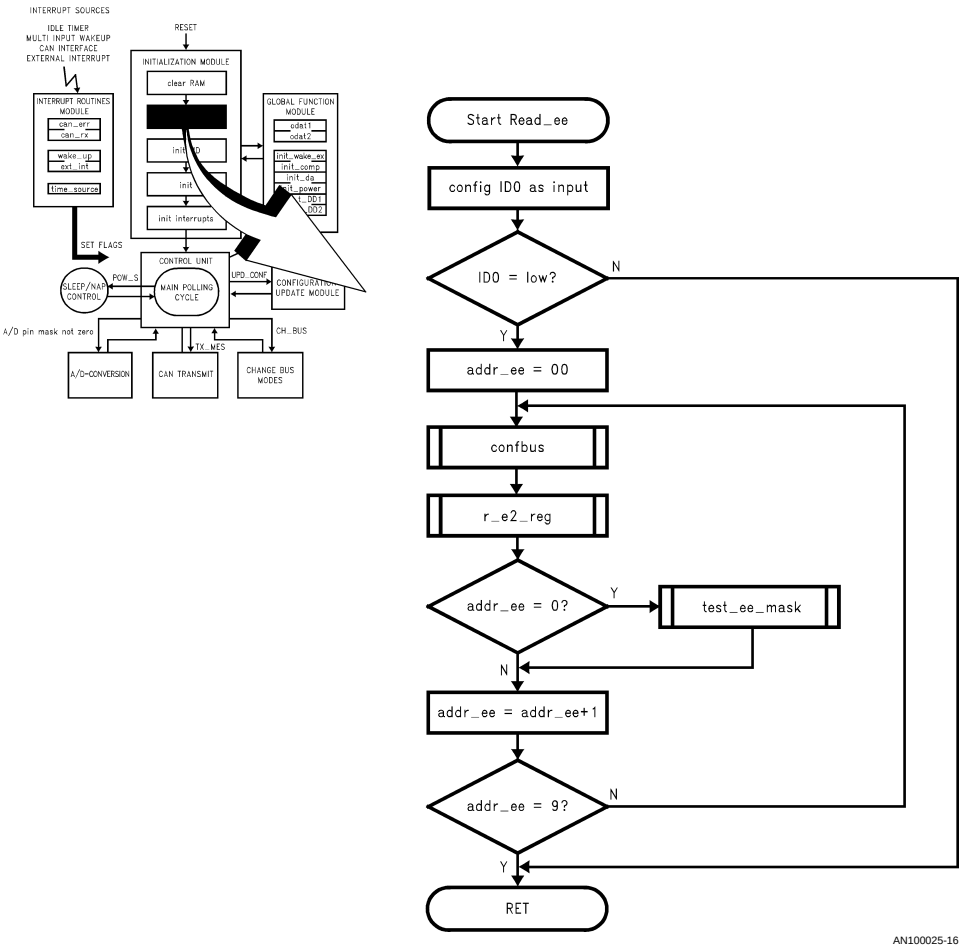


FIGURE 16. Flow of read_ee

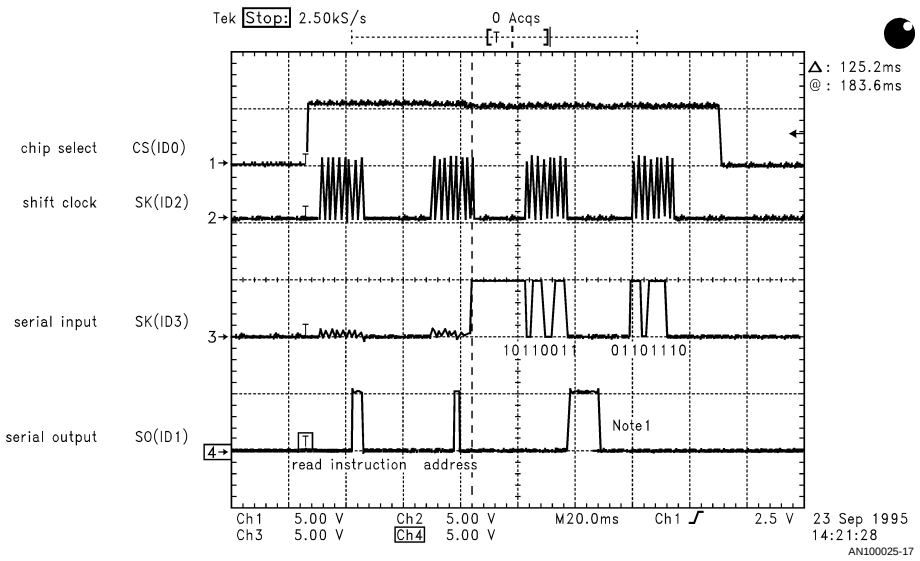
After checking ID0, the process is prepared for reading values of the EEPROM. The number of external 16-bit register is initiated to nine and for every external register there are two RAM-addresses necessary. The first address is pointed through X-pointer.

In the subroutine confbus, the MICROWIRE serial bus system is configured. The next subroutine, r_e2_reg reads one 16-bit register with the address saved in accumulator and writes the data to the RAM place pointed from the X-pointer. The Accumulator and X-pointer are used as variables for this subroutine. In the process, the EEPROM address signed with the variable addr_ee is incremented. This will be cycled until the EEPROM address 0x09 is read.

One 16-bit reading cycle consists of four 8-bit shiftings, which is shown in *Figure 17*.

Example:

- 8-bit shifting one: transmit read instruction (SO)
- 8-bit shifting two: transmit e2address to the EEPROM (SO:address 04)
- 8-bit shifting three: higher data byte transfer (SI: B3)
- 8-bit shifting four: lower data byte transfer (SI: 6E)



Note 1: At this time the MSB of the first databyte is seen in shift register of the controller. After the falling edge the shift register is written out.

FIGURE 17. Reading Cycle of One 16-bit e2Register

3.2.2 Control Unit

After initialization the program is alwayscycled in this main polling cycle (see *Figure 18*). It can only be broken by changing the status of the polling flags or from an interrupt. In addition to that, the WATCHDOG control can create a RESET if WATCHDOG is enabled (CTR register).

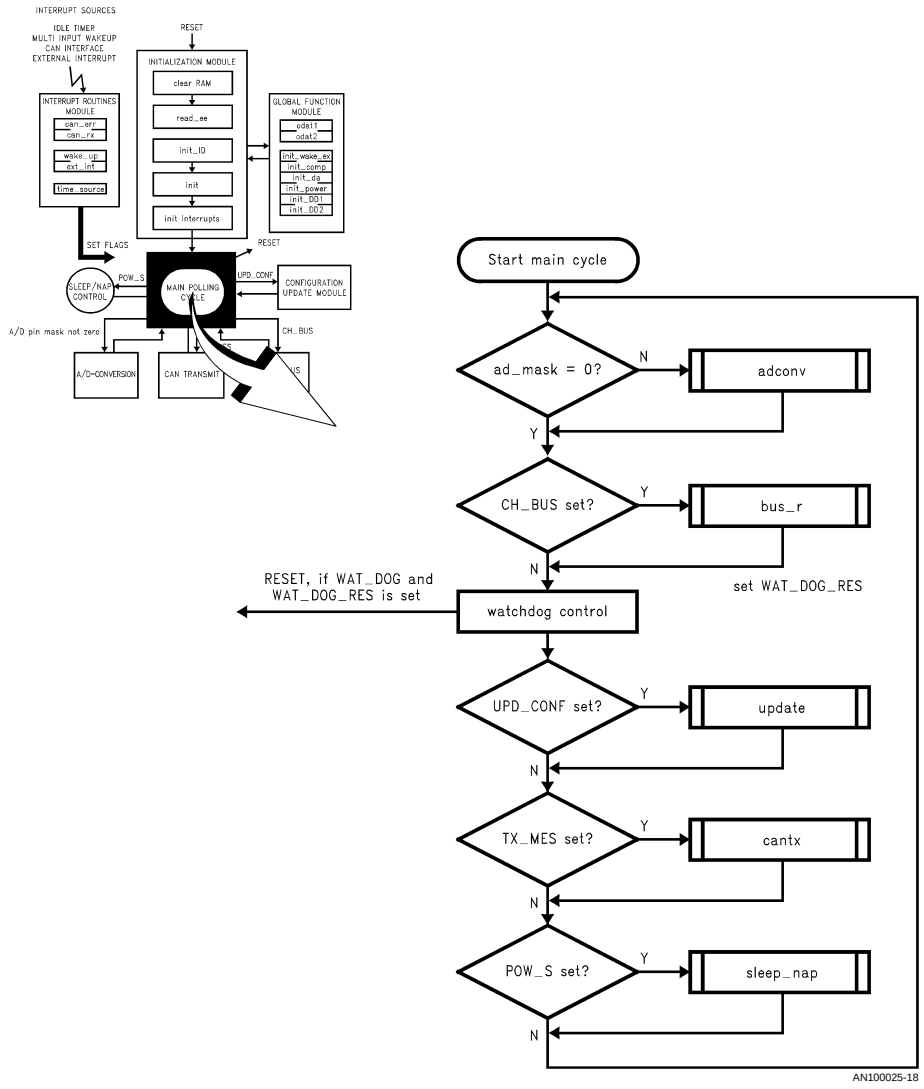


FIGURE 18. Flow of Main Polling Cycle

3.2.3 CAN Transfer Module

CAN Transmit Routine

This routine, which is shown in *Figure 19*, is activated by the Control Unit the TX_MES control flag is set. It loads the transmit data register with specified data and gives information to the CAN interface to transmit a message. This action is only executed if the CAN interface is not in the transmission phase. When it is, the process waits until the transmission is ready. After the request to transmit a new messsage from the CAN interface, the global interrupt flag will be set again. Now the CAN communication from a receiving message (can_rx) updates the SLIO register and changes the configuration of the SLIO (update routine) and transmits a message to the bus to inform the master that the communication was correct.

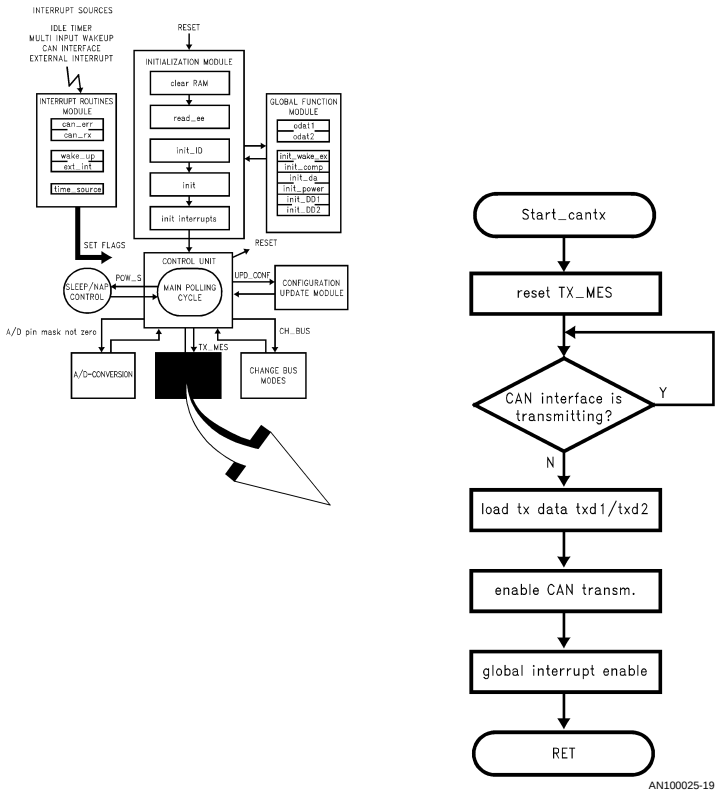


FIGURE 19. Flow of cantx

CAN Receive Routine

After a valid CAN message is received, the CAN interface generates a receive interrupt and can_rx (*Figure 20*) will be executed. All interrupts during the receive process and transmit process are suspended globally. This will be done with the global interrupt mask bit GIE, which is supported from the Controller. Afterwards, the data direction identifier (CID0) is checked. If this CID0 is set, the CAN frame is a receive frame for the SLIO device and the process is going. Then the data length code from the received frame is checked and only 2 byte messages will be allowed with the SLIO application. In the flow within the identifier logic, the receive identifiers are compared with the programmed identifiers during initialization phase. When the identifiers during start up process are read over the EEPROM, two different receive identifiers are allowed, the receive ID and the global receive ID. Global ID's can be used to communicate with more than one SLIO node at the same time by one CAN message.

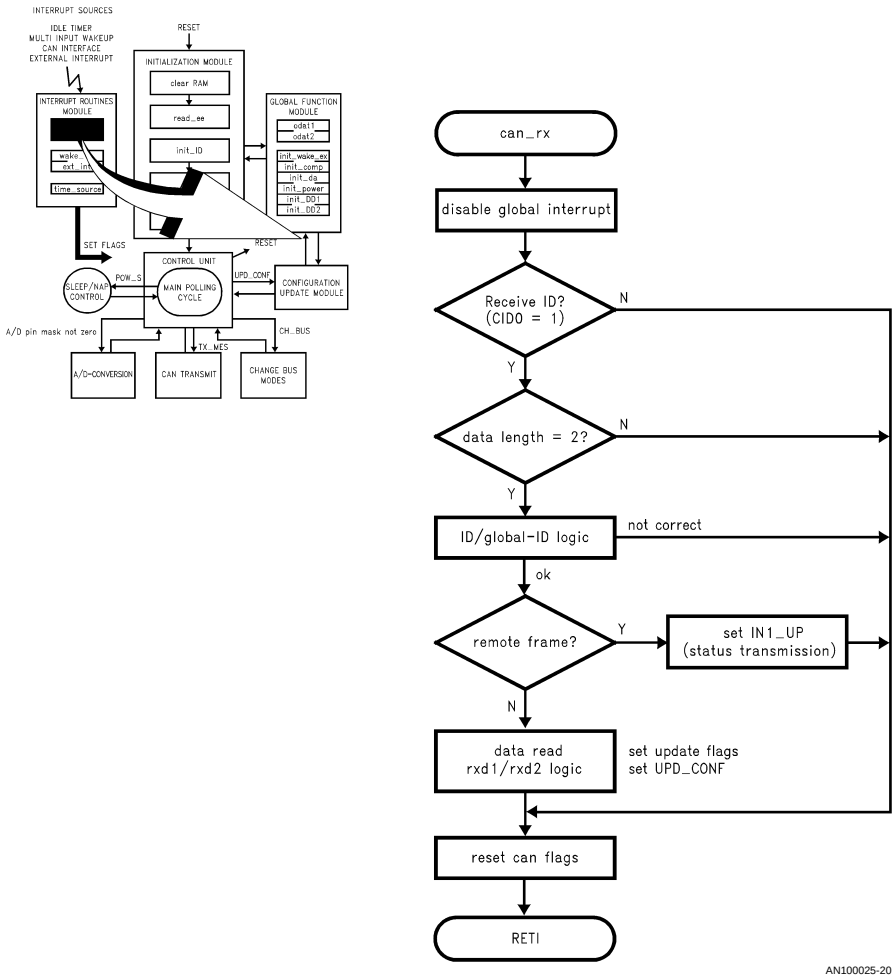


FIGURE 20. Flow of can_rx

Next, the remote bit of the CAN frame is checked. If the received message is a remote frame, the device will answer with a status message. In this case the read instruction of P0 to P7 will be prepared.

Then the main task of can_rx is started. This means on the one hand the decoding of the register marker, which is located in the first data byte of the CAN frame. The result is saved in specified update flags in the registers UPDAT_ST1 and UPDAT_ST2, which can be considered as the interface between CAN frame and configuration update process. On the other hand, the second data byte is saved in the specified SLIO Register if the received message does not belong to a “Read” instruction.

After this decoding logic, all Control flags of the CAN interface are cleared and the interrupt routine is finished.

3.2.4 Configuration Update Module

The important interface between the can_rx interrupt routine and the update module consists on the update flag registers UPDAT_ST1/UPDAT_ST2. One exception is the Analog Inputs, which are decoded through a pin_mask register and are executed from the Control Unit directly. The Analog Input module is described in the next section in detail.

Depending on the status of the update flags on the one hand the specified configuration routine is executed or on the other hand through “Read instructions” the specified read routine is started. In this context only, one of these update flags is set, because only one message is received at the same time and during the communication process, all other possible messages are ignored. *Figure 21* shows the flow of update.

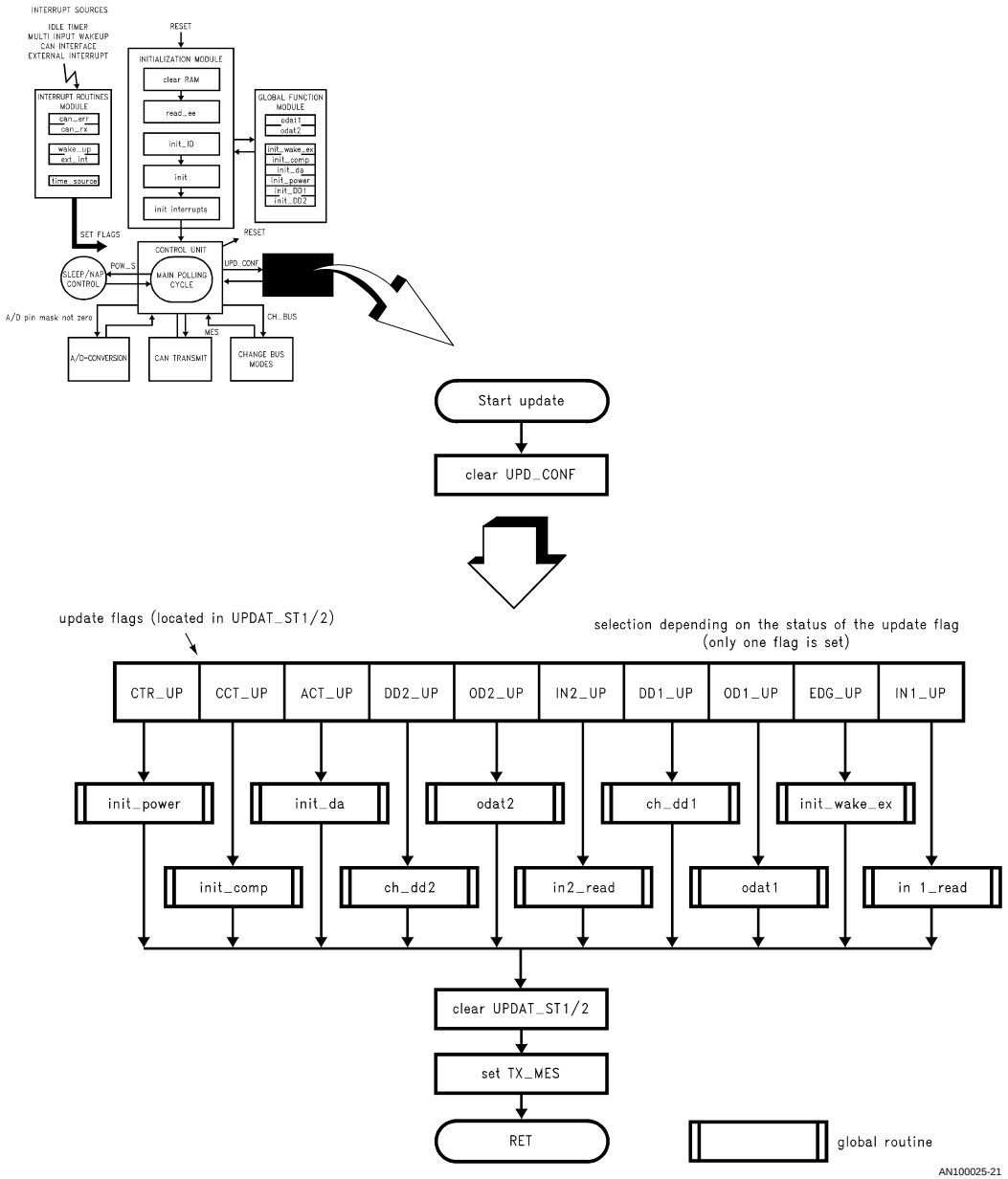


FIGURE 21. Flow of Update

3.2.5 Analog Input Module

When the can_rx routine recognizes one of the ADC Register Markers (RM 08h to 0Eh point to P0 to P6), the process does not take the usual way through the update module, because there is no hardware A/D periphery block on the microcontroller.

This conversion is done by software only. The configuration of the specified I/O pin is changed during the conversion process.

Therefore this A/D module will be executed directly over the Control Unit by the pin mask register, which selects the specified pin. The pin_mask register is configured in can_rx

depending on the ADC register marker. If this pin_mask register in not zero the process of adconv is started (see *Figure 22*).

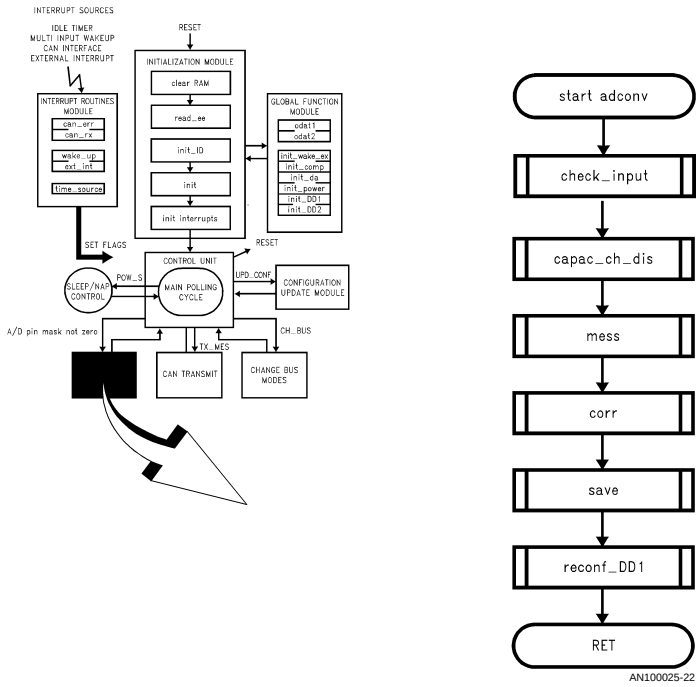


FIGURE 22. Flow of adconv

- **check_input** — At first in check_input, the specified pin is configured as input and the digital status after the Schmitt Trigger input is tested. The Schmitt Trigger range lies close to a voltage level of 2.2V. The digital status is saved with the STAT_AD control flag.

  STAT_AD = low →voltage level under Schmitt Trigger range

  STAT_AD = high →voltage level above Schmitt Trigger range
- **capac_ch_dis** — At first in this routine, the specified A/D pin is configured as output. After that the pin status of this pin is changed depending on the result of check_input. If the condition was logical high, the pin will be set to zero for a short time (nearly 15 tc). To the contrary, if the condition of the pin was logical low, the pin will be set to high. This is done with an internal driver of the port pin.
- **mess** — After the configuration of the A/D pin as input again, the time will be measured by the software counter register counter1 until the original status of the pin is executed again. This counter value can be considered as a proportional value of the analog voltage level. One problem lies in the fact that the current that is charging or discharging the external capacitor is not linear and so the counter values also are not linear (see also Section 3.3.2.1. Analog Input Module).
- **corr** — In this subroutine, the counter1 result is prepared to distinguish the voltage ranges above and under the Schmitt Trigger level, because all 8 bits from counter1 are used for measurement. Therefore, the counter1 is shifted to the right for one bit and the MSB of the new counter value is a sign for voltage levels under and above the Schmitt Trigger level. This process deletes the LSB of the counter value.

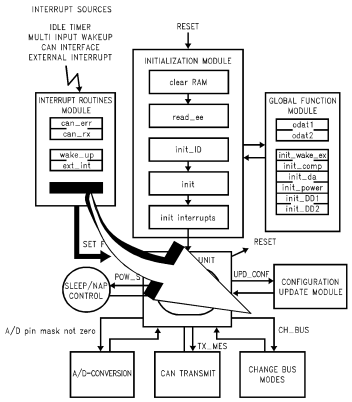
  voltage level above Schmitt Trigger range → MSB = one

  voltage level above Schmitt Trigger range → MSB = zero
- **save** — This subroutine prepares the correct counter value for transmitting in the CAN transmit module.
- **reconv_DD1** — The adconv routine is finished by reconfiguration of the data direction the data direction register DD1 by reconf_DD1. The reason for that reconfiguration is the insurance that the data direction can only be changed by CAN data direction messages.

3.2.6 Time Source Interrupt Routine

This routine is started every Bt, when the device is not in power_save condition or in CAN communication process. In these cases, the interrupt of the time_source is suspended. After starting (see Flow of time_source — *Figure 23*) the power_save_nap logic, it is decided if POW_S will be set. This means that if NAP mode is enabled, the program flow will enter in to power save control as soon as the control unit can do so. The question, how long the program will stay in NAP mode, will be decided in the power_save routine itself.

Afterwards, if automatic or cycled bus mode is enabled, it is decided depending on the programmable time (multiple of Bt) whether to change the mode (set CH_BUS). Then the interrupt routine is finished and the idle timer interrupt pending flag will be cleared.



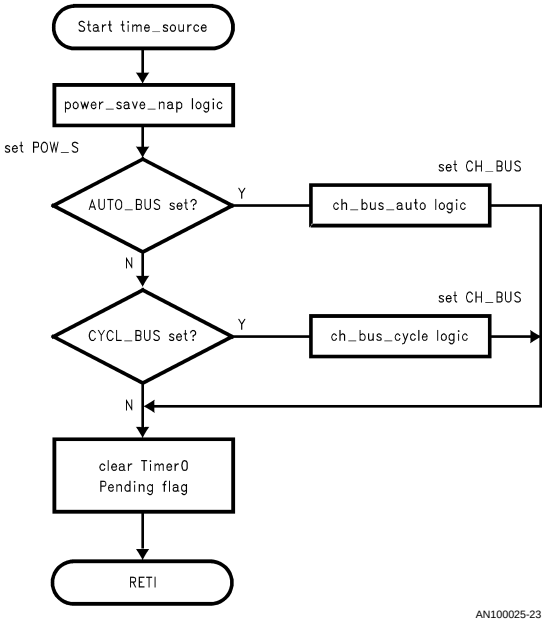


FIGURE 23. Flow of time_source

3.2.7 Power Save Routine

Power_save is executed from the control unit when POW_S is set. Of interest in this routine is the NAP logic. Within this logic, the pending flag of the time_source (IDLE timer) is counted and depending on the NAP programming time in SLIO CTR register, the process will wake up or not. After awakening from SLEEP or NAP mode a status message is prepared for transmitting. After transmittion of the status message in can_tx, all interrupts are enabled again.

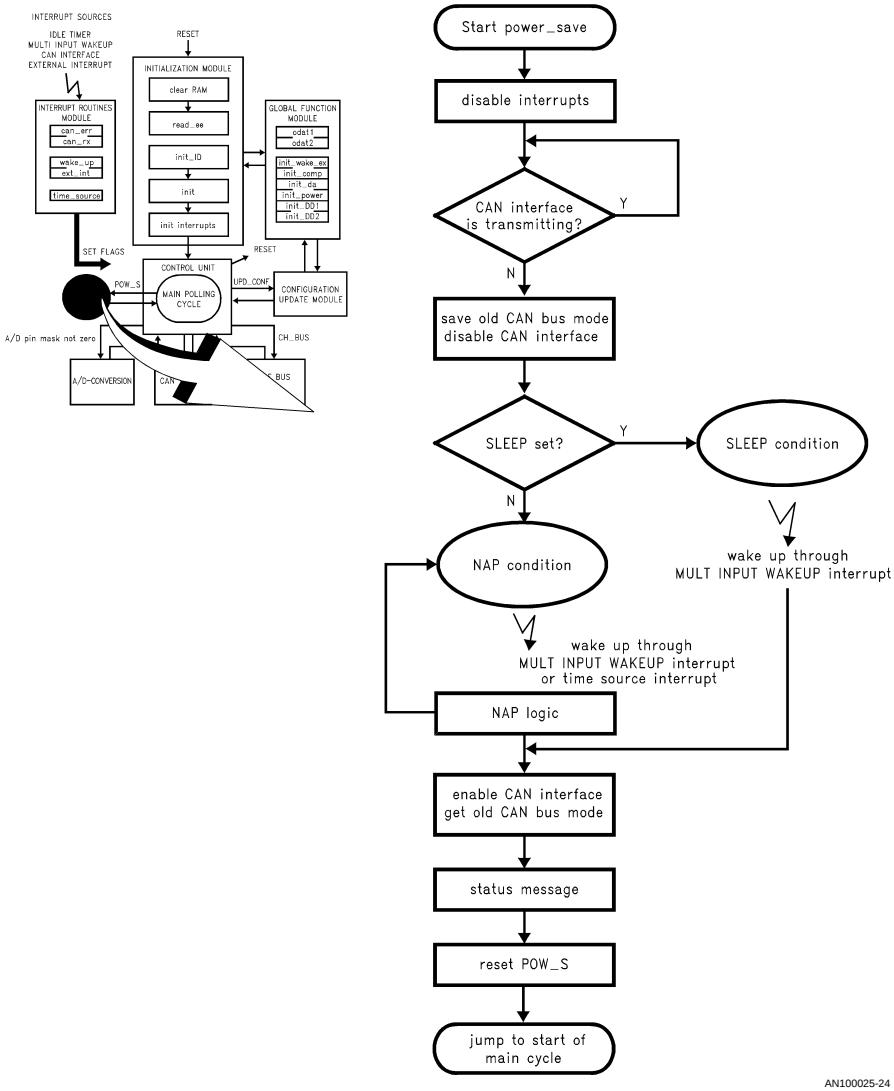


FIGURE 24. Flow of power_save

3.2.8 Change Bus Mode Routine

If the control flag CH_BUS is set, this routine will be executed. At first the WATCHDOG logic proves how may bus mode changes are being done. When in every bus mode no messages are received, the WATDOG_RES control flag is set. This is only important if WATCHDOG is enabled in the CTR register.

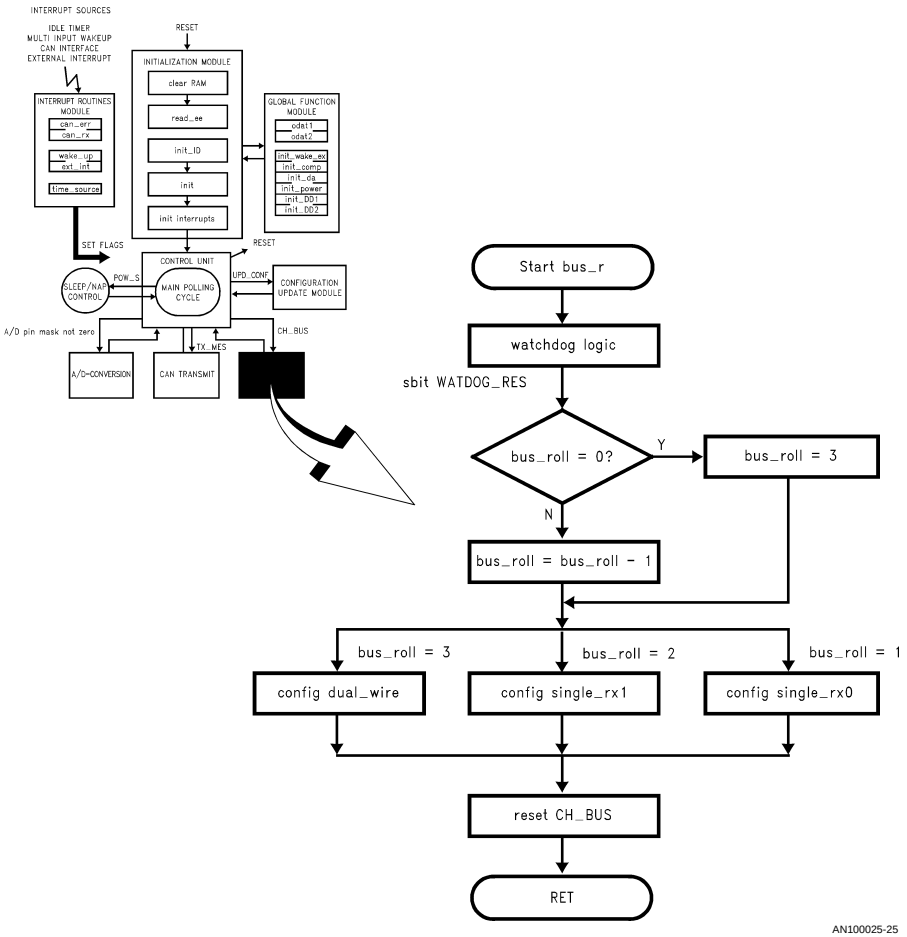


FIGURE 25. Flow of bus_r

After WATCHDOG logic the bus_roll user register controls the bus change process. In the further course of bus_r, one of the configuration routines changes the current bus state. After that the CH_BUS control flag will be cleared.

4.0 USAGE OF THE SLIO

4.1 SLIO Registers

In the SLIO module concept the SLIO memory contains several parameter defined registers (see *Table 3*), which can be configured by messages sent over the CAN bus or by an external EEPROM during the initialization phase. The configuration of the Multi-I/O Function Block, the power save conditions and the bus mode can be set and altered by the parameters in the Register Block. The Identifiers and the CAN prescaler are configured via the Identifier port.

TABLE 3. SLIO Register Block

Register Marker (hex)	Name	Function	Message type	config. over CAN	config over EEPROM
0x00	IN1	read status P0 to P7	r	read only	no
0x01	PE	config P0 to P7 positive edge	r/w	yes	yes
0x02	NE	config P0 to P7 negative edge	r/w	yes	yes
0x03	OD1	write data to P0 to P7	r/w	yes	yes
0x04	DD1	config P0 to P7 data direction	r/w	yes	yes

TABLE 3. SLIO Register Block (Continued)

Register Marker (hex)	Name	Function	Message type	config. over CAN	config over EEPROM
0x05	IN2	read status P8 to P13	r	read only	no
0x06	OD2	write data to P8 to P13	r/w	yes	yes
0x07	DD2	config P8 to P13 data direction	r/w	yes	yes
0x08 to 0x0E	ADC	read specified analog input	r	read only	no
0x0F	IN1	reset status register marker point to IN1	r	read only	no
0x10 to 0x13	DAC1/DAC2	config analog output	r/w	yes	yes
0x1C	ACT	analog output control	r/w	yes	yes
0x1E	CCT	comparator control	r/w	yes	yes
0x1F	CTR	configuration register	r/w	yes	yes

4.2 CAN Message Format

CAN messages to and from SLIO are limited to two byte messages. The first databyte is reserved for the register marker and system information. The register marker can be considered as a pointer of the specialized SLIO register, which should be changed through the data of the second databyte. The upper 3 bits of the first databyte include information about the bus mode of the SLIO and give information about the CAN Error status of the SLIO. The content of the two data bytes and from the control field is shown in *Table 4*.

TABLE 4. SLIO Frame Format

CNTRL		data byte 1							data byte 2
DLC = 2	ST	BM		RM					data
		1	0	4	3	2	1	0	

ST CAN Error Status of the SLIO

0 = error active

1 = the device became error passive since the last frame transmitted by the SLIO.

BM[1:0] Current Bus mode

0 = dual wire

1 = single wire RX0

2 = single wire RX1

3 = not allowed

RM[4:0] Register marker bits

Example: Status message

Status messages are created from SLIO without any demand messages from CAN. These messages are transmitted after the following actions:

1. initialization is finished
2. external event on the pins P0 to P7 (if they are enabled by PE or NE register)
3. awakening from NAP/SLEEP mode

This message contains the status of the pins P0 to P7. The content of *Figure 26* below describes a status message.

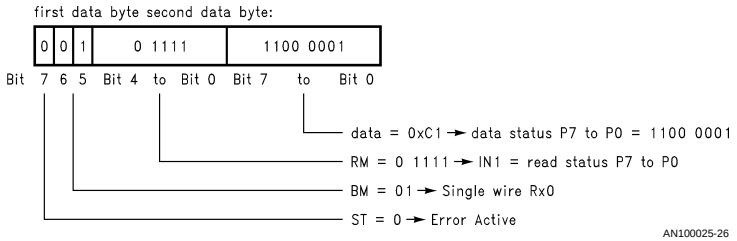


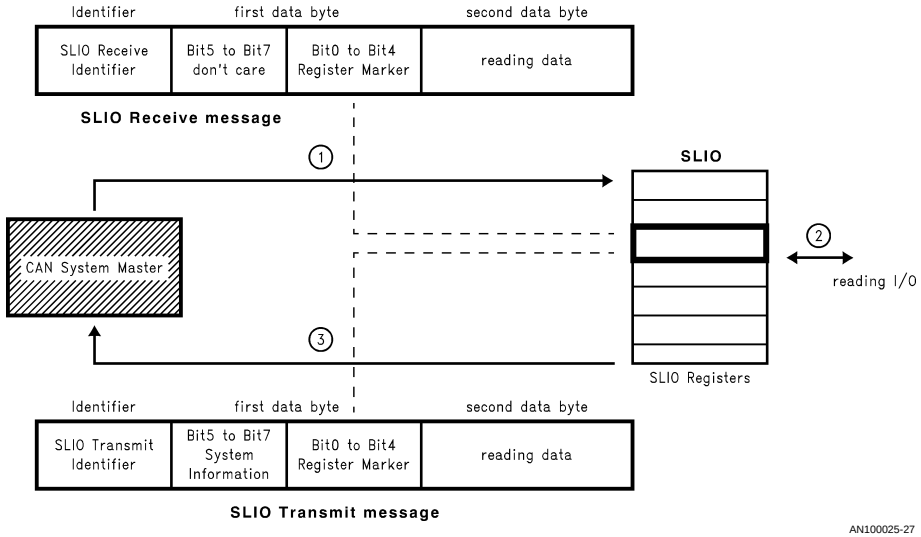
FIGURE 26. status message

4.3 CAN System Master Communication

Communication between the SLIO and the CAN System Master is achieved through query and response in addition to those messages which are initiated from the SLIO as a result of interrupts and wake-up conditions. There are two different message types, read only and read/write.

4.3.1 Read Message Transfer

Read Only messages demand the status of digital and analog pins (see also *Table 3* SLIO Register Block). *Figure 27* shows the data transfer of read only messages between CAN System Master and the SLIO device.



The following steps are executed:

1. SLIO receives a read message from System Master with correct Receive Identifier and with the correct Read register marker.
2. The SLIO reads the port pins or the analog input pin and writes the reading data in the specified SLIO Register.
3. After the reading process, the SLIO creates a message with its transmit ID and the reading data.

FIGURE 27. CAN Communication with Read Messages

Example: Read Pin P0 to P7

This example describes reading the digital status of pin P0 to P7 over CAN by the following configuration. The SLIO is configured in Single Wire Rx0/TX0 Bus mode and the error state of the SLIO is Error Active. The Identifiers are configured in Pin Mode as following:

ID0 = GND; ID1 = GND; ID2 = GND; ID3 = GND

ID configuration: Transmit ID = 0400 H (from the SLIO)

Receive ID = 0401 H (to the SLIO)

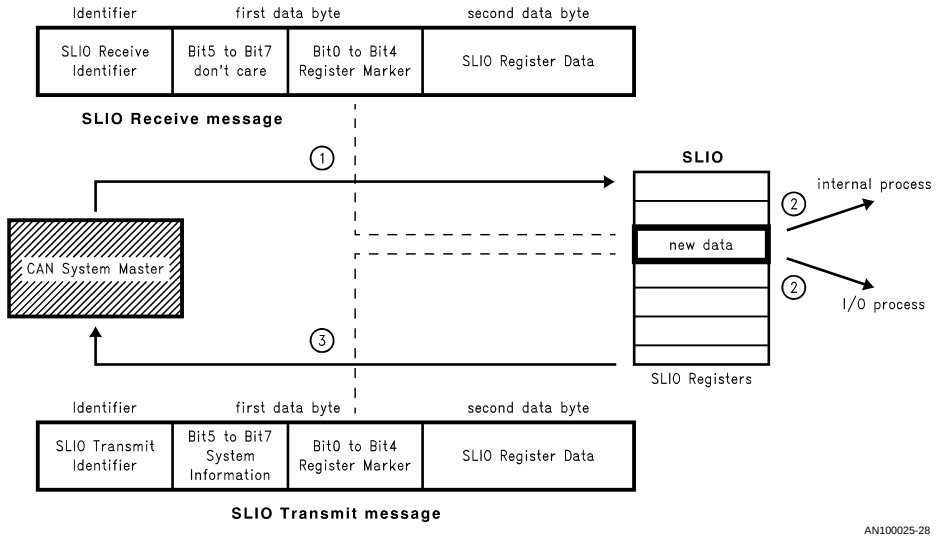
The data of the SLIO pins are 0xC0. In the following *Table 5* the data transfer between CAN System Master and the SLIO is monitored.

TABLE 5. Read Transfer Example

Name	ID	databyte1	databyte2
receive message	0401 H	00 H	don't care
transmit message	0400 H	20 H	C0 H

4.3.2 Read/Write Message Transfer

Read/write transfer updates the configuration data within the SLIO register block by writing data into the specified register as indicated by the register marker in the first data byte. The response is a read from the specified data register subsequent to the update process. The data transfer is shown schematically in *Figure 28*.



The following steps are executed:

1. The SLIO receives a read/write message from System Master with the correct Receive Identifier and with the correct Read register marker and the new data placed in the second databyte
2. The SLIO changes the configuration of the internal condition or, optionally, of the I/O pin configuration.
3. After this process the SLIO creates a response message with its transmit ID and the new status of the specified SLIO register. This informs the System Master that the message data transfer was executed correctly.

FIGURE 28. CAN Communication with Read/Write Messages

Example: Configuration of Data Direction Pin P0 to P7

The data direction of the SLIO pins P0 to P7 should be changed via CAN. It will be assumed that the configuration of the Identifier and Bus mode is the same as in the Read Only example.

The data direction of the pins should be changed as follows:

P0 to P3 as input

P4 to P7 as output

In *Table 6* the read/write data transfer between CAN System Master and SLIO is monitored.

TABLE 6. Read/Write Date Transfer Example

Name	ID	databyte1	databyte2
receive message	0401 H	00 H	F0 H
transmit message	0400 H	20 H	F0 H

4.4 CAN System Master in CAN — SLIO Network

In the CAN network at least one node is assigned as the CAN System Master. The system master handles the communication with the connected SLIO nodes. Each of these nodes must have two sequential IDs, an even one for transmit and an odd one for receive. In addition, when using the EEPROM mode, a global receive ID may be defined.

An example of one CAN-message address space is shown in the following Example (*Figure 29*).

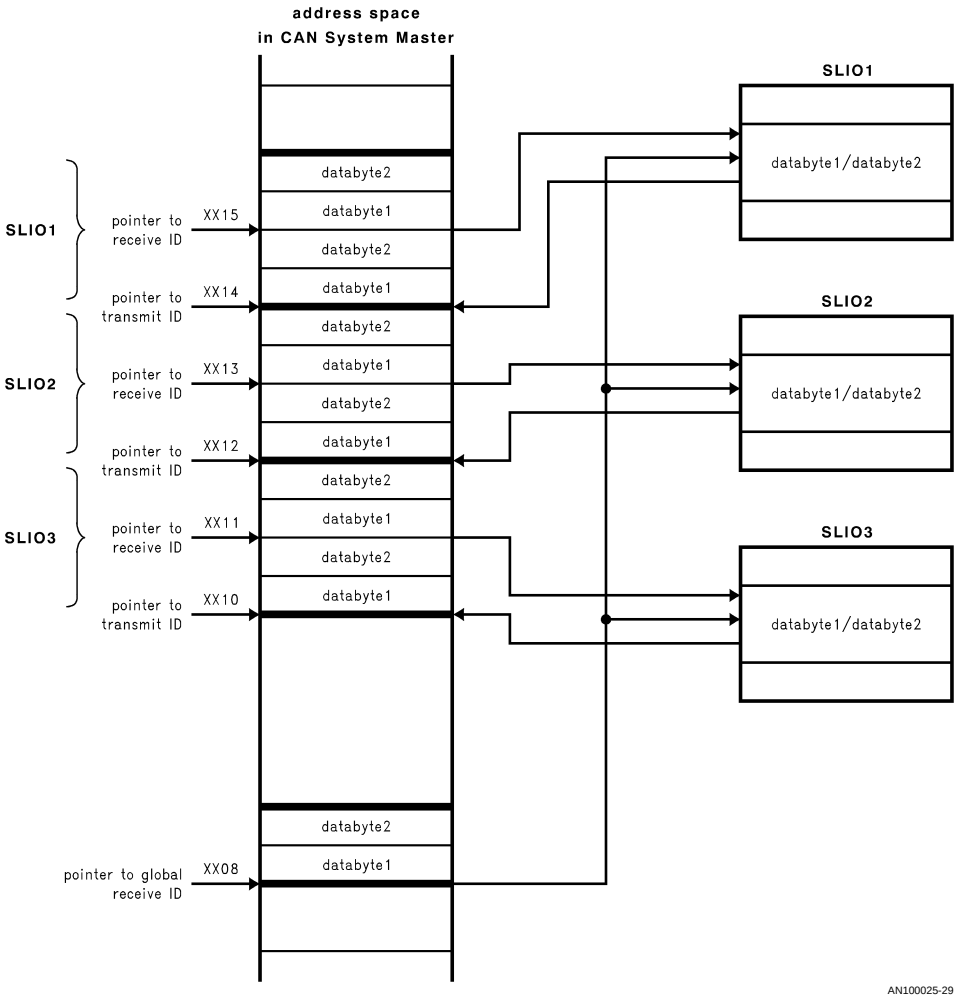


FIGURE 29. Example — CAN System Master Address Space

4.5 SLIO System

Figure 30 shows the schematics of a CAN SLIO node. The pins P0 to P13 can be connected with Sensors or Actuators over the I/O Feature Connector. The Power Supply circuit with LM2925 generates firstly V_{CC} = 5V and secondly the external RESET. V_{IN} can be obtained from an external source of 6V < V_{IN} < 26V, or from the CAN wiring system. Because of the external crystal oscillator on the pins CKI and CKO and SLIO does not need synchronization messages from the System Master. This device has Master capabilities, which means that it can synchronize itself to the CAN bus.

The two signal wires of CAN, BUS_H and BUS_L, are connected with the integrated CAN interface of the SLIO over the Physical Bus interface. The bus timing programmability of the SLIO CAN interface is limited, with the exception of the CAN prescaler and CKI. Refer to Section 1 for circuit description of the interfaces.

During the initialization phase, the SLIO application reads the identifiers from the identifier circuit over the pins ID0 to ID3. There are two different capabilities to read the identifier. Two means of determining the identifier exist; direct pull up/down of the ID pins or via the EEPROM.

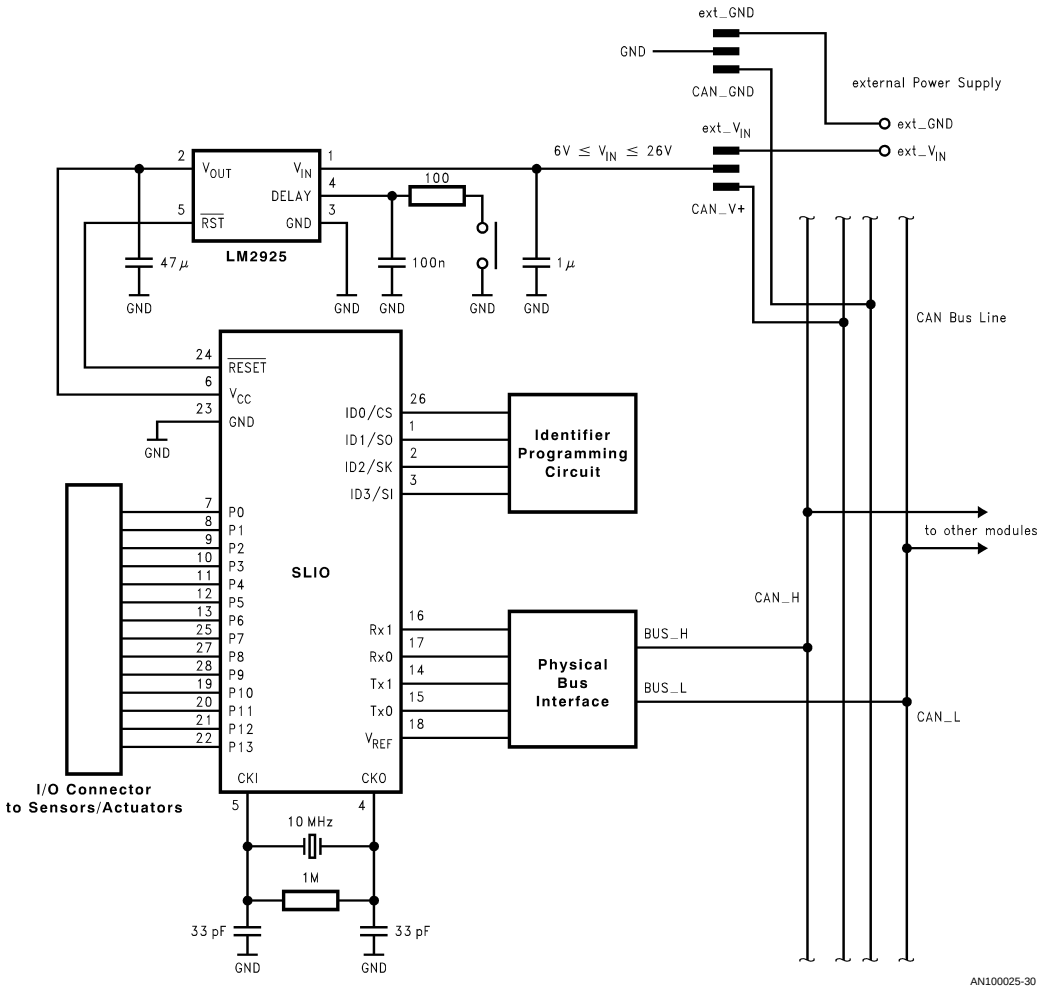


FIGURE 30. SLIO CAN Node

4.6 Initialization

4.6.1 General Description

There are two different modes of reading the CAN identifier. In Pin mode four of the 11 available identifier bits may be read from ID0-3, the remainder of the bits are predetermined, thus 16 different SLIO nodes may be connected to one CAN network system. Two avoid this restriction an EEPROM (NMC93C06) mode may be used on the ID port allowing programming of 11 of the identifier bits. This means that the ID port is internally connected to the MICROWIRE interface. Furthermore the use of an EEPROM allows the programming of the Register Block and avoids the necessity of having the system master transmit the setup configuration when the node is brought on line.

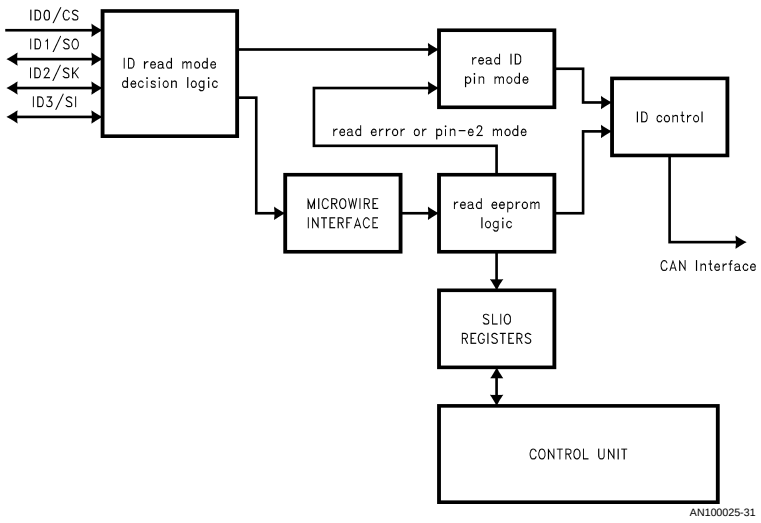


FIGURE 31. Read CAN Identifier Port

A mixture between the EEPROM mode and the pin mode was implemented as Pin_e2 mode. In this Pin_e2 mode the identifier are read in pin mode and the configuration of the SLIO registers can be read from the external EEPROM. In this case the information to go in Pin_e2 mode is given from the EEPROM (*Figure 31*). At the end of the initialization phase the SLIO will transmit a status message. After this message the SLIO is ready to communicate with the CAN bus.

4.6.2 Initialization Example in Pin Mode

In this section, an example to initialize the SLIO in Pin Mode is shown. Initialization means on the one hand to program the CAN Identifier and on the other hand the configuration of the SLIO Registers.

Example: CAN Identifier Programming — Pin Mode

In Pin Mode the CAN Identifier is programmed through pull up/down resistors on the Identifier Port pins ID0 to ID3. This means that 4 CAN Identifiers can be programmed and so 16 different SLIO Nodes can be connected on the CAN bus. In *Figure 32* a connection example is shown.

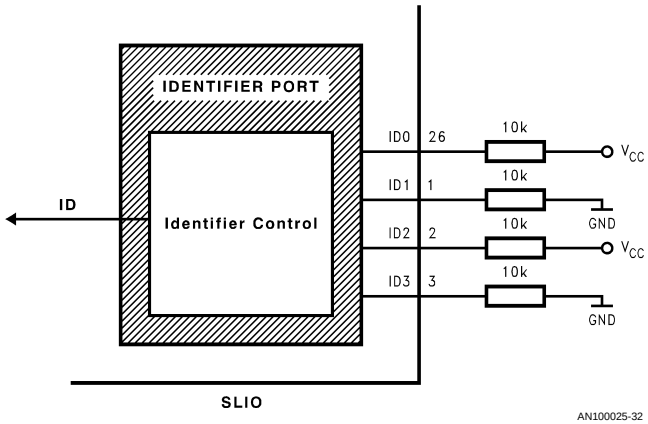


FIGURE 32. Identifier Configuration in Pin Mode

The result of this programming is shown in *Table 7*, columns ID3, ID2, ID1 and ID0.

TABLE 7. SLIO CAN Identifiers in Pin Mode

ID-Name	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	DIR
transmit ID (0x414 h)	1	0	0	0	0	0	1	0	1	0	0
receive ID (0x415 h)	1	0	0	0	0	0	1	0	1	0	1

In Pin Mode the SLIO Registers are fixed to default values. They can not be configured during initialization phase. The bus rate is fixed to CKI/40 and the bus mode is automatic. That means that the device cycles through all bus modes if no message is received for 8*Bt.

In *Table 8*, the default SLIO Register values are shown.

TABLE 8. Configuration of the SLIO Registers in Pin Mode

Name	Function	configured after Initialization (hex)
IN1	read status P0 to P7	can not be configured
PE	config P0 to P7 positive edge	0x00
NE	config P0 to P7 negative edge	0x00
OD1	write data to P0 to P7	0xFF
DD1	config P0 to P7 data direction	0x00
IN2	read status P8 to P13	can not be configured
OD2	write data to P8 to P13	0xFF
DD2	config P8 to P13 data direction	0x00
ADC	read specified analog input	can not be configured
DAC1	config analog output	0x00
DAC2	config analog output	0x00
ACT	analog output control	0x00
CCT	comparator control	0x00
CTR	configuration register	0x00

4.6.3 Initialization Example in EEPROM Mode

In EEPROM mode, all bits of the CAN standard identifier are programmable and each of the SLIO registers, as well as the CAN prescaler register, may be configured separately. Prior to reading the EEPROM, the CS (ID0) pin must be held low to prevent interference with any other MICROWIRE users available to the node. If an EEPROM is connected to the SLIO for purposes of programming the identifier and registers, the first location must read an AA hex value. If the value is other than AA hex, the device assumes the EEPROM is for purposes other than of programming the SLIO. When AA is detected in the EEPROM location E2-MASK, the data contained in the EEPROM is transferred to the internal registers of the SLIO.

Example: CAN Identifier Programming — EEPROM Mode

In *Figure 33* the connection between EEPROM NMC93C06 and Identifier port are shown.

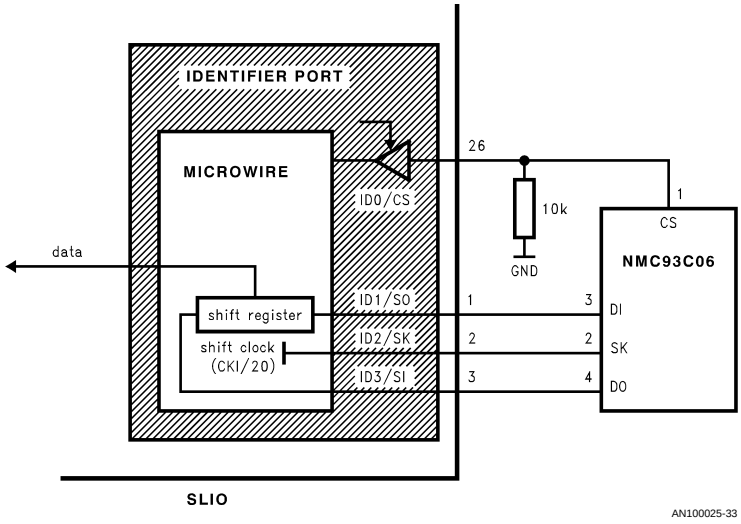


FIGURE 33. EEPROM Connection

The programming of the identifier ID0 to ID6 will be done by the EEPROM register RXIDL. DIR (Bit0) of this register can not be configured, it is don't care. The data direction will be configured automatically. ID7 to ID9 are configured with the EEPROM register RXIDH. A configuration example of the identifier through RXIDH/RXIDL is shown in *Table 9*.

TABLE 9. Receive/Transmit Identifier Programming with EEPROM

CID10	CID9	CID8	CID7	CID6	CID5	CID4	CID3	CID2	CID1	CID0
ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	DIR
1	1	1	0	0	0	0	1	0	1	—
RXIDH			RXIDL							

The result of programming receive and transmit Identifiers which are programmed by RXIDH and RXIDL, are shown below:

Transmit ID → 0x70AReceive ID → 0x70B

The programming of the global identifier ID0 to ID6 will be done by the EEPROM register RXIDGL. DIR (Bit0) of this register is set to 1 (receive data direction) automatically. ID7 to ID9 are configured with the EEPROM register RXIDGH. A configuration example is shown in *Table 10*.

TABLE 10. Global Identifier Programming

CID10	CID9	CID8	CID7	CID6	CID5	CID4	CID3	CID2	CID1	CID0
ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	DIR
1	1	1	0	0	0	0	0	0	0	—
RXIDGH			RXIDGL							

The result of programming global receive Identifiers, which are programmed by RXIDGH and RXIDGL, are shown below:

Global Receive ID → 0x701Transmit ID → 0x70A

In *Table 11* a configuration example of the external EEPROM register settings are shown.

TABLE 11. Example of Configuration of SLIO Registers in EEPROM Mode

E2-address	EEPROM Registers	SLIO Registers
0x00	**E2-MASK** 0xAA	—
	PIN-E2-MASK 0x00	—
0x01 0x02	**RXIDH/RXIDL RXIDGH/RXIDGL**	
0x03	**PEDGE** 0xF0	**PE** 0xF0
	NEDGE 0x0B	NE 0x0B
0x04	**ODATA1** 0x01	**OD1** 0x01
	ODATA2 0x00	**OD2** 0x00
0x05	**DATADIR1** 0x00	**DD1** 0x00
	DATADIR2 0x00	**DD2** 0x00
0x06	**DACH** 0x01	**DAC2** 0x01
	DACL 0xB0	**DAC1** 0xB0
0x07	**ACR** 0x03	**ACT** 0x03
	CCR 0xE0	**CCT** 0xE0
0x08	**DCR** 0x08	**CTR** 0x08
	CAN_PSC 0x03	**CAN Prescaler** 0x03

4.6.4 Initialization Example in Pin_e2 Mode

If the e2 register PIN_e2 MASK is programmed with 0x55, the Pin_e2 mode is enabled. This allows the reading of the SLIO default values from the EEPROM and the Identifiers ID1 to ID3 in Pin mode. The pin ID0/CS can not be used for Identifier programming because this pin needs a pull down resistor for the reading process of the EEPROM. Therefore in Pin_e2 mode, only eight different Identifiers can be configured.

Example: Initialization in Pin — e2 Mode

In *Figure 34* the connection between EEPROM NMC93C06, pull up/down resistors and Identifier port is are shown.

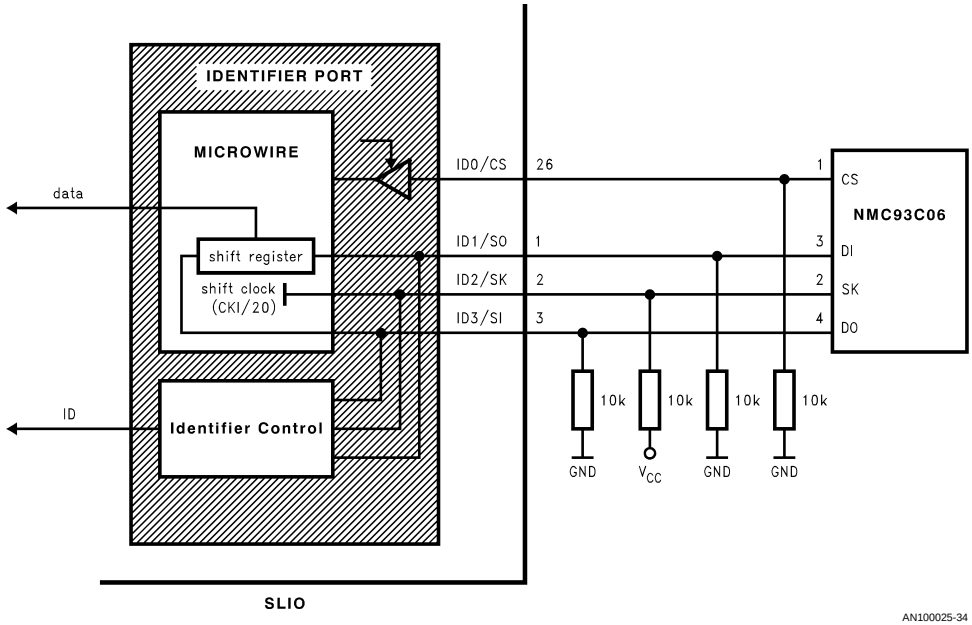


FIGURE 34. Pin_e2 Configuration

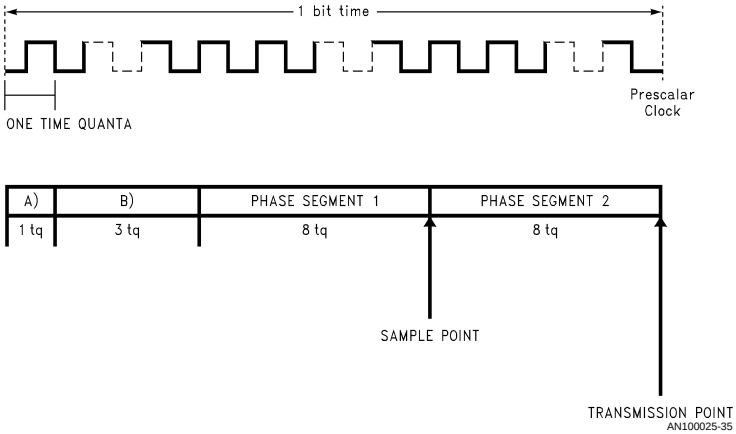
The result of this programming is shown in *Table 12*. Hereby the identifier ID0 always has low level. Therefore, in Pin_e2 mode, only ID1 to ID3 can be configured over pull up/down resistors. ID4 to ID9 can be configured over the EEPROM.

TABLE 12. SLIO CAN Identifiers in Pin_e2 Mode

ID-Name	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	DIR
transmit ID (0x408 h)	0	1	1	0	0	0	0	1	0	0	0
receive ID (0x409 h)	0	1	1	0	0	0	0	1	0	0	1
	configurable over EEPROM						configurable over resistors			fix to 0	

4.6.5 CAN Bus Rate Configuration

In EEPROM Mode and in Pin_e2 Mode, the bus rate of the SLIO can be configured by the EEPROM register CAN_PSC and CKI. In pin mode the bus rate is fixed to CKI/40. Furthermore, the segments of one bit time are predefined as described in *Figure 35*. This means that the sample point is fixed to 60% up to 500 kbit/s bus rate and to 80% using 1 Mbit/s. Hereby the synchronization jump width is configured to 4 time quanta up to 500 kbit/s and 2 time quanta using 1 Mbit/s.



A) synchronization segment
B) propagation segment

FIGURE 35. Bit Timing Up to 500 kbit/s

Example: bus time configuration — EEPROM Mode/ Pin_e2 Mode

If EEPROM Mode/Pin_e2 Mode is used, the bus rate can be configured with the CAN_PSC register during initialization phase (see NM93C06 memory map datasheet).

Configuration formula: bus rate = CKI/(10 * (CAN_PSC + 1))

In the following *Table 13* some examples of initialization are shown.

TABLE 13. Examples Bus Rate (CKI = 10 MHz)

CAN_PSC (hex)	Bus Rate (kbits/s)
01	500
03	250
04	200
07	125
09	100
19	50

4.7 Usage of Analog Input

The analog input is not intended to be a high performance A/D-conversion, but provides the capability of reading up to 16 different voltage levels with any of seven I/O pins. *Figure 36* shows an example by reading different voltage levels of a resistor array. This will be done by measuring the charge or discharge time of an external capacitor. The internal construction of an I/O pin (see *Figure 36*) will support the analog input. At first the level of the Schmitt Trigger Input is measured. Depending on the result, low or high, the internal driver, which is controlled by the charge/discharge logic, charges or discharges the external capacitor.

Schmitt Trigger level = low → charge capacity

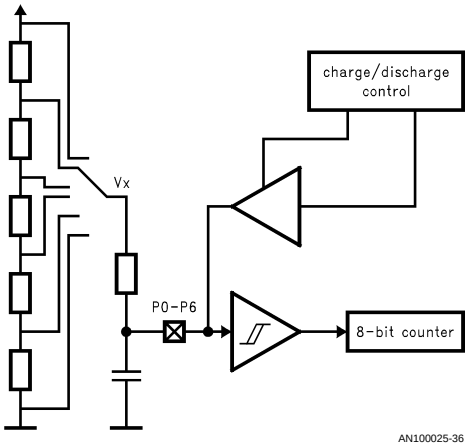


FIGURE 36. Analog Input

Schmitt Trigger level = high → discharge capacity

The charge/discharge control is then disabled and the time to get the original digital (after Schmitt Trigger) state is measured by a counter register. These counter values consider different input voltages.

Example: Read 16 different voltages on pin P0 using R/C

The restriction of this A/D conversion is shown in *Figure 37*, because the charge or discharge time of the capacitor is dependent on the current and this current is not linear. Especially with voltages near the Schmitt Trigger level, the 8-bit counter value is overflowed and no measurement is possible. This measurement is dependent upon the CPU speed, hence the R/C values may have to be adjusted to accommodate a change in CKI value from 10 MHz. The external components, which are connected to the pin P0, are shown in *Figure 38*.

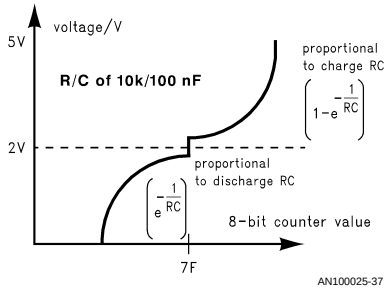


FIGURE 37. Input Voltage Depending on Counter Value

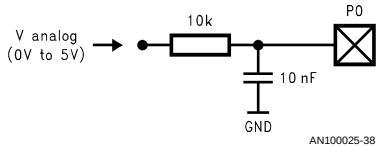


FIGURE 38. Example of Analog Input Components

Before the analog input Register marker can be executed, the pin has to be configured as High-Z input. This means that DD1 and OD1 have to be configured to low for the pin P0. The following CAN frame examples assume that the SLIO is configured to SINGLE WIRE RX0 bus mode, the error condition is error active and the receive ID = 0021. The data frames for the P0 configuration are shown below in *Table 14*. The pin configuration frames have to be transferred one time only.

TABLE 14. Read/Write Data Transfer Example

Direction	Name	ID	first databyte	second databyte
→	conf SINGLE WIRE RX0	0021 H	1F H	08 H
←	answer from SLIO	0020 H	3F H	08 H
→	OD1 to 11110000	0021 H	03 H	F0 H
←	answer from SLIO	0020 H	23 H	F0 H
→	DD1 to 11110000	0021 H	04 H	F0 H
←	answer from SLIO	0020 H	24 H	F0 H
→	analog input from P0	0021 H	08 H	XX H
←	answer from SLIO	0020 H	28 H	13 H (0.0V)

The values of the 16 different values are shown in *Table 15*.

TABLE 15. Reading of 16 Different Analog Voltages

Voltage Input (V)	Counter Value (hex)	Counter Range (±4 Counter Steps)	Range Number
0.0	13	0F to 17	0
1.3	23	1F to 27	1
1.5	2D	29 to 31	2
1.8	3D	39 to 41	3
1.9	46	42 to 4A	4
2.0	53	(±8) 4B to 5B	5
2.1	6C	(±8) 64 to 74	6
2.5	7F	(±8) 77 to 87	7
2.9	90	(±8) 88 to 98	8
3.0	9D	99 to A1	9
3.1	AB	A7 to AF	A
3.3	B7	B3 to BB	B
3.5	C1	BD to C5	C
3.8	CB	C7 to CF	D
4.2	D4	C1 to D8	E
5.0	DF	DB to E3	F

The different ranges of the example in *Table 15* are shown in *Figure 39*.

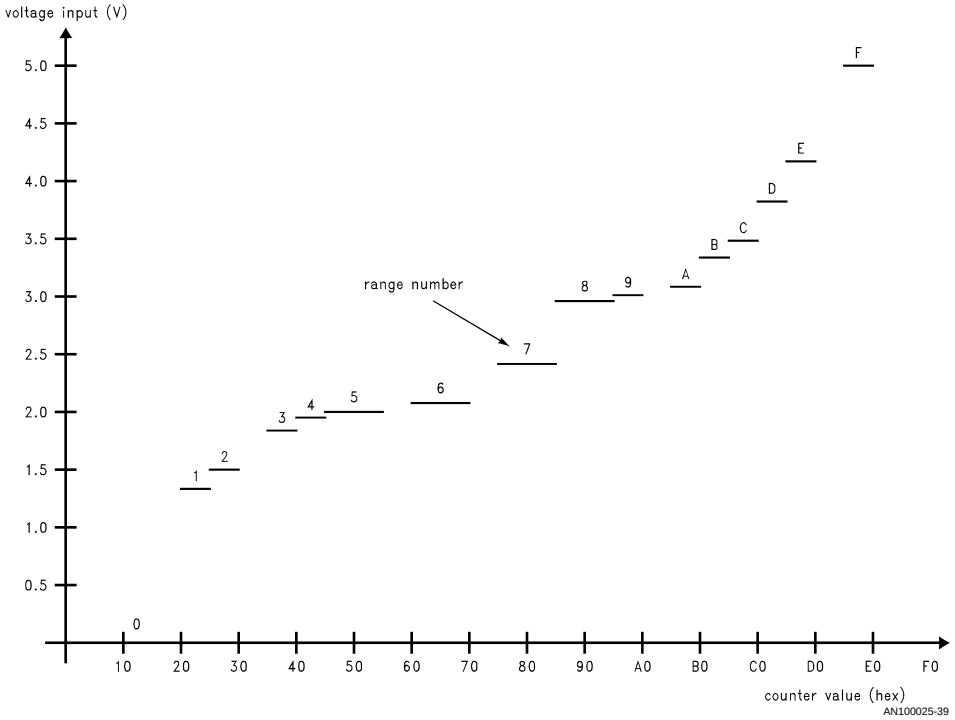


FIGURE 39. Graph of the Different Ranges

4.8 Usage of D/A Output

A user programmable PWM signal is provided on pin P9. This signal may be configured to either a 10-bit or 8-bit resolution. This PWM signal is CKI dependent. For example, by using CKI = 10 MHz one PWM cycle is 255 µs (8-bit) or 1023 µs (10-bit). In order to calculate the cycle time of the PWM use the following formula.

$$T_{pwm} = \frac{10 \times (2^{resolution} - 1)}{CKI}$$

AN100025-47

By using an external low pass filter analog voltages can be generated. An example of the RC is shown in *Figure 40*. The analog output will be configured with the SLIO registers DAC1, DAC2 and ACT.

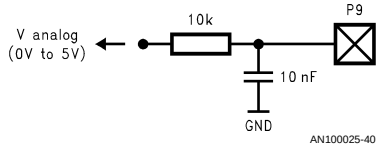


FIGURE 40. Example of Analog Output Components

To generate PWM signals on P9 the following steps have to be executed:

- configure P9 as output (over e2 or over CAN)
- configure High/Low Time of the PWM signal with the registers DAC2 and DAC1 (over e2 or over CAN)
- configure 8-Bit or 10-Bit PWM signal with the DAR Bit of the register ACT (over e2 or over CAN)
- enable PWM output with DACEN of ACT (over e2 or over CAN)

10-Bit PWM Configuration

The configuration of the SLIO registers DAC2/DAC1 via CAN is shown in *Table 16*.

TABLE 16. 10-Bit D/A Output Examples

Register Marker					second databyte (hex)	10-Bit Format D/A	
Bit4	Bit3	Bit2	Bit1	Bit0		DAC2	DAC1
1	0	0	0	0	B0	00	B0
1	0	0	0	1	B0	01	B0
1	0	0	1	0	B0	02	B0
1	0	0	1	1	B0	03	B0

Example: 10-Bit PWM over CAN

Table 17 summarizes all messages which are necessary to configure pin P9 (as a ten-bit PWM output). It is assumed that the SLIO is configured in Single wire RX0 bus mode and the error mode is error active. The CKI is configured with 10 MHz. In *Figure 41* the PWM output resulting from the configuration of *Table 17* is shown.

TABLE 17. Data Transfer Example for 10-Bit D/A

Direction	Name	ID	first databyte	second databyte
→	conf SINGLE WIRE RX0	0401 H	1F H	08 H
←	answer from SLIO	0400 H	3F H	08 H
→	OD2 to 00000000	0401 H	06 H	00 H
←	answer from SLIO	0400 H	26 H	00 H
→	DD2 to 00000010	0401 H	07 H	02 H
←	answer from SLIO	0400 H	27 H	02 H
→	DAC2/DAC1 to 02 B0 H	0401 H	12 H	B0 H
←	answer from SLIO	0400 H	32 H	B0 H
→	ACT to 00000011	0401 H	1C H	03 H
←	answer from SLIO	0400 H	3C H	03 H

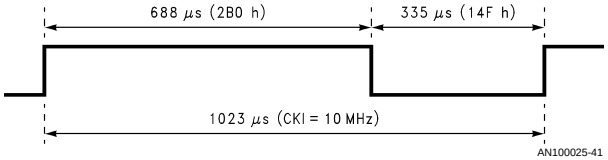


FIGURE 41. 10-Bit PWM Output

8-Bit PWM Configuration

The 8-Bit configuration of the SLIO registers DAC1 via CAN is shown in *Table 18*. In this case, Bit 0/Bit 1 of the register marker are don't care. This means that all register marker bits, which are reserved for DAC, can be used for 8-Bit PWM configuration.

TABLE 18. 8-Bit D/A Output Examples

Register Marker					second databyte (hex)	8-Bit Format D/A
Bit4	Bit3	Bit2	Bit1	Bit0		DAC1
1	0	0	x	x	B0	B0

Example: 8-Bit PWM Configuration

In *Table 19* all CAN messages are summarized that are necessary to configure pin P9 as 8-Bit PWM output. Hereby, it is assumed that the SLIO is configured in Single wire RX0 bus mode and the error mode is error active. Moreover, the CKI is configured with 10 MHz.

TABLE 19. Data Transfer Example for 8-Bit D/A

Direction	Name	ID	first databyte	second databyte
→	conf SINGLE WIRE RX0	0401 H	1F H	08 H
←	answer from SLIO	0400 H	3F H	08 H
→	OD2 to 00000000	0401 H	06 H	00 H
←	answer from SLIO	0400 H	26 H	00 H
→	DD2 to 00000010	0401 H	07 H	02 H
←	answer from SLIO	0400 H	27 H	02 H
→	DAC1 to 10110000	0401 H	12 H	B0 H
←	answer from SLIO	0400 H	32 H	B0 H
→	ACT to 00000010	0401 H	1C H	02 H
←	answer from SLIO	0400 H	3C H	02 H

4.9 Handling of External Events

Pins P0 to P7 provide monitoring of external events through detection of rising and/or falling edge transition. The configuration is done through the SLIO registers PE and NE. A one in a given bit of these registers enables the external event mode for the corresponding pin.

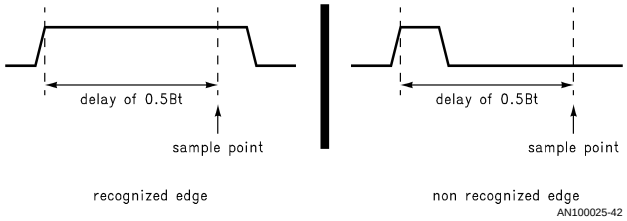
Example: configuration P0 — pos. edge and P1 — pos./ neg. edge

Table 20 depicts the configuration of the status of P0 — P7 via the CAN bus. Subsequent to this configuration a matching edge on the port will result in a transmission of P0 — P7 status on the bus from the SLIO. In order to eliminate the possibility of noise or switch bounce, the port is resampled after a time period of Bt. Note that this period is dependent

on the CPU clock frequency. If an event occurs during a bus transaction the reporting of the event will be delayed until the bus is clear.

TABLE 20. Configuration of PE and NE via CAN

Direction	Name	ID	first databyte	second databyte
→	conf SINGLE WIRE RX0	0401 H	1F H	08 H
←	answer from SLIO	0400 H	3F H	08 H
→	PE to 00000011	0401 H	01 H	03 H
←	answer from SLIO	0400 H	21 H	03 H
→	NE to 00000010	0401 H	02 H	02 H
←	answer from SLIO	0400 H	22 H	02 H



NOTE: 1 Bt = 40960/CKI

FIGURE 42. Delay Time External Rising Event

During the receive/transmit phase of the SLIO, the process caused through an event is delayed until CAN communication is finished.

4.10 Power Save Mode Examples

The SLIO device supports two different power save modes, SLEEP mode and NAP mode. SLEEP mode stops all activities and the clock. NAP mode stops all activities but the clock and an internal counter. This counter will wake-up the device every Bt time (*Figure 43*). The device will wake-up from both modes by an external signal being applied on one or more of the port pins P0 to P6, by a recessive to dominate transition on the CAN bus and by pulling the /RESET pin low. Waking-up triggers and automatic wakeup in NAP mode through the internal counter, cause the transmission of a status message. If the device wakes up from SLEEP mode, it will stay in active mode (*Figure 43*) and all previous settings of the registers are valid again.

The power mode bits PO0 to PO2 in the control register CTR, set up the power saving modes SLEEP and NAP. The different configurations are summarized in the *Table 21*.

TABLE 21. Power Mode Configurations (CTR Register)

PO2	PO1	PO0	Power Mode
0	0	0	active
0	0	1	NAP: 1 * Bt
0	1	0	NAP: 2 * Bt
0	1	1	NAP: 4 * Bt
1	0	0	NAP: 8 * Bt
1	0	1	NAP: 16 * Bt
1	1	0	NAP: 32 * Bt
1	1	1	SLEEP

Example: SLEEP Mode

The CAN messages, described in *Table 22*, enables the SLEEP mode.

TABLE 22. Configuration of SLEEP Mode

Direction	Name	ID	first databyte	second databyte
→	conf SINGLE WIRE RX0 and enable SLEEP mode	0401 H	1F H	E8 H
←	answer from SLIO	0400 H	3F H	E8 H

After this data transfer, the device enters SLEEP mode, all activities including the CKI clock are stopped. The SLIO will wake up on a rising/falling edge on any enabled pin PO - P6 or upon a recessive/dominant transition on the CAN bus. *Table 23* gives an example of a wake-up transaction from SLEEP mode over CAN.

TABLE 23. Example of Wake-Up SLEEP Mode

Direction	Name	ID	first databyte	second databyte
→	wakeup message (SOF = rec./dom. transition)	0401 H	xx H	xx H
←	status message answer	0400 H	20 H	00 H

After wake-up the clock is running and the SLIO will stay in active mode.

Example: configuration NAP mode — 16*Bt

The CAN message in the following *Table 24* enables the 16*Bt NAP mode.

TABLE 24. Configuration of 16*Bt NAP Mode

Direction	Name	ID	first databyte	second databyte
→	conf SINGLE WIRE RX0 and enable NAP mode	0401 H	1F H	A8 H
←	answer from SLIO	0400 H	3F H	A8 H

After this data transfer, the device goes in to NAP mode, all activities excluding the internal timer are stopped. This internal timer was configured through the second data byte of the previous message (*Table 24*) that after every 16*Bt, the device wakeup for 1*Bt (see also *Figure 43*). If during the NAP condition a wakeup is coming, the device will be active during the next 16*Bt period. If during this period the power mode is not changed, the NAP mode is entered again.

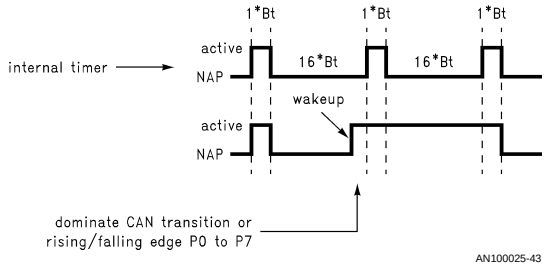


FIGURE 43. Timing NAP-Mode (16*Bt)

5.0 SLIO SYSTEM EXAMPLE

5.1 Start Up Consideration

In this section an example is shown to start a first CAN application. Before starting, the following steps have to be checked:

- At least two CAN nodes have to be connected on the CAN Bus because every message on the bus needs an acknowledge
- Usage of the same physical bus interface as described in Section 1
- Usage of the same bus mode (differential/single wire)
- Configuration of the termination on the two CAN bus endings depending on the physical bus interface

  →ISO High Speed (ext. Transceiver): 120Ω between CAN_H and CAN_L

  →ISO Low Speed: voltage divider 1.75V/3.25V recessive levels
- Usage of the same bus timing (described in Section 1) for all CAN nodes
- Consideration of length/frequency and the number of CAN nodes
- Consideration of the number of SLIO nodes depending on the SLIO Identifier mode

  →Pin mode: connected SLIO < 16 (only 4+1 Identifier can be configured)

  →EEPROM mode: quasi no limit (all ID in standard CAN format are used)

5.2 Network Description

These CAN communication examples between COP884BC and the SLIO describe the basis of an application with National CAN interface. The COP884BC software controls the CAN data transfer, which means that the counter value of a decrement 8-Bit counter is transmitted to the SLIO pins P0 to P7. In order to control the CAN data, the status of the counter is also given out to L_port of the COP884BC. The communication is restricted to SLIO CAN format. The schematics of COP884BC node and SLIO node are shown in *Figure 44* and *Figure 45*.

To start the application the following steps have to be executed:

- Reset COP884BC
- Create a rising edge to the port pin G0 for COP884BC
- Reset the SLIO

After the Controller receives the Status Message of the SLIO, the counter will be enabled and the data transfer begins. Next all CAN frames from COP884BC will be requested from the SLIO by an answering message. The software of COPCAN waits for this message and will generate the next data frame after a delay caused through the IDLE Timer pending flag T0PND.

The physical features are summarized in the next points:

- CKI = 10 MHz
- Bus Rate = 250 kbit/s
- External transceiver chip connection (ISO High Speed)
- Usage of the external EEPROM NMC93C06

The EEPROM configures the receive/transmit ID's to/from the SLIO (0023/0022), the bus mode and the data direction of P0 to P7. The configuration of the EEPROM registers are shown in *Table 25*.

TABLE 25. Example of Configuration of SLIO Registers in EEPROM Mode

E2-address	EEPROM Registers	SLIO Registers
0x00	**E2-MASK** 0xAA	—
	PIN-E2-MASK 0x00	—
0x01	**RXIDH** 0x00	—
	RXIDL 0x22	—
0x02	**RXIDGH** 0x00	—
	RXIDL 0x00	—
0x03	**PEDGE** 0x00	**PE** 0x00
	NEDGE 0x00	NE 0x00
0x04	**ODATA1** 0x00	**OD1** 0x00
	ODATA2 0x00	**OD2** 0x00
0x05	**DATADIR1** 0xFF	**DD1** 0xFF
	DATADIR2 0x00	**DD2** 0x00
0x06	**DACH** 0x00	**DAC2** 0x00
	DACL 0x00	**DAC1** 0x00
0x07	**ACR** 0x00	**ACT** 0x00
	CCR 0x00	**CCT** 0x00
0x08	**DCR** 0x08	**CTR** 0x08
	CAN_PSC 0x03	**CAN Prescaler** 0x03

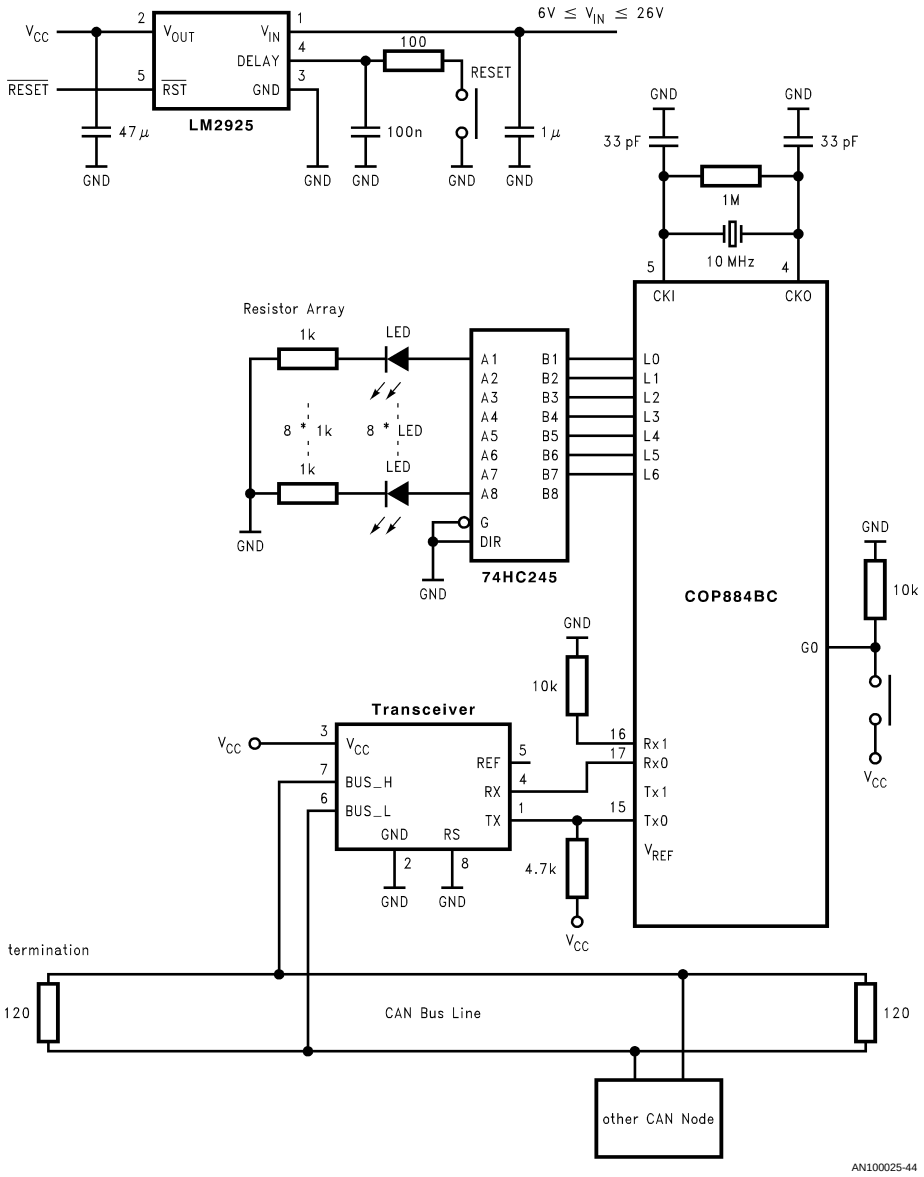


FIGURE 44. COP884BC Node

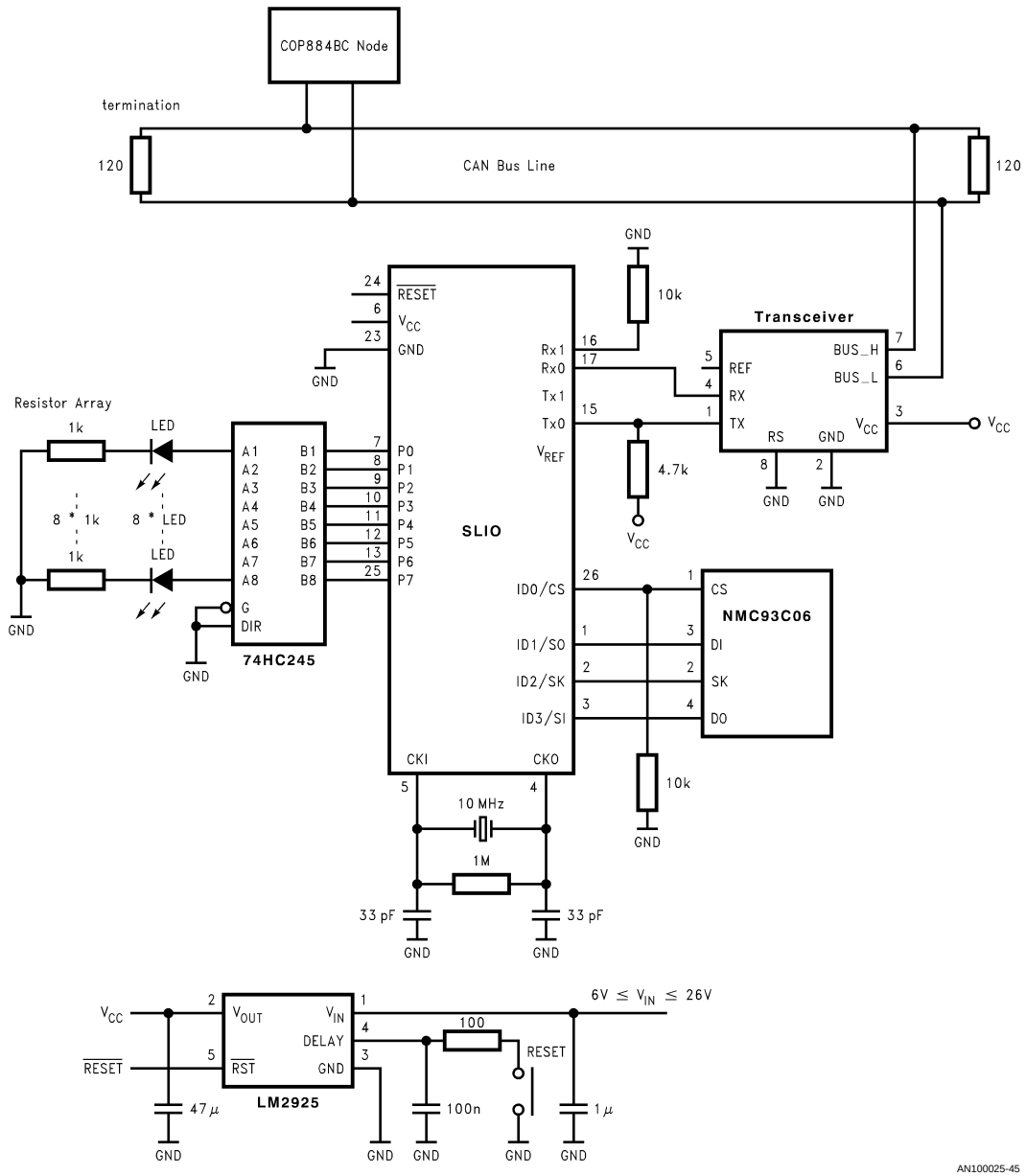


FIGURE 45. SLIO Node Schematic

5.3 Software Structure

The Block Diagram in *Figure 46* describes the software process. The software can be separated into the following modules.

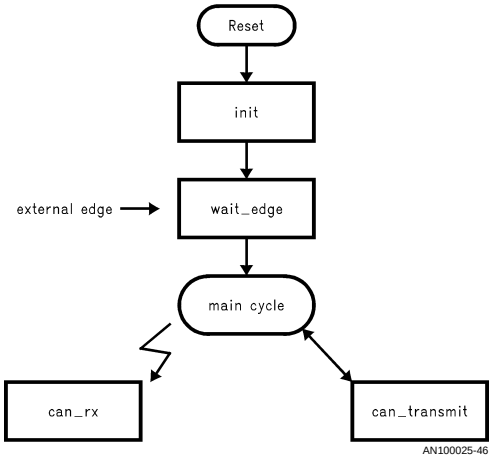


FIGURE 46. Software Block Diagram

- **initialization (init)**

After Reset the software will execute the initialization routine. Within this routine the various interrupts and the initialization of the CAN interface will be configured.

- **rising edge wait (wait_edg)**

Next the software waits for the rising edge on pin G0. If this rising edge is received, the interrupt routine enables the access of the application.

- **main cycle (main)**
- **CAN receive interrupt routine (can_rx)**

This is the same receive interrupt routine as that which was described for 2 bytes or less in Section 2. It processes the answering message from the SLIO and saves the data in the receive object rx_obj. Then the control bit is set and the COP884BC can transmit the next data message with the next counter value after a self defined delay.

- **CAN Transmit routine (can_tx)**

This is the same transmit interrupt routine as that which was described for 2 bytes or less in Section 2.

5.4 Source Code

```
.incld cop888bc.inc
tx_cnt = 0    ; flag equations for the control register
tx_dly = 1    ; flag equations for the control register
action = 2    ; flag equations for the control register
lo =   00a      ; delay (lo * 40960/CKI)
.sect msg_buf, base
     tx_obj:      .dsb 4
     ; transmit object format:
     ; tx_obj[0] = trtr, tid[10:4]
     ; tx_obj[1] = tid[3:0], tdlc[3:0]
     ; tx_obj[2] = txd1 !
     ; tx_obj[3] = txd2 !
     rx_obj0:.dsb 4
     ; rx_obj[0] = lock, rid[10:4]
     ; tx_obj[1] = rid[3:0], rdlc[3:0]
     ; tx_obj[2] = rxd1 !
     ; tx_obj[3] = rxd2 !
.endsect
;=============================================================================
.sect base,base
        control:        .dsb 1    ; allocation of flag control register
.endsect
;=============================================================================
.sect register,reg
        counter:        .dsb 1
        light:          .dsb 1
.endsect
```

```
;=========================================================================
.sect code,rom,abs=0
main:
reset:
        ld sp,#02f                      : load stack pointer
;-------------------------------------------------------------------------
; clear ram from 0x00 to 0x2f
; stack area will overwrite as well -> don't use as a subroutine
;-------------------------------------------------------------------------
clr_ram:
        ld      b,#02f          ; pointer to the last ram location
clr_loop:
        ld      [b],#0          ; clear ram byte
        drsz    b               ; decrement and "skip if zero"
        jp      clr_loop        ; ..counter>0
        ld      [b],#0          ; clear first ram byte
;-----------------------------------------------------------
init:
        ld      counter,#000    ; reset counter
        ld      light,#000      ; reset light counter
init_prt_l:
        ld      portlc,#0ff
init_G0:
        rbit    iedg,cntrl      ; ->rising edge
        sbit    exen,psw        ; enable extrn int
        rbit    expnd,psw       ; clear extern int pending
init_can:
        jsr     can_init
conf_rx_obj0:
        ld      b,#rx_obj0      ; configure receive message box
        ld      [b+], #082      ; with ID 0022 , 2 byte messages
        ld      [b], #022
enable_can:
        ld      cbus,#058       ; conf single wire rx0
                                ; RIAF enabled->compare with higher id's
                                ; enable CAN
enable_int:
        ld      b,#tcntl
        sbit    rie,[b]         ; enable can receive int
        sbit    gie,psw         ; enable global interrupt
;-------------------------------------------------------------------
;main cycle
;-------------------------------------------------------------------
start:
wait_begin:
        ifbit   action,control  ; wait until rising edge is coming
        jp      start_loop      ; yes.. process
        jp      wait_begin
start_loop:
        ifbit   tx_cnt,control  ; transmission
        jsr     cantx
        jp      start_loop
;-------------------------------------------------------------------
cantx:
        jsr     delay           ; delay routine
        jsr     action_count    ; count lights
        jsr     can_tx          ; transmit
        sbit    7,rx_obj0       ; enable receive buffer 0
        ret
;-------------------------------------------------------------------
delay:
        ld      counter,#lo     ; conf t0pnd_counter
        ld      b,#icntrl       ; point icntrl
dlay:
        rbit    t0pnd,[b]       ; reset t0 pending flag
```

```
loop_w:
        ifbit   t0pnd,[b]       ; wait unti t0pnd is set
        jp      count           ;
        jp      loop_w          ;
count:
        drsz    counter         ; count x*(40960/CKI)
        jp      dlay
        rbit    t0pnd,[b]       ;
        ret
;-----------------------------------------------------------------
action_count:
        rbit    tx_cnt,control  ; reset

        drsz    light
        nop

        ld      a,light
        x       a,portld
conf_tx_obj:
        ld      b, #tx_obj
        ld      [b+], #002      ; tid,#002
        ld      [b+], #032      ; tdlc,#032
        ld      [b+], #003      ; rxd1, #003
        ld      a,light
        x       a, [b]          ; rxd2, #count value light
        ret
;=============================================================================
.sect   code_can_init, rom

; this code initializes the CAN with minimum
; possible instructions/rom space
can_init:
        ldb, #cscal
        ld      [b+], #3        ; CAN prescaler
        ld      [b+], #00f      ; ctim (BTL)
        ld      [b], #0         ; TCNTL ; don't point to RTSTAT
                                ; clear RERR, TERR, etc..
        ret
.endsect
;=============================================================================
.sect   code_can_tx, rom
; this code transmits a 0 to 2 byte or remote CAN message
; from a transmit buffer tx_obj[0:3]
; this code intentionally does not check for remote or
; DLC (data length code) as the COPCAN interface will
; automatically transmit no data bytes in a remote frame
; and not more than DLC data bytes

can_tx:

        rc                      ; (*) reset error flag
        ifbit   TXPND, RTSTAT   ; check if transmit busy
        jp      tx_busy         ; .. yes then exit
        ld      b, #tx_obj      ; point to tx_obj[0]
        ld      x, #TID         ; point to TID
        ld      a, [b+]         ; get tx_obj[0]; point to tx_obj[1]
        x       a, [x-]         ; .. and save
        ld      a, [b+]         ; get tx_obj[1]; point to tx_obj[2]
        x       a, [x-]         ; .. and save
        ld      a, [b+]         ; point to tx_obj[3]
        ld      a, [b-]         ; get tx_obj[3]; point to tx_obj[2]
        x       a, [x-]         ; save to TXD1
        ld      a, [b]          ; get tx_obj[3]
        x       a, [x]          ; save to TXD2
```

```
tx_done:
        sbit    txss, tcntl     ;set pending transmission
                                ;automatic reset of txss after transmission
        ret                     ; exit without error
tx_busy:
        sc                      ; (*) indicate tx_busy
        ret                     ; (*) exit with error
        ; retsk                 ; optional use retsk instead
                                 ; 1st and last 2 lines to skip next
.endsect
;=============================================================================
.sect   int,rom,abs=0ff
interrupt:
        push    a
        ld      a,b
        push    a
restore:
        vis
int_end:
        pop     a
        x       a,b
        pop     a
        reti
.endsect
;=============================================================================
.sect   inttab, rom , abs=01E0
        .addrw restore          ; default VIS
        .addrw restore          ; PortL interrupt/wake-up
        .addrw restore          ; reserved
        .addrw restore          ; reserved
        .addrw restore          ; reserved
        .addrw restore          ; PWM Timer
        .addrw restore          ; MicroWire/Plus
        .addrw restore          ; T1B
        .addrw restore          ; T1A
        .addrw restore          ; Idle Timer
        .addrw int_g0           ; Pin G0
        .addrw restore          ; CAN Transmit
        .addrw restore          ; CAN Error
        .addrw can_rx           ; CAN Receive
        .addrw restore          ; reserved
        .addrw reset            ; Opcode 00 Software-Trap
.endsect
;=============================================================================
.sect   code_can_rx, rom        ; from interrupt
can_rx:
        ; this interrupt is triggered by RBF, RRTR or RFV
        ; RRTR and RBF are cleared by reading or b's pointing to RXD1
        ; RFV is cleared by reading RTSTAT to A
        ; or executing the equiv. of LD B, #RSTAT; LD A, #xx
;------
        sbit    tx_cnt,control
;------
        ld      b, #rx_obj0     ; (*) receive id hi   ; * only with RIAF = 0
        ifbit   7, [b]          ; buffer free
        jp      receive_msg     ; .. yes then receive
        ld      b, #rx_obj1     ; next buffer
        ifbit   7, [b]          ; buffer free
        jp      receive_msg     ; .. then receive msg
        jp      can_rx_exit     ; else exit
receive_msg:
        rbit    7, [b]
        ld      a, rid          ; (*) get received id
        ifne    a, [b]          ; (*) check if accept
        jp      can_rx_exit     ; (*) .. no then exit
        x       a, [b+]
        ld      a, ridl         ; get received IDLC
```

```
        x       a, [b]          ; save message
        ifbit   RRTR, RTSTAT    ; received frame remote frame?
        jp      can_rx_rtr      ; yes
        jp      save_data       ; no
can_rx_rtr:
        ld      a,[b]           ; remote frame is signed
        or      a,#0F           ; through rdlc = F
        x       a,[b]           ;
        jp      wait_rx         ;
save_data:
        ld      a,[b+]          ;dummy read -> point rx_data register
        ld      a, RXD1         ;
        x       a, [b+]
        ld      a, RXD2
        x       a, [b]
        ld      b, #RTSTAT
wait_rx:
        ifbit   RFV, [b]
        jp      rx_done
        ifbit   RERR, TCNTL
        jp      rx_error
        jp      wait_rx
; this is the error routine error interrupt must not be enabled
rx_error:
        ld      b, #rx_obj1
        ifbit   7, [b]
        jp      check_obj0
        jp      end_error
check_obj0:
        ld      b, #rx_obj0
end_error:
        sbit    7, [b]          ; free buffer
        rbit    RERR, TCNTL
rx_done:
can_rx_exit:
        ld      a, RXD1         ; dummy read to clear RBF, RTR
        ld      a, RTSTAT       ; dummy read to clear RFV
        jp      int_end
.endsect
;=========================================================================
.sect   rom,rom
int_g0:
        sbit    action,control
        rbit    expnd,psw       ;clear extern int pending
        jp      int_end
.endsect
.end main
```

Book Extract End

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component in any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

www.national.com

National Semiconductor Europe
Fax: +49 (0) 1 80-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 1 80-530 85 85
English Tel: +49 (0) 1 80-532 78 32
Français Tel: +49 (0) 1 80-532 93 58
Italiano Tel: +49 (0) 1 80-534 16 80

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: sea.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5620-6175
Fax: 81-3-5620-6179