MOTOROLA
intelligence everywhere™

digital dna™

Konnex PL132 Over Power Line Based on the M68HC08 — Demo Application

Designer Reference Manual

DRM009/D
Rev. 0
2/2003

M68HC08
Microcontrollers

MOTOROLA.COM/SEMICONDUCTORS

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Konnex PL132 Over Power Line Based on the M68HC08 — Demo Application Reference Design

By: Zdenek Kaspar
Jaromir Chocholac
Marek Stricek
System Applications Engineering
MCSL — Motorola Czech Systems Laboratories
Roznov, Czech Republic

Freescale Semiconductor, Inc.

Motorola and the Stylized M Logo are registered trademarks of Motorola, Inc.
DigitalDNA is a trademark of Motorola, Inc.
This product incorporates SuperFlash® technology licensed from SST.
© Motorola, Inc., 2003

For More Information On This Product,
Go to: www.freescale.com

Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://motorola.com/semiconductors

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision History

Date	Revision Level	Description	Page Number(s)
February, 2003	N/A	Initial release	N/A

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Sections

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Sections

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table of Contents

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Table of Contents

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 5. Hardware Design Description

Section 6. Conclusions

Section 7. Source Code

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table of Contents

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Figures

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Figures

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Tables

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

List of Tables

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 1. Introduction

1.1 Contents

1.2 Introduction

The main objective of this project is to provide a design that shows the key parts of the implementation of a power line modem according to the Konnex PL132 specification. This protocol is chosen as a European communication standard.

The idea of using the power line (mains) wiring infrastructure as a communication medium is very old. It is quite easy to see why — it has the most ubiquitous coverage of any other possible media. It is almost certain that you can find at least one or two power outlets in every room of your house. And according to the phrase “no new wires to your homes”, this is a very low cost solution.

Konnex is a European consortium that is driving a European Standard for low-cost, low-speed connectivity for home and building markets. The Konnex Standard up to now covers three physical mediums:

1. Twisted Pair — TP0 and TP1 variants with data rates of 1200 and 2400 baud
2. Power line (mains) — PL110 specification with:
   – Data rate of 1200 baud
   – PL132 variant with center frequency of 132.5 kHz
   – Data rate equal to 2400 baud
3. RF with 868 MHz carrier and data rate 16.384 baud

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Introduction

The power line modem is based on Motorola's MC68HC908GR8 8-bit microcontroller unit which controls a half-duplex synchronous FSK modem device. The project consists of:

- Hardware design of the PCBs based on the Motorola MCU
- Software development of the physical and data-link layers according to the Konnex PL132 specification
- A demo application communicating bidirectionally

The concept of the whole system is shown in **Figure 1-1**.

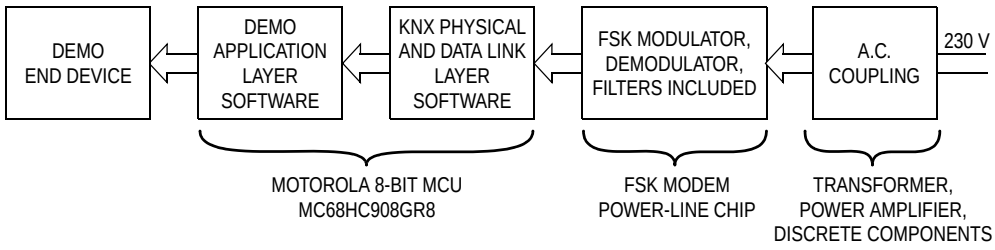


Figure 1-1. Project Design Block Diagram

The heart of the design is the Motorola MC68HC908GR8 MCU, where the central part of the physical layer and the data link layer are implemented. The FSK modem device chip with external coupling circuits serves as part of the physical layer of the ISO-OSI reference model. An interface to the higher layers is done through API functions. Because of the demonstration purpose, a basic proprietary implementation of an application layer, dedicated to a particular end-device, is also made.

This document consists of the following sections:

- **Section 2. Quick Start** showing how to setup and use the project demo
- **Section 3. Konnex Introduction** providing a basic overview of the Konnex consortium
- **Section 4. Software Development** providing a description of the software portion of the project
- **Section 5. Hardware Design Description** providing a description of the hardware design

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

- **Section 6. Conclusions** providing a summary and evalution of the project
- **Section 7. Source Code** providing the source code listings associated with this project

NOTE: *It is supposed that the reader of this project has at least a basic understanding of the structure and naming convention of the Konnex PL132 specification.*

1.3 Project Goal

The demonstration configuration (see **Figure 1-2**) consists of two kinds of power line modem (PLM) boards communicating with each other via the normal mains power supply; on one side the Master PLM board, while on the other side the Slave PLM board.

The master PLM board (Master PCB) has the KNX132 communication functions as shown in **Figure 1-2**. They are:

- An isolated power supply
- A user interface with four buttons, four LEDs, and one knob

The slave PLM board (Slave PCB) also has the KNX132 communication functions, as shown in **Figure 1-2**. They are:

- A non-isolated power supply
- A running intelligent domestic light demo application, which shows control features of the connected lamp as an end-device

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

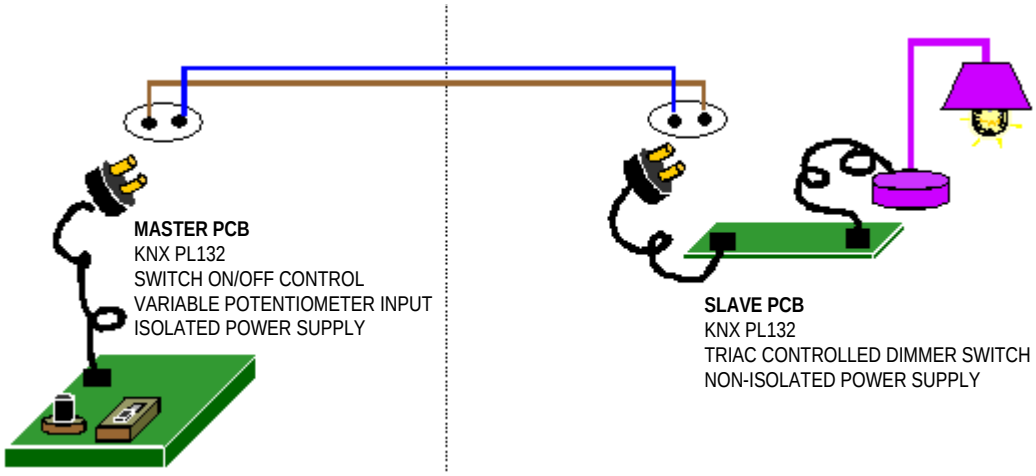


Figure 1-2. Configuration of Intelligent Domestic Light Demo Application

1.4 Developed Boards and Demo Application Features

Features of the master device (board):

- Power line connectivity according to the key parts of the Konnex PL132 physical and data link layers specification
- Proprietary application layer implementation for the master device
- User interface of the demo application
- Isolated power supply

Features of the slave device (board):

- Power line connectivity according to the key parts of the Konnex PL132 physical and data link layers specification
- Proprietary application layer implementation for the slave device
- Triac for a light dimmer
- Lamp as an end-device
- Non-isolated power supply

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 2. Quick Start

2.1 Contents

2.2 Introduction

This section describes the connection and startup of the Konnex PL132 over power line based on M68HC08 demo application.

2.3 Demo Application Overview

As mentioned previously, the configuration of the application uses two kinds of power line modem boards:

- One master board — see **Figure 2-1** and **Figure 2-2**
- Up to three slaves — see **Figure 2-3** and **Figure 2-4**

While one connected slave is a minimal configuration, three is the maximum number of connected slave devices for the demo application.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Quick Start

Figure 2-1. Master Device (Master PCB Board and Box)

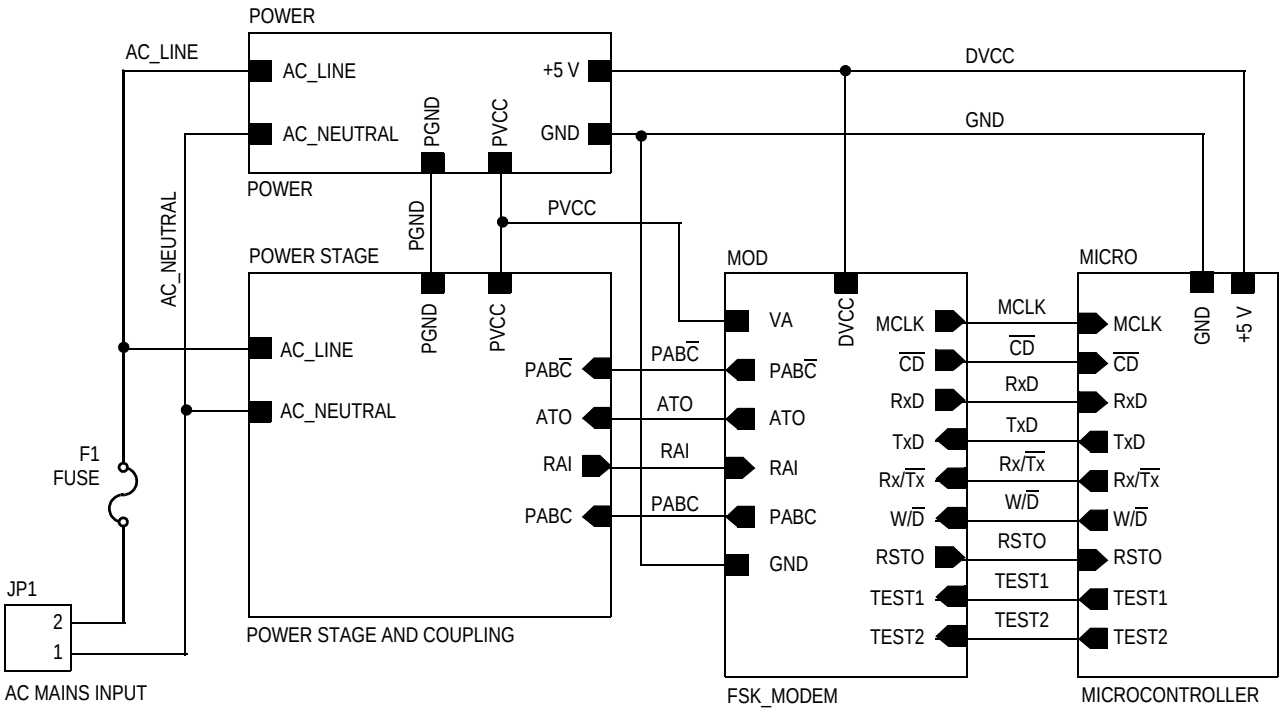


Figure 2-2. Master Device Block Diagram

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Figure 2-3. Slave Device (Controlled Lamp and Slave PCB)

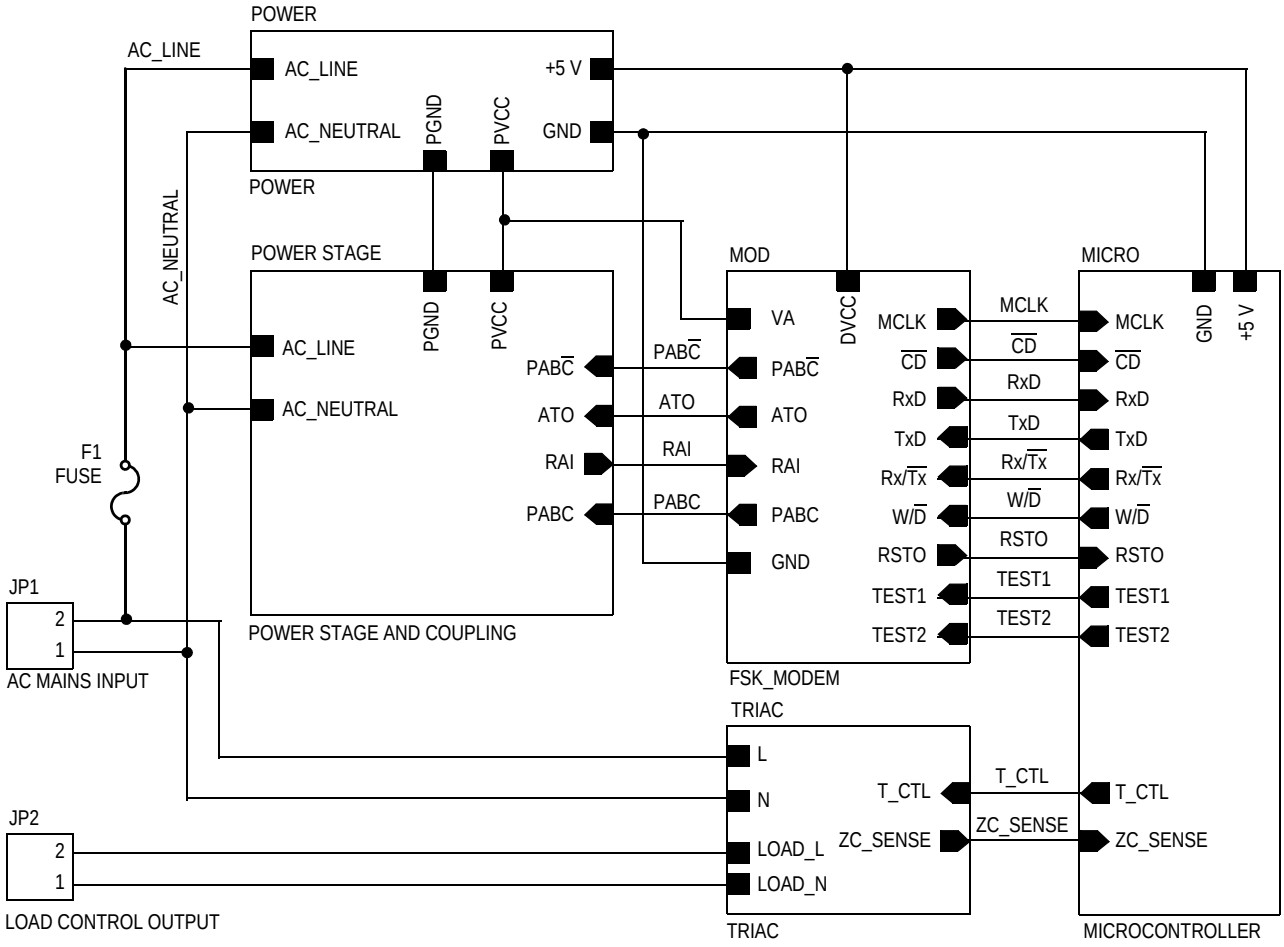


Figure 2-4. Slave Device Block Diagram

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

2.4 User Interface Description

The master board user interface is shown in **Figure 2-5**. It has:

- Four buttons
- Four LED indicators (red is activated when the logical state is equal to 0; otherwise, green is switched on)
- A knob for the light intensity control

Device A to *Device C* buttons select the desired device for control. Three LEDs belonging to those buttons signal the state of the respective device (green stands for *"Device is connected"*, red for *"Device is off-line"*. The *Power ON / OFF* button switches over *"Light is ON according to the desired light intensity"* and *"Light is OFF"* states of the selected device, while its LED signals the state (red light for OFF, green for ON) to the user. Change in the position of the intensity knob sends a "*Switch light ON according to the intensity value*" command to the lamp of the active slave device.

There is no user interface on the slave boards, as these boards are completely controlled via the mains. However, the application running on the slave boards controls the dimmer triac of the lamp, and is responsible for communication according to the application layer code.

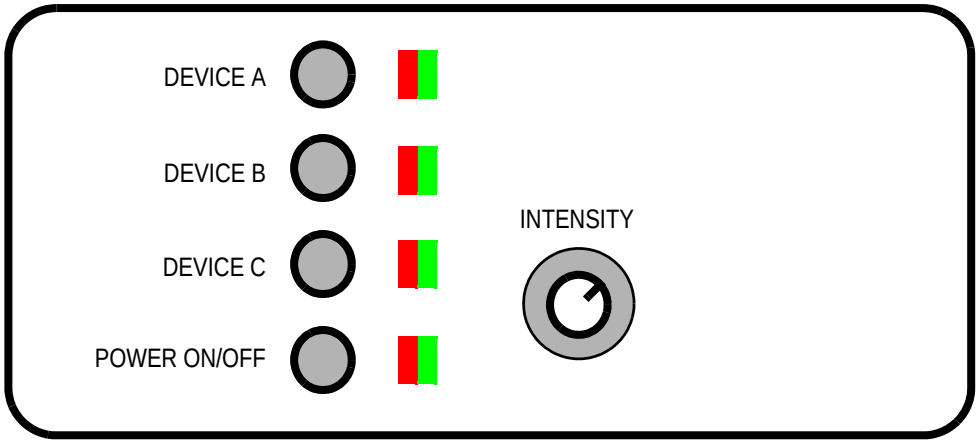


Figure 2-5. User Interface on the Master Board

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

2.5 Demo Application Startup

The necessary startup of the demonstration is very straightforward. Simply connect all devices (one master and up to three slave devices) to the mains and wait a second till the master board detects all connected slaves. After that, the configuration of the demo application is finished and the application should be ready for use.

NOTE: *Make sure that all devices are connected to the same phase of the power line. In addition, a maximal distance between the master and slaves boards where communication is still possible mainly depends on the condition on the mains. The channel between any two outlets in a home has an extremely complicated transfer function with many stubs having terminating loads of various impedances. Such a network has an amplitude and phase response that varies widely with frequency. Worse, the transfer function can change with time. This might happen because the homeowner has plugged or unplugged a device from the power line, or if some of the devices plugged into the network have time-varying impedances. As a result, the nature of the channel between outlet pairs may vary over a wide range. In addition to the transfer function problem, and equally significant, interference over the power line must be considered. Typical sources of noise are brush motors, fluorescent and halogen lamps, switching power supplies and dimmer switches. The net impact of these different interference sources is that received raw data bits tend to have significant numbers of bit errors, which must somehow be corrected. These are the main obstacles of PLC, and they all have to be considered.*

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Quick Start

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 3. Konnex Introduction

3.1 Contents

3.2 Introduction

As mentioned previously, Konnex is a European consortium that is driving a European Standard for low-cost, low-speed connectivity for home and building markets.

The main features of the Konnex PL132 specification are show here. The physical levels definitions of the PL132 are according to the European regulation norm CENELEC EN 50065-1 *Signaling on low-voltage electrical installations in the frequency range 3 kHz to 148.5 kHz*.

- Frequency Shift Keying (FSK) modulation with low-level frequency deviation
- Center frequency of 132.5 kHz (± 0.25 kHz)
- Deviation of ± 600 Hz ± 1%
- Bit rate of 2400 bits per second
- Forward Error Correction (FEC) implemented with a capability of correction up to three bit bursts in a block of 14
- Protection by a 16-bit long Cyclic Redundancy Check (CRC)

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

- There are two kinds of frames available:
  - *Short frame* can transport up to 15 octets
  - *Long frame* format supports from 16 to 65 octets

The brief introduction to the *ISO / OSI* (International Standard Organization / Open System Interconnection) *reference model* and to the basic role of the physical and data link layers, is provided in the introduction to **Section 4. Software Development**. For the unexperienced developers it is recommended that this chapter be read first.

3.3 Physical Layer of Konnex PL132

Format of the Konnex PL132 frame on the physical layer can be seen in **Figure 3-1**.

PREAMBLE (16)	HEADER (16)	DATA (8)	**FEC** (6)	DATA (8)	**FEC** (6)	ETC.	POSTAMBLE

Bold *denotes ones complement*

Figure 3-1. Complete Frame Encapsulation (Datagram)

Frame Preamble of Konnex PL132 has a 16-bit field equal to AAAAh.

For data frames (*Datagrams*), Konnex PL132 requires a value equal to 1C53h as a *Datagram Header*, while the Acknowledge *message header* is 1CA1h.

Parts of the physical layer frame, called *Data*, carry the data link layer information as shown in **Figure 3-2** and **Figure 3-3**.

Forward Error Correction (FEC) field consists of redundant information for error detection and possible correction when an error during reception occurs. Six bits of FEC information are generated for each eight bits (octet) of physical layer data.

Frame Postamble is 2-bit long field xx, where x is the complement of the last bit of the FEC for the last octet.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

3.4 Data Link Layer of the Konnex PL132

On the data link layer, two frame types are provided:

- **Acknowledge (ACK) frame** format (**Figure 3-2**) is used to acknowledge an L_Data frame; it consists of a data field of two octets, its value is equal to the transmitted Frame Check Sequence (FCS).

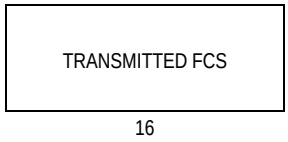


Figure 3-2. Data Field in Acknowledge (ACK) Frame

- **L_Data frame** format is dedicated for the data transmission; it can be seen in **Figure 3-3**.

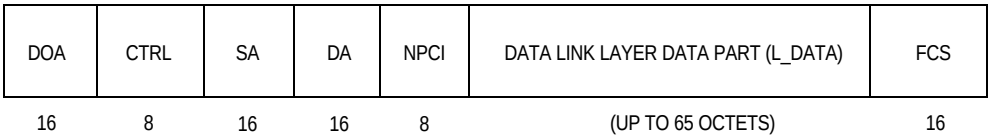


Figure 3-3. L_Data Frame Format with Standard Field Name Abbreviations

The encoding of the frame fields is done as follows:

- **Domain Address (DOA)** — Each domain is allocated a 16-bit address, assigned according to the Domain. The domain address 0000h is for broadcast. Some values are reserved for dedicated use.
- **Control Field (CTRL)** — The control field contains the information about the Layer-2 (data link layer) service which includes frame priority, identification of group or individual addresses, a flag indicating whether the frame is a repeated one, and a flag indicating if an acknowledgement is requested. See **Figure 3-4**.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
L/S	REP	G/I	ACK	1	1	PR_1	PR_0

Figure 3-4. Control Field (CTRL)

- **Source address (SA)** — The source address is the individual address of the device that requested the transmission of the frame.
- **Destination address (DA)** — The destination address defines the device(s) that shall receive the frame.
- **Network Protocol Control Information (NPCI)** — This field is a composite octet containing a reserved bit, routing information (Hop Count, HC), and length information (LG). See **Figure 3-5**.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	HC_2	HC_1	HC_0	LG_3	LG_2	LG_1	LG_0

Figure 3-5. Network Protocol Control Information Field (NPCI)

- **L_DATA** — This field contains the transported data octets of the data link layer.
- **Frame Check Sequence (FCS)** — The last two octets of a frame transport the frame check sequence.

For more information about this topic see the Konnex PL132 specification and regulation norm CENELEC EN 50065-1 *Signaling on low-voltage electrical installations in the frequency range 3 kHz to 148.5 kHz*.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 4. Software Development

4.1 Contents

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

4.2 Introduction

Software development can be divided into several independent stages. The first key part of the development is the software implementation of a physical layer according to the Konnex PL132 specification. This layer is responsible for the connection of the dedicated power line modem FSK modem device chip, as the hardware part of the physical layer implementation. This layer provides transmission of unstructured bits across the physical medium, as defined by the ISO / OSI (International Standard Organization / Open System Interconnection) reference model says.

NOTE: *Konnex PL132 specification treats the Forward Error Correction (FEC) and the framing of the transferred data as a part of the physical layer.*

The second important part of the development is the implementation of a data link layer. This layer of the reference model is accountable for data transfer across the physical links, error detection and possibly correction. When there are no upper layers implemented in the system, it will also substitute all the necessary functions of those layers, such as addressing.

For demonstration purposes an application layer of the ISO / OSI reference model was written as well. An introduction to the application itself is given in **Section 2. Quick Start**.

NOTE: *It was the authors intention to make the implementation of each layer of the Konnex PL132 specification as independent as possible.*

For More Information On This Product,
Go to: www.freescale.com

Because of the fact that the implementation of the Konnex PL132 physical and data link layer was the main portion of the project, this part of the development is prominently featured in the documentation. The developed demo application is aimed only as an example of the Konnex PL132 connectivity application. Therefore, focus is centred on the physical and the data link layer related sections. Because of this fact, it was also decided that the document structure follows the ISO / OSI (International Standard Organization / Open System Interconnection) reference model layer structure.

4.3 List of the Project Files

The project was written using Metrowerks® Code Warrior V1.2 environment. Here is a list of all the project source code files:

- **phs.c** contains a complete physical layer implementation (transmission and reception routines on the physical layer level)
- **phs.h** is a header file of **phs.c**; it contains the whole set of physical layer related symbolic constants and function style macros
- **dll.c** consists of a data link layer implementation (transmission and reception of the Konnex frames on the data link layer level)
- **dll.h** is a header file of **dll.c**; it includes type and structure definitions of the data link layer related implementation
- **app.c** and **app.h** contain routines of the demo application (user interface control for the master board, triac control for the slave board)
- **main.c** has the *main()* routine of the demo project
- **board.h** contains hardware dependant definitions for both master and slave board
- header file **hc08gp32.h** defines all of the MCU registers
- **hc08gr8.prm** is a standard parameter file of the project

Metrowerks and the Metrowerks logo are registered trademarks of Metrowerks, Inc., a wholly owned subsidiary of Motorola, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

NOTE: *There is only one Metrowerks Code Warrior project for both master and slave devices written. The differentiation between those two types of software is made using two different targets: “Master - release” for Master, and “Slave - release” for slave.*

For the MCU flashing, the Developer’s serial bootloader for M68HC08 was utilized. For more information refer the application entitled *Developer’s Serial Bootloader for M68HC08* (Motorola document order number AN2295).

4.4 Used MCU Peripheries

This subsection briefly describes, in the form of tables, all MCU resources such as peripheral components, interrupts, and memory used in the project.

Table 4-1 provides a list of the GPIO (general-purpose inputs and outputs) and analog input pins used. The table is divided into Konnex connectivity related, master board application related, and slave board application related areas.

Table 4-2 provides a list and a short description of the timer modules used in the project. Note that the table contains Konnex PL132 connectivity related usage of the timer, as well as the application ones.

All interrupts of the Konnex PL132 over power line based on M68HC08 project are briefly detailed in **Table 4-3**.

Table 4-1. GPIO and Analog inputs

Pin	Direction	Symbolic Name	Purpose
Konnex connectivity related (valid for both master and slave boards)			
PTA0	Input	$\overline{CD}$	Carrier detection signal from the FSK modem device
PTB0	Output	Rx / $\overline{Tx}$	Enable / disable transmission signal to the FSK modem device
PTB1	Output	$\overline{WD}$	Watchdog service signal to the FSK modem device
PTB2	Output	TEST1	Signal to control the FSK modem device
PTB3	Output	TEST2	Signal to control the FSK modem device
PTD4	Output	RxD	RxD signal from the FSK modem device
PTD6	Input	TxD	TxD signal to the FSK modem device

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 4-1. GPIO and Analog inputs (Continued)

Pin	Direction	Symbolic Name	Purpose
Application related, valid only for master boards			
PTA1	Input	SW1	Switch 1 of application
PTA2	Input	SW2	Switch 2 of application
PTA3	Input	SW3	Switch 3 of application
PTB5 / AD5	Analog input	POT_INT_CTRL	Intensity control of application
PTD5	Input	SW4	Switch 4 of application
PTD0	Output	IN1	Indication LED 1 of application
PTD1	Output	IN2	Indication LED 2 of application
PTD2	Output	IN3	Indication LED 3 of application
PTD3	Output	IN4	Indication LED 4 of application
Application related, valid only for slave boards			
PTD0	Input	JP4	Node address value
PTD1	Input	JP5	Node address value
PTD2	Input	JP6	Node address value
PTD3	Input	JP7	Node address value
PTC0–C1	Output	T_CTL	Triac control signal of application
PTA1	Input	ZC_SENSE	Zero-cross sense signal for triac control timing in application

Table 4-2. Timers

Timer	Tx / Rx / Application	Master / Slave / Both	Configuration and Description	ISR Function
T1Ch0	Rx	Both	Input capture mode with falling edge detection of the Rx bit (synchronization of incoming bit stream)	phs_RxEdgeISR()
T1Ch1	Application	Slave	Output compare event for triac control	app_TriacTmrISR()
T1 overflow	Application	Slave	Overflow in half period of AC line	No ISR
T2Ch0	Rx	Both	Output compare in one half of bit period	phs_RxBitISR()
T2Ch0	Tx	Both	Write new output value into the Tx bit during the output compare event	No ISR
T2 overflow	Rx	Both	Bit period timer overflow during reception stage (synchronization of incoming bit stream)	No ISR
T2 overflow	Tx	Both	Load new value for the next output compare based transmission	phs_TxBitISR()

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Table 4-3. Interrupts

Periphery Module	ISR Function	Type of Interrupt	Konnex / Application Related	Purpose
$\overline{\text{IRQ1}}$	phs_IRQ_ISR()	IRQ	Konnex	Reset from the FSK modem device
Tmr1Ch0	phs_RxEdgeISR()	Input capture	Konnex	PL reception synchronization timer
Tmr1Ch1	app_TriacTmrISR()	Output compare	Application	Triac control timer
Tmr2Ch0	phs_RxBitISR()	Output compare	Konnex	PL reception timer
Tmr2 overflow	phs_TxBitISR()	Timer overflow	Konnex	PL transmission timer
KBD0 (keyboard interrupt)	phs_CDdetectISR()	Falling edge	Konnex	Start and stop of the PL reception
TBM (timebase module)	dll_TBModuleISR()	TBM	Konnex and application	Low-speed timer

4.5 State Diagram of the Project

State diagram of the project is shown in **Figure 4-1**. Detailed information about the function and data flow descriptions are presented in **4.6 API Functions of the PL132 Physical and Data Link Layer**, **4.7 Buffers and Data Flow Details** and **4.10 Data Link Layer Implementation Details**.

4.6 API Functions of the PL132 Physical and Data Link Layer

The connection between the Konnex PL132 physical and data link layers and the application layer is done by simply using two API functions and two private functions as shown in **Figure 4-1**.

A brief description of the functions is given in this subsection. More detailed information can be found in the **4.10 Data Link Layer Implementation Details**. Information about the buffers used is given in **4.7 Buffers and Data Flow Details**.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

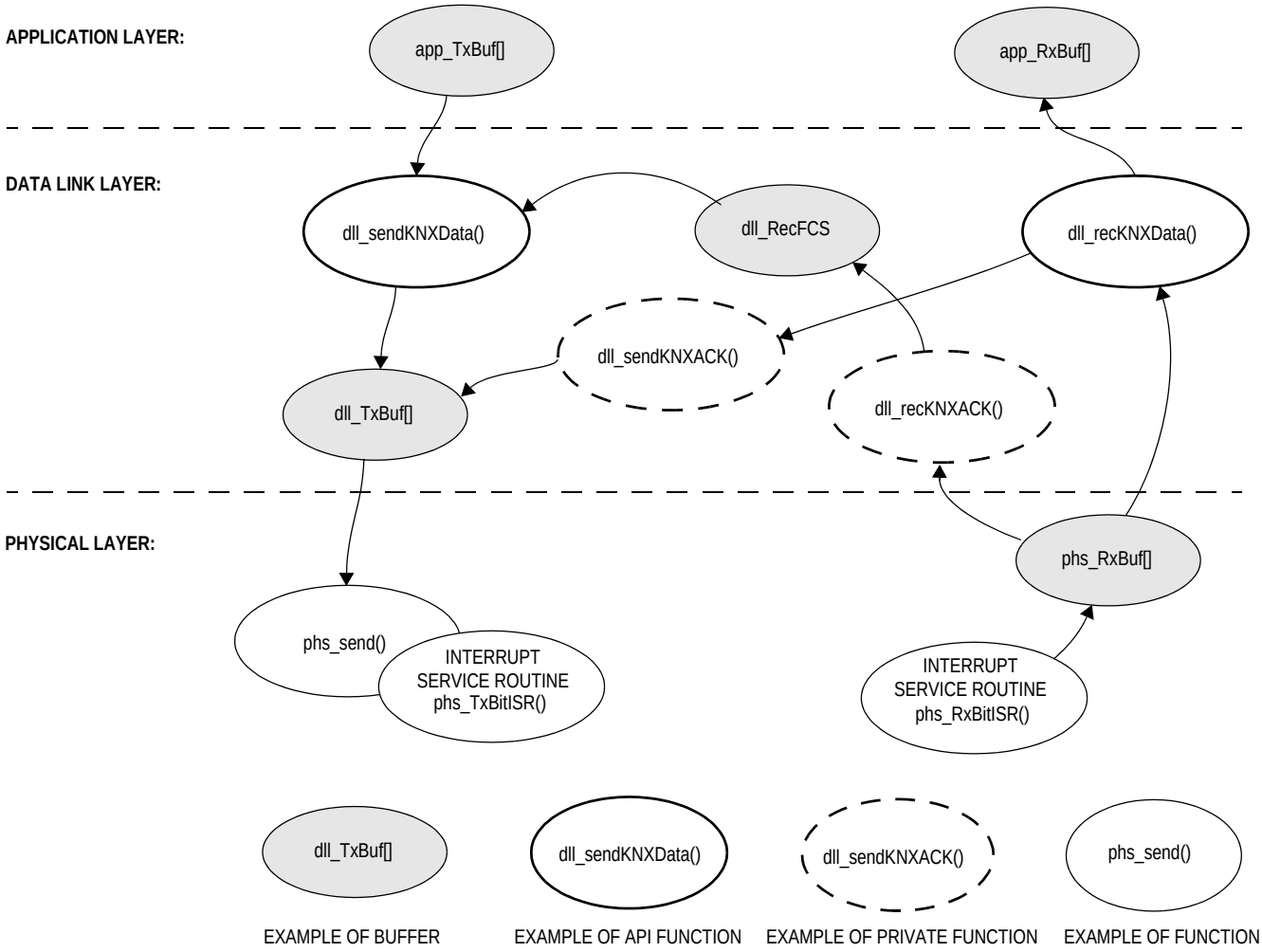


Figure 4-1. State Diagram

API functions of the Konnex connectivity are:

- For **data frame transmission**, the *dll_sendKNXData()* function is used
- For **data packet reception**, the *dll_recKNXData()* function is used

Private functions of the Konnex connectivity implementation are:

- For **acknowledge (ACK) packet transmission**, the *dll_sendKNXACK()* function is used
- For **acknowledge (ACK) packet reception**, the *dll_recKNXACK()* function is used

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.6.1 API for Data Frame Transmission

For a transmission of the data packet, the *dll_sendKNXData()* function is used. It sends the data message of the PL132 format to the FSK power line modem device on the data link layer level. Control fields of the data message are taken from the function parameters, see function prototype below.

```
dll_tTxStatus dll_sendKNXData(unsigned int doa, unsigned char ctrl, unsigned int sa,
                              unsigned int da, unsigned char npci, unsigned char * pAppBuf);
```

Arguments of the function are:

- *doa* — Domain Address (DOA)
- *ctrl* — Control Field (CTRL)
- *sa* — Source Address (SA)
- *da* — Dource Address (DA)
- *npci* — Network Protocol Control Information (NPCI)
- *pAppBuf* — pointer to application transmission buffer

The function returns a *status* variable, with the following declaration of *dll_tTxStatus* type:

- *DLL_TX_INITIAL* — initial state of the variable describing the transmission on data link layer
- *DLL_TX_OK* — transmission completed on data link layer, no ACK signal was required
- *DLL_TX_ACK_OK* — transmission completed on data link layer, ACK required => ACK message received successfully
- *DLL_TX_ACK_BAD* — transmission not completed on data link layer since no ACK received from first initial + two retransmit transmissions
- *DLL_TX_BUSOCC* — transmission not completed on data link layer since unexpected bus occupation during retransmits

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.6.2 API for Data Frame Reception

Function *dll_recKNXData()* is dedicated to reception of the Konnex PL132 data frame. It gets the message in the physical layer format, as stored in the *phs_RxBuf[]* buffer, and then it checks the address and the control FCS part of the message. If the message requires an acknowledge (ACK), it also sends the ACK message to the sender.

If both the address and FCS are correct, flag *dll_FlgRxDataComp* (reception of the data message completed) is set in order to inform the application layer level, and the data is moved into the application buffer *app_RxBuf[]*.

4.6.3 Private Functions of the Konnex Connectivity

There are two other functions of the data link layer implemented. They are both private since they are only called from the previously mentioned *dll_sendKNXData()* and *dll_recKNXData()* API functions.

1. First one, called *dll_sendKNXACK()*, is dedicated for transmission of the Datagram ACK messages on the data link layer level.

   Argument of the function is:
   fcs — Frame Check Sum (FCS)
2. Second function, called *dll_recKNXACK()*, is used for reception of Datagram ACK messages on the data link layer. It gets the message in physical layer format (stored in *phs_RxBuf*) and extracts the FCS part of the message to the *dll_RecFCS* buffer for the application, while setting *dll_FlgRxACKComp* flag (reception of ACK message completed).

4.7 Buffers and Data Flow Details

In this subsection the description of the message buffers and data flows will be given for both application and physical and data link layer implementation.

NOTE: *For transmission as well as for reception, there are two buffers defined; one is an application layer level buffer, and the second is used for the physical and data link layers.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.7.1 Application Buffer for Reception

For the reception on the application layer the *unsigned char app_RxBuf[APP_BUF_LEN]* array is used. Content of this buffer is “pure” application data. In *app_RxBuf[0]* the value of the length of the message is stored, which can be up to 15 octets (bytes) long. This value corresponds to the fact that short frames of PL132 can transport from 0 to 15 octets.

When successful reception occurs, the *dll_recKNXData()* API function writes its results into the *app_RxBuf[APP_BUF_LEN]* buffer. For this reason, use of this buffer is obligatory.

4.7.2 Application Buffer for Transmission

On the other hand, for the application transmission buffer, either the predefined *unsigned char app_RxBuf[APP_BUF_LEN]* or a user defined buffer can be used (as can be seen in example given in **4.8 Application Template**). The pointer to one of these buffers is one of the input arguments of the *dll_sendKNXData()* API function described in the **4.6 API Functions of the PL132 Physical and Data Link Layer**. The content of this type of buffer is “pure” application data. The length of the message can be up to 15 octets (bytes) long. This value corresponds to the fact that short frames of PL132 can transport from 0 to 15 octets.

4.7.3 Data Link Buffer for Transmission

The transmission buffer of the data link layer, defined as *unsigned char dll_TxBuf[DLL_TXBUF_LEN]*, contains a message in its final format, as it will be sent to the power line via a physical layer transmission routine called *phs_Send()*.

For the Datagram message transmission this buffer is completely filled by the *dll_sendKNXData()* function. It adds an additional 11 octets into a message; application data is grabbed from the *app_TxBuf[]* without change. These additional octets are the standard frame fields of Konnex PL132 data link layer, except the first one which is a second part of the message header of the physical layer.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

NOTE: *For data messages (Datagrams) Konnex PL132 requires 53h as the second part of the header (1C53h is a whole Datagram header), while the second part of the Datagram's ACK message header is A1h (1CA1h).*

Structure of the message fields of the PL132 as well as the whole Datagram transmission data flow can be seen in **Figure 4-2**.

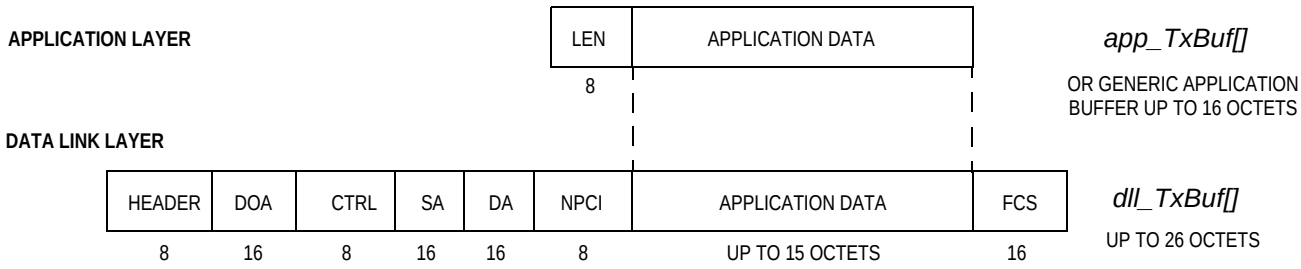


Figure 4-2. Data Flow of Transmission of the Datagram Message

The transmission buffer *dll_TxBuf[DLL_TXBUF_LEN]* is also used for the transmission of the ACK Datagram messages via the *dll_sendKNXACK()* function. This function is only called from the *dll_recKNXData()* routine and it does not use information from the application app_TxBuf[] buffer. Therefore, is not available to the application designer. The structure of the message can be seen in **Figure 4-3**.

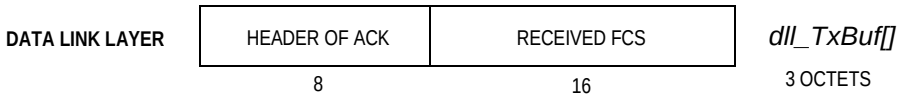


Figure 4-3. Structure of the ACK Datagram message

4.7.4 Physical Layer Buffer for Reception

The second reception buffer running on the lowest physical layer is defined as *unsigned char phs_RxBuf[KNX_BUF_LEN]*. It is up to 25 octets long and is filled bit by bit during the power line reception in the *phs_RxBitISR()* function. When reception finishes, either the *dll_recKNXData()* or the *dll_recKNXACK()* function is called, according

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

to the value of the received message header. Those two functions extract the application information from the physical layer buffer.

First, *dll_recKNXData()* is called if the received message in *phs_RxBuf[]* contains the Datagram information. Structure of the Datagram reception dataflow can be seen in **Figure 4-4**.

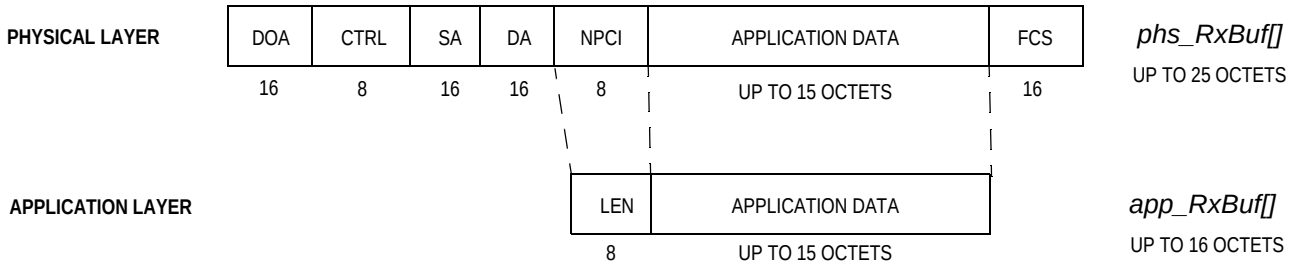


Figure 4-4. Data Flow of Reception of Datagram Message

For the Datagram ACK messages the second function, *dll_recKNXACK()*, is utilized. The *dll_recKNXACK()* function is only called from the*dll_sendKNXData()* routine; thus, it does not use information from the application *app_RxBuf[]* buffer. Therefore, is not available to the application designer. It extracts the received FCS (Frame Check Sum) value from the *phs_RxBuf[]* and stores it in the dedicated *unsigned int dll_RecFCS* variable. The structure of the message can be seen in **Figure 4-5**.

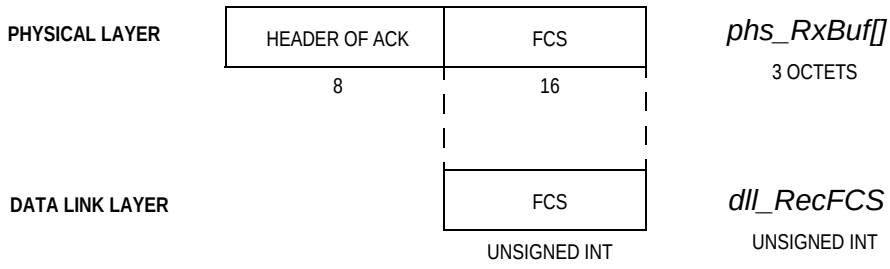


Figure 4-5. Structure of the ACK Datagram Message

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.8 Application Template

In this subsection an application template will be given for both transmission and reception, using routines described in **4.6 API Functions of the PL132 Physical and Data Link Layer**.

For transmission, this simple piece of code demonstrates the usage of the API function *dll_sendKNXData()*.

```
unsigned char tmpCmd[DEF_VAL_LEN];
dll_tTxStatus status;   /* status of the Tx operation */

tmpCmd[0] = 0xFF;                           /* Alive message - 1st byte */
tmpCmd[1] = (unsigned char) dll_nodeAddr;   /* Alive message - 2nd byte */
status = dll_sendKNXData(DEF_VAL_DOA, DEF_VAL_CTRL_NA, dll_nodeAddr,
                         dll_masterAddrInS, DEF_VAL_LEN, tmpCmd);
```

For reception, the following technique is used. Any reception is detected by the *dll_FlgRxDataComp* flag set to 1. As described in **4.7 Buffers and Data Flow Details**, the first byte of the *app_RxBuff[]* buffer carries the data length information. Function style macro *dll_ClearDataFlgs()* clears all reception flags, and has to be called at the end of each data reception handling routine.

```
if (dll_FlgRxDataComp)      /* if frame successfully received */
{
    if ((app_RxBuf[0] == 2) && (app_RxBuf[1] == 0xFF))/* length = 2 and ALIVE command */
    {
       if (app_RxBuf[2] == 0)
           ledA_Rd();                   /* device A alive */
       else if (app_RxBuf[2] == 1)
           ledB_Rd();                   /* device B alive */
       else if (app_RxBuf[2] == 2)
           ledC_Rd();                   /* device C alive */
    }
    dll_ClearDataFlgs();    /* Flag clearing routine for Data Rx */
}
```

4.9 Physical Layer Implementation Details

This subsection is divided into the power line transmission and power line reception parts of the physical layer implementation.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.9.1 Power Line Transmission

For the power line transmission, timer T2 is used. As already shown in **Table 4-2**, T2 overflow interrupt is enabled and T2Ch0 is running in the output compare mode controlling the TxD (PTD6 / T2Ch0) pin. When the counter reaches the value of the output compare channel register, the TIM can (by hardware means) set, clear, or toggle the channel pin. A new bit value is determined by the previous T2 timer overflow ISR. The process is completely interrupt driven and all data edges are generated by TIM hardware. Thus, they appear at the exact time irrespective of any latencies caused, for example, by other interrupt sources.

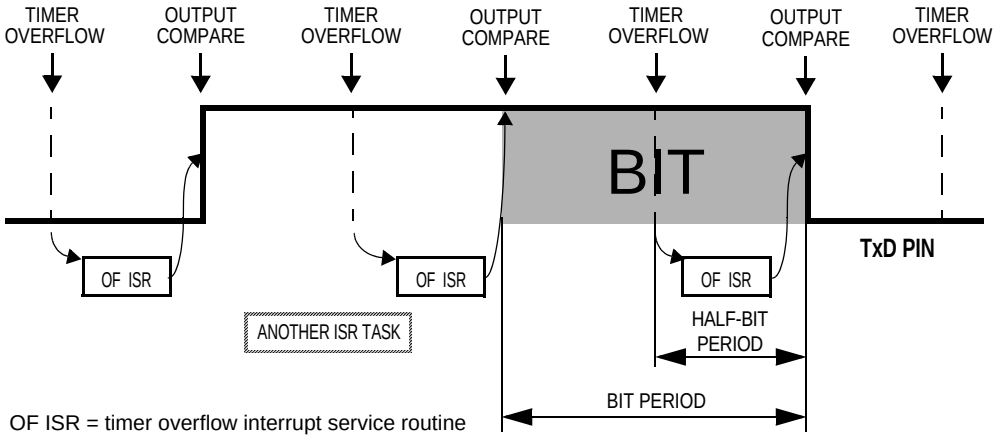


Figure 4-6. Power Line Transmission on the Physical Layer Level

The modulo register T2MOD of the TIM2 module is set to interrupt the process (overflow ISR) at the required bit period (symbolic constant *TIM_PERIOD* defined in **7.3 phs.c**) while the T2CH0 register is set to half of this period (symbolic constant is called *TIM_P_HALF* and defined in **7.3 phs.c**). At 2400 baud, the time period is equal to 416.66 μs and time of the half period is equal to 208.33 μs.

The T2CH0 channel is running in the output compare mode with the 'Set / clear output on compare' feature enabled. The exact value to be sent at the next output compare event is written to the T2SC0_ELS0A bit of the TIM module (therefore, to the PTD4 / T2Ch0, as the TxD pin is connected to the FSK modem device) during the *phs_TxBitISR()* timer T2 overflow ISR. See **Figure 4-7** for the timer 2 ISR *phs_TxBitISR()* flowchart.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

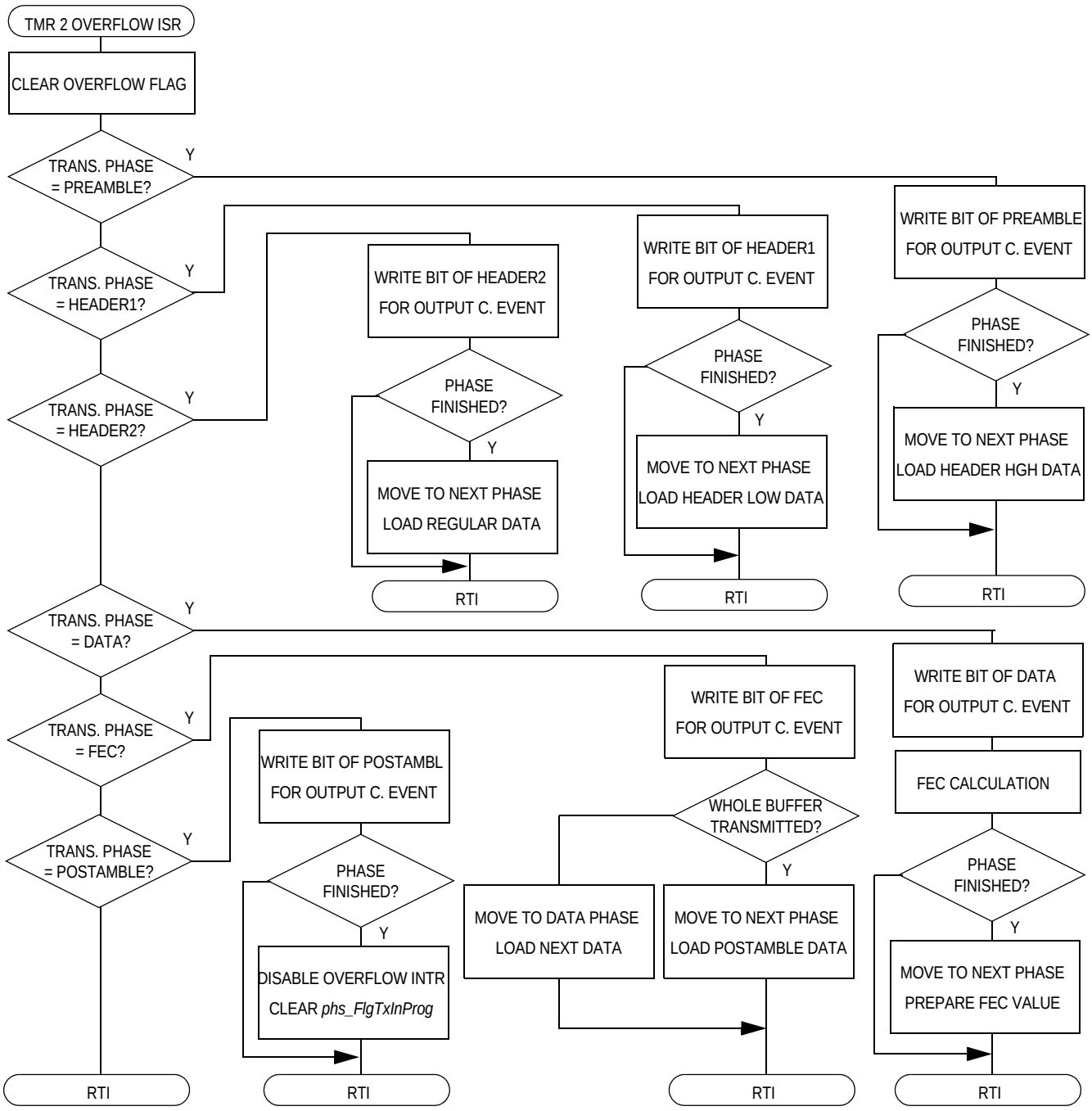


Figure 4-7. Timer 2 Overflow ISR *phs_TxBitISR()* Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

These constants are written to the modulo and output compare registers. They are calculated at compile time by the following series of macros:

in **board.h**

```
/* Timing parameters */
#define XTAL_FREQ  11059200L  /* crystal frequency used on board
*/
#define BUS_CLK     XTAL_FREQ/4
```

in **phs.h**

```
#define KNX_BAUD_RATE 2400L  /* comm. speed over powerline
according to PL132 */
#define TIM_PERIOD   BUS_CLK/KNX_BAUD_RATE
#define TIM_P_HALF   TIM_PERIOD/2
```

The complete transmission process described is controlled by the *phs_Send(unsigned char * pBuf, unsigned char len)* routine of **phs.c**, which is the transmission routine of the physical layer. Parameters of the function are a length of the buffer to be sent and a pointer to the buffer to be sent. This function sets the FSK modem device into the transmission mode, configures timers for the transmission calling function style macro *phs_TmrForTx(),* and starts the transmission itself by enabling timer T2 *phs_StartTxTmr()*. When the transmission is finished (through the transmission in progress *phs_FlgTxInProg* flag), the routine switches transmission off and enables the power line reception again.

NOTE: *Forward error correction (FEC) calculations for each transmitted octet are performed during its transmission in the phs_TxBitISR() routine, stored to tmpFec variable and then sent in its respective place according to the Konnex PL132 format.*

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.9.2 Power Line Reception

For the power line reception, timer T2 and channel 0 of timer T1 are used as well as the keyboard interrupt (KBI) pin PTA0 (see **Table 4-1** and **Table 4-2** for an overview of the MCU resources used).

A falling edge detected on PTA0 pin $\overline{CD}$ (carrier detection of the FSK modem device) starts the power line reception. The $\overline{CD}$ signal is active low and only when $\overline{CD}$ is active can power line reception be performed. Detection itself is implemented as an interrupt service routine (ISR) and is called *phs_CDdetectISR()*. In this routine, the configuration of the reception timers is done by calling the function style macro *phs_TmrForRx().*

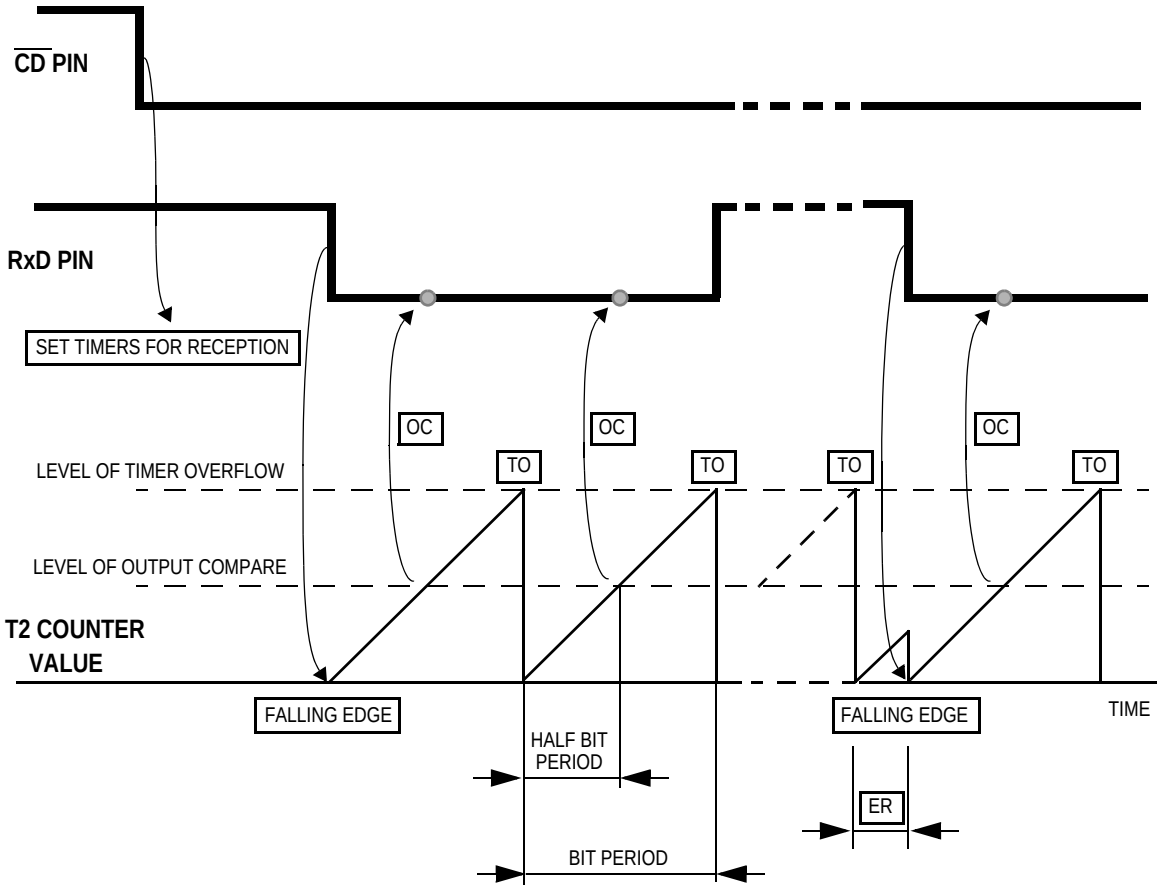
The mission of the reception is to sample the Rx bit stream of the FSK modem device chip, which is connected via the RxD (PTA4 / T1Ch0) pin of the MCU. Knowing that there is no data clock signal available from the FSK modem device, the reception technique is a bit more sophisticated.

The T2CH0 timer channel is running in output compare mode, with enabled interrupt in half-bit period of communication baud rate 2400 baud (symbolic constant *TIM_P_HALF* defined in **7.3 phs.c**). In the interrupt service routine *phs_RxBitISR()* the actual value of the RxD pin is sampled and stored in the *phs_RxBuf[]* buffer. The modulo register T2MOD of the TIM2 module is set to overflow with PL132 bit period (*TIM_PERIOD* defined in **7.3 phs.c**). This technique is used to generate the clock period of 2400 baud from the data stream of the RxD pin. In order to synchronize timer T2 with the incoming data stream, there is a timer T1 channel 0 utilized in the input capture mode with a falling edge detection interrupt service routine called *phs_RxEdgeISR()*. For each edge detected, a reset and consecutive restart of timer T2 is performed.

This approach enables data reception from the bit stream without any clock information. The only assumption is that there are enough edges in the data stream, but this is secured by the Konnex PL132 specification. The whole technique is shown in **Figure 4-8**. A flowchart of the *phs_RxBitISR()* timer 2 channel 0 output compare ISR function can be seen in **Figure 4-9** and **Figure 4-10**.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com



SET TIMERS FOR RECEPTION	Timers are configured for reception by calling *phs_TmrForRx()*
FALLING EDGE	Falling edge detection of timer T1Ch0 resets and restarts timer T2 for bit synchronization; *phs_RxEdgeISR()*
OC	Output compare event of timer T2Ch0 samples an RxD bit value; *phs_RxBItISR()*
TO	Timer overflow event of timer T2 generates the bit period
ER	Possible error eliminated by this synchronization technique

Figure 4-8. Power Line Reception on the Physical Layer Level

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

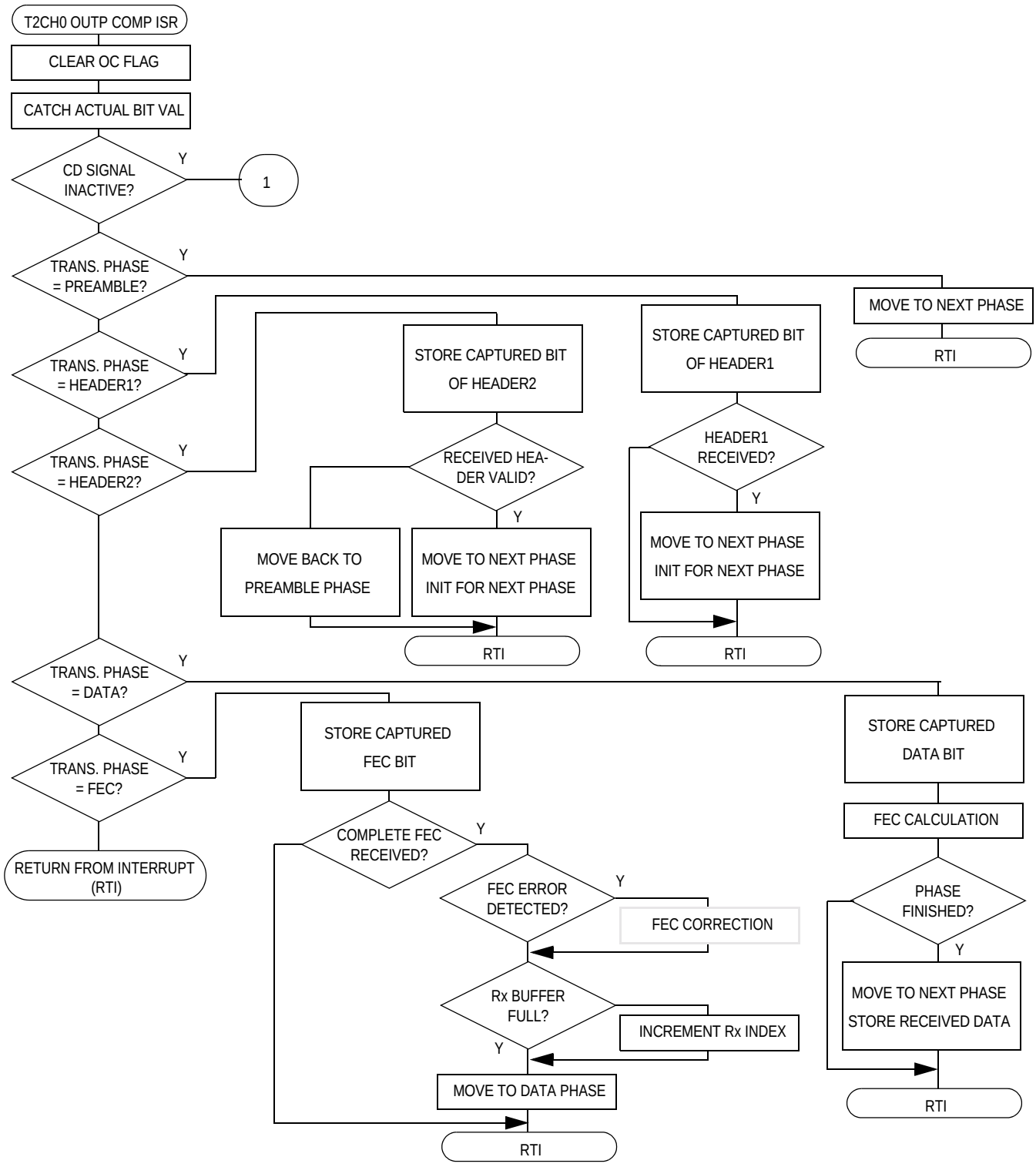


Figure 4-9. Timer 2 Channel 0 Output Compare ISR *phs_RxBitISR()* Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

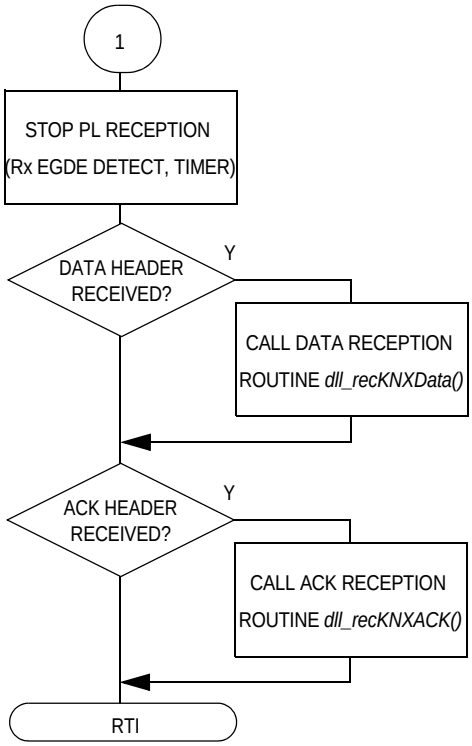


Figure 4-10. Timer 2 Channel 0 Output Compare ISR *phs_RxBitISR()* Flowchart — Second Part

NOTE: *Forward error correction (FEC) calculations for each received octet are performed during the reception itself in the phs_RxBitISR() routine. When FEC error is detected at the end of the octet, the FEC correction routine should be called. However, because of the lack of the details in Konnex PL132 specification, this routine is not implemented.*

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.10 Data Link Layer Implementation Details

All four routines of the data link layer were already introduced in the **4.6 API Functions of the PL132 Physical and Data Link Layer**. More detailed information about two key functions, *dll_sendKNXData()* and *dll_recKNXData()* can be found in this subsection in the form of flowcharts and function headers.

There are also three support functions of the data link layer which are called from the code of above mentioned API functions. More details about them can be found in this subsection as well.

4.10.1 Power Line Transmission — dll_sendKNXData()

For the transmission of the Konnex data frame (Datagram) the *dll_sendKNXData()* function is used. It sends the data message, according to the Konnex PL132 format, to the power line modem on the data link layer level. Refer to the flowchart in **Figure 4-11** and the following API function header field for additional information.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Software Development

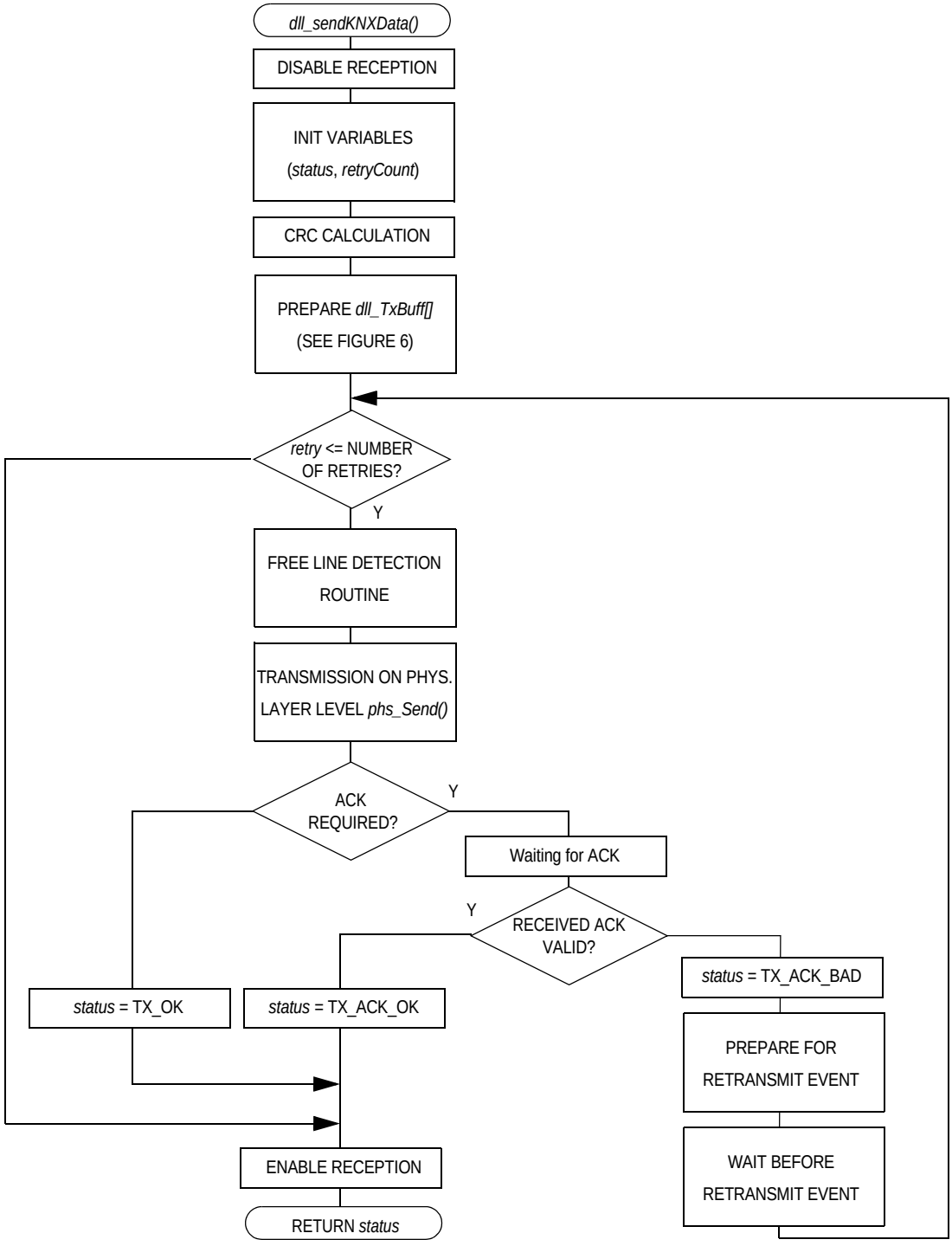


Figure 4-11. *dll_SendKNXData()* Function Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/*****************************************************************************
* Module: dll_tTxStatus dll_sendKNXData(unsigned int doa, unsigned char ctrl,
*                          unsigned int sa, unsigned int da, unsigned char npci,
*                          unsigned char * pAppBuf)
*
* Description: This is the data link layer power line transmission routine.
*   It sends the data message of the PL132 format to the power line modem (ST
*   chip) on the application layer level. Control fields of the data message is
*   taken from the parameters of the function.
*
* Returns:
*   status with the following declaration of dll_tTxStatus type:
*   DLL_TX_INITIAL -   initial state of the variable describing the transmission
*                      on data link layer
*   DLL_TX_OK -        transmission completed on data link layer,
*                      no ACK signal required
*   DLL_TX_ACK_OK -    transmission completed on data link layer,
*                      ACK required => ACK message received succesfully
*   DLL_TX_ACK_BAD -   transmission not completed on data link layer since
*                      no ACK from 1st initial + 2 retransmit transmissions
*   DLL_TX_BUSOCC -    transmission not completed on data link layer since
*                      unexpected bus occupation during retransmits
*
* Arguments: doa - Domain Address (DOA)
*            ctrl - Control Field (CTRL)
*            sa - Source Address (SA)
*            da - Dource Address (DA)
*            npci - Network Protocol Control Information (NPCI)
*            pAppBuf - pointer to application transm. buffer
*
* Range Issues: Note that the implementation supports only short frames format
*   of the Konnex PL132 specification
*
* Special Issues: Only the following bits are taken from the ctrl parameter of
*       the function, rest are set in the routine itself:
*   unsigned char ctrlGroupAddr : 1;     0 - individual frame
*                                        1 - group frame
*   unsigned char ctrlAckReq    : 1;     0 - no Layer 2 ack requested
*                                        1 - Layer 2 ack requested
*   unsigned char ctrlPriority :  2;    11 - low (mandatory for long frames)
*                                       01 - normal (defaul for short frames)
*                                       10 - urgent (reserved for urgent frames)
*                                       00 - system (reserved for high priority
*                                            system config + management)
*
*****************************************************************************/
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.10.2 Power Line Reception — dll_recKNXData()

The *dll_recKNXData()* function is dedicated to the reception of the Datagram. It gets the message in the physical layer format, as stored in *phs_RxBuf[]*, then it checks the address and the control FCS part of the message. If the message requires an acknowledgement, it also sends the Datagram ACK message. If both address and FCS are correct, flag *dll_FlgRxDataComp* (reception of the data message completed) is set for the application layer level, and the data is moved to the application buffer *app_RxBuf[]*.

Refer to the flowchart in **Figure 4-12** and the following API function header field for additional information.

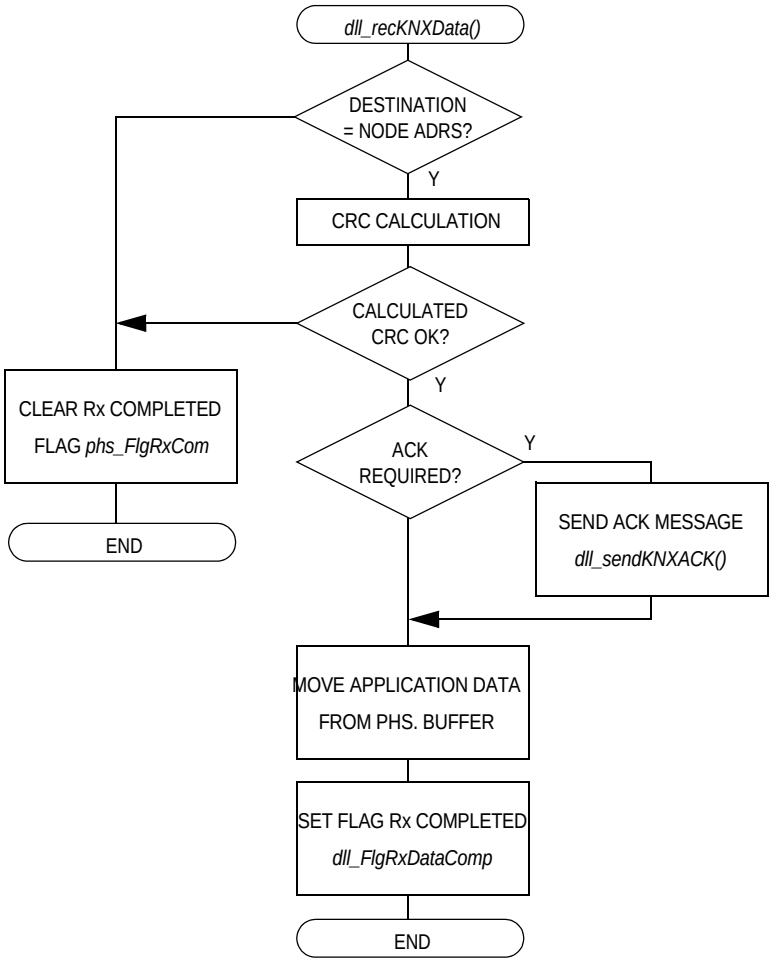


Figure 4-12. *dll_recKNXData()* Function Flowchart

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
/*****************************************************************************
* Module: void dll_recKNXData(void)
*
* Description: This is the data link layer power line reception routine.
*   It gets the message in physical layer format (phs_RxBuf) and check address,
*   control FCS part of the message; if message requires an acknowledge, it
*   also sends the ACK message.
*   If both address and FCS are correct, flag dll_FlgRxDataComp (Reception
*   of Data message completed) is set for the application layer level and data
*   are moved to the application buffer app_RxBuf[].
*   Note that only Destination Address (DA) field is checked, the Source Address
*   (SA) and Domain Addres (DOA) are not checked.
*
* Returns: None
*
* Arguments: None
*
* Range Issues: Note that the implementation supports only short frames format
*   of the Konnex PL132 specification
*
*****************************************************************************/
```

4.10.3 Support Functions

As mentioned, there are also three data link layer support functions which are called from the aforementioned API functions. They are:

- *dll_CalcCRC(unsigned char *buf, unsigned char n)* is used for the Cyclic Redundancy Code (CRC) computation

NOTE: *Konnex PL132 specification calls this part of the frame the Frame Check Sequence (FCS), although the calculation uses a 16-bit Cyclic Redundancy Code algorithm.*

- *dll_Init()* is an initialization function of the data link layer, as well as the application addresses; for more about this topic see **4.11.1.1 Application Address Structure Details**
- *dll_TBModuleISR()* is an interrupt service routine of the Timebase Module (TBM), used for the timing of the data link layer

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.10.3.1 CRC Computation

According to the PL132 specification, the Cyclic Redundancy Code (CRC) method is used to verify the integrity of every frame sent. An additional field is added to every data block at the time of transmission, and then it is checked at the time of reception for correctness. The following 16-bit CRC polynoms called CRC-16 is used in the Konnex PL132 over power line based on M68HC08 project: $x^{16}+ x^{15}+ x^{2}+ 1$.

A lookup table computation algorithm has been chosen to implement the CRC calculation. A table *static const int tableCRC[256]*, located in the **dll.c** file, is stored in the program memory area. The CRC calculation routine is as follows.

```
/*****************************************************************************
* Module: void dll_CalcCRC(char *buf, unsigned char n)
*
* Description:
*       This function generates the 16 bit CRC      16    15   2
*       using the following polynom:               X   + X  + X  + 1
*       Precise CRC computation algorithm definition:
*       - CRC computation algorithm starts with zero
*       - it treats the data msb first
*       - CRC result is not complemented
*
* Returns: calculated CRC value
*
* Global Data:
*       tableCRC[256] - look-up table for 16 bit CRC computation
*
* Arguments:
*       *buffer - pointer to buffer to be calculated
*       n - length of the buffer
*
*****************************************************************************/
unsigned int dll_CalcCRC(unsigned char *buf, unsigned char n)
{
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
  unsigned int crc = 0;
#pragma DATA_SEG DEFAULT

  while (n--)
     crc = tableCRC[((crc>>8) ^ *buf++) & 0xFF] ^ (crc << 8);
  return (crc);
}
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.10.3.2 Data Link Layer Timing Details

The *dll_TBModuleISR()* interrupt service routine of the TBM is set to interrupt the process circa every 0.75 ms (8192/11.0592E6). The TBM is both configured and started by a *dll_SetStartTBM()* function style macro, called from the *dll_Init()* routine.

The ISR routine is used for the timing of the following events (according to the Konnex PL132 specs):

- Period of the Free line detection (as defined in the CENELEC EN 50065-1 norm) during the data frame transmission. This period is between 85 ms and 115 ms long with at least seven possible values. This state is noted in the *dll_FlgFreeLineDet* flag.
- Period of the Waiting for ACK (acknowledge) during data frame transmission (period is 35 ms long). This state is noted in the *dll_FlgWaitForACK* flag
- Period of the Waiting before retransmission during the data frame transmission (period is between 0 ms and 30 ms long with 7 steps of roughly 4 ms). This state is noted in the *dll_FlgWaitBefRetr* flag.

Time period generation is started by setting the respective flag and by filling the *dll_DesiredDelay* variable according to the desired time period. As a random value of the generation process, the *dll_nevEndCount* variable is used. For example, when there is a need for seven different 5 ms long values between 0 ms and 30 ms, simple modulo division is used as shown below.

```
/* it generates delay period 0 - 30ms a 5ms; */
dll_DesiredDelay = (dll_nevEndCount % 7) * DLL_DELAY_5MS;
```

NOTE: *This routine is also used for some application related timing; for more see* ***4.11 Demo Application Implementation Details****.*

The *dll_TBModuleISR()* interrupt service routine flowchart is shown in **Figure 4-13**.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

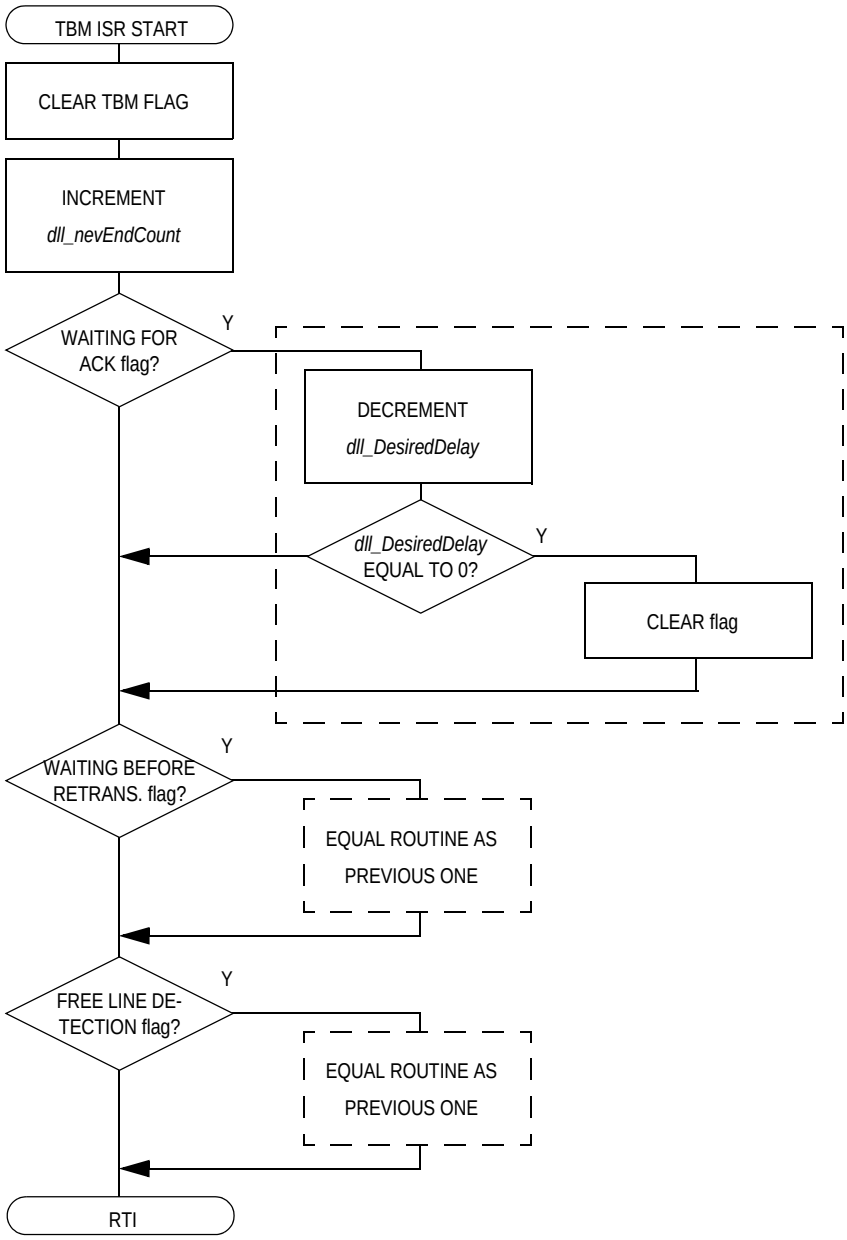


Figure 4-13. Time Base Module (TBM) ISR Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.11 Demo Application Implementation Details

This subsection briefs the application layer implementation. As already mentioned, there are two different versions of the software; one for a master device and one for a slave device (controlled lamp). While the master device application layer is taking care of a demo user interface, the application of the slave device is primarily controlling the triac of the lamp.

4.11.1 Structure of the Messages in the Application

A general description of both application buffers is provided in **4.7 Buffers and Data Flow Details**. From this point on, any description given is related to the demo application itself.

The structure of the application messages, which are valid for both transmission and reception buffers, is described in **Table 4-4**. All messages used in the demo application have a message length equal to two octets (the length of received messages are stored in *app_RxBuf[0]*).

Table 4-4. Application Messages Description

First Byte Value	Second Byte Value	Message Description	Sender	Recipient
0x0	Whatever	SWITCH OFF	Master	Slave
0x1	Light intensity	SWITCH ON to intensity given in second byte	Master	Slave
0xFF	Address of sender	ALIVE MESSAGES	Slave	Master

NOTE: *The higher the app_RxBuf[2] byte of a message the lower the light intensity.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.11.1.1 Application Address Structure Details

Each slave device has its own address called *dll_nodeAddr*, which is taken from the “wired” address value of the device (pins PD0, PD1, PD2, and PD3 — see **Table 4-1**). It also has the address of its master stored in *dll_masterAddrInS*.

The master device also has two types of address; its own is stored in the *dll_masterAddr* variable and the slaves address information is stored in array variable *dll_nodeAddrInM[3]*. Refer to the code listing below, taken from *dll_Init()* function of the **dll.c** file, for default values of above described variables.

```
#ifdef SLAVE
    dll_nodeAddr = NODE_ADDR;   /* set address of slave node */
        /* address is read from the address configuration on 4 pins of port D */
    dll_masterAddrInS = 0xFF;   /* set address of the master node */
#endif
#ifdef MASTER
    dll_masterAddr = 0xFF;  /* set address of master node */
    dll_nodeAddrInM[0] = 0; /* set address of slave node No. 1 */
    /* button "A" control device with Address = 0 */
    dll_nodeAddrInM[1] = 1; /* set address of slave node No. 2 */
    /* button "B" control device with Address = 1 */
    dll_nodeAddrInM[2] = 2; /* set address of slave node No. 3 */
    /* button "C" control device with Address = 2 */
#endif
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.11.2 Introduction to “Alive” Messages

An “Alive” messages technique is used in the system in order to enable the master device to detect when and which slave devices are connected and on-line. One message type is dedicated for this purpose. Those messages are sent by the slave devices and received by the master device.

Timing of the “Alive” message transmission of the slave devices is based on the *dll_TBModuleISR()* routine. Dedicated *app_AliveCount* variable is incremented in a 191 ms period. When its value reaches some specific threshold, a new value *APP_SEND_ALIVE* is written to it.

```
if (dll_nevEndCount == 0xFF)    /* do each 0.75ms * 255 = 191ms */
{
    app_AliveCount++;   /* 1 tick in counter is approx. 191ms */
    if (app_AliveCount>=APP_ALIVE_SLV_LIMIT+(NODE_ADDR*APP_ALIVE_SLV_PRIOR))
    {
        app_AliveCount = APP_SEND_ALIVE;
    /* set counter to "Send alive message, message will be send in main() */
    }
}
```

NOTE: *There is a prioritization included in the frequency of the transmission, based on the address of the Slave device.*

When an *APP_SEND_ALIVE* value is detected in the main() routine counter, the counter has to be cleared and an “Alive” message is sent.

```
    if (app_AliveCount >= APP_SEND_ALIVE) /* condition for "device alive" */
    {
        app_AliveCount = 0;         /* clear counter */
        status = app_SendAlive();   /* send alive message */
    }
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.11.3 Slave Device Application Implementation Details

The Slave device application flowchart can be found in **Figure 4-14**.

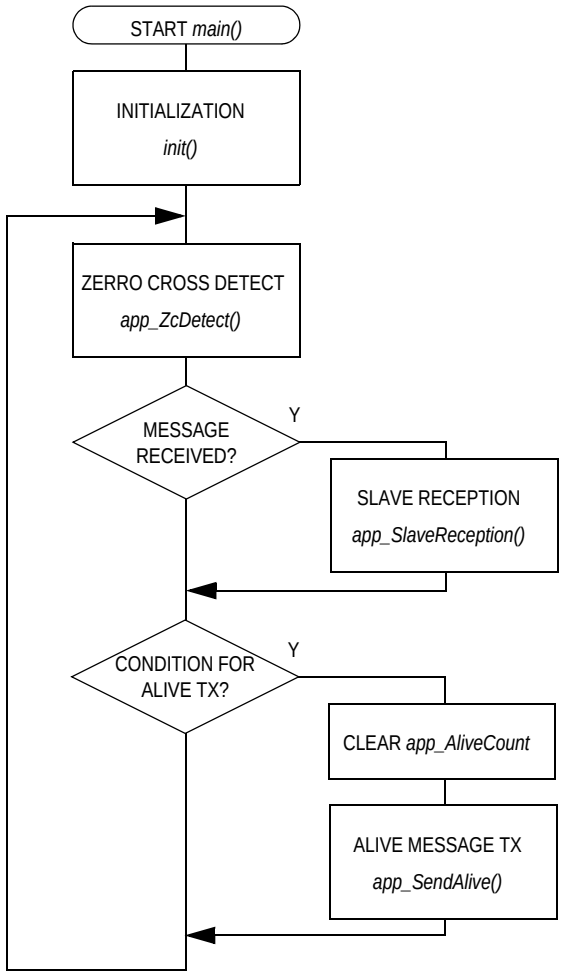


Figure 4-14. Slave Device Application Flowchart

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.11.3.1 Triac Control Implementation Basics

Standard triac control technique of the lamp is used. It is based on an output compare interrupt service routine of timer 1 channel 1. Timer 1 overflow is set to half period of the AC line, while the value of the output compare register is set according to the desired light intensity. The lower the light intensity value, the longer the portion of the AC line half period that the triac is switched on.

Although the half period of the AC line is generated by timer 1 overflow, it is necessary to use the zero cross detection routine *app_ZcDetect()*. This routine synchronizes timer 1 with the AC line using the *APP_ZC_INP* (PTA1) signal (see **Figure 4-15**). It was intended to connect this signal to a pin with edge interrupt capability; however, in this generic design that was not the case. Therefore, the signal has to be polled in code, as noted in **Figure 4-14**.

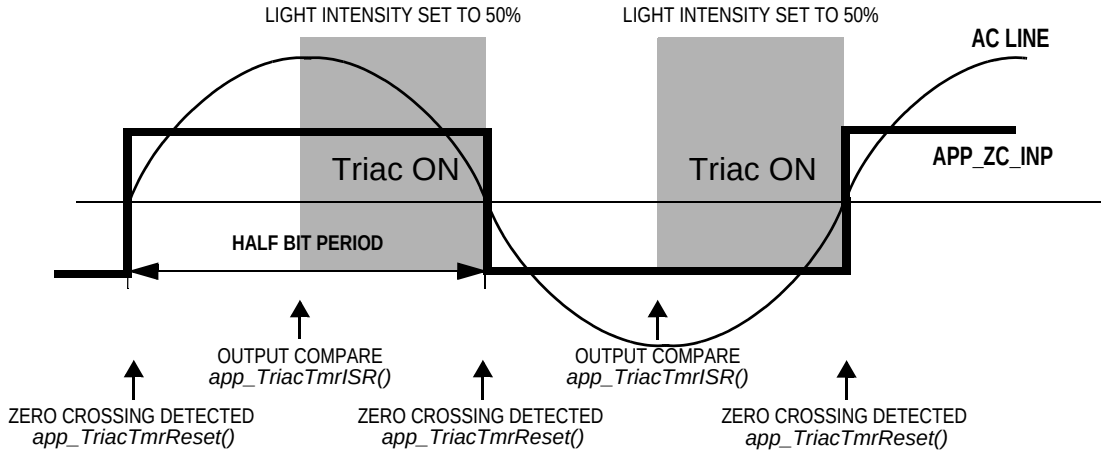


Figure 4-15. Zero Cross Detection Technique and Triac Control

```
void app_ZcDetect()
{
    static unsigned char wasZC; /* zero cross temp variable */

    if (wasZC)       /* Zero crossing detection routine */
    {
        if (APP_ZC_INP)     /* from negative to positive zero cross */
        {
            wasZC=0;        /* set temp zero cross variable */
            app_TriacTmrRst();  /* synchronize Triac control timer */
        }
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
    }
    else
    {
        if (!APP_ZC_INP)    /* from positive to negative zero cross */
        {
            wasZC=1;        /* set temp zero cross variable */
            app_TriacTmrRst();  /* synchronize Triac control timer */
        }
    }
}
```

Therefore, there are three possible commands for the triac:

1. Desired intensity lower than *POT_DELTA* threshold switches the dimmer to forever ON

```
        if (app_RxBuf[2] <= POT_DELTA)   /* SWITCH ON forever command when full
                                            intensity set */
        {
            app_TriacTmrID();          /* disable TRIAC switch ON routine*/
            app_TriacBurstOnForever(); /* and switch TRIAC ON forever */
        }
```

2. Desired intensity higher than *(0xFF - POT_DELTA)* threshold switches the dimmer to OFF state

```
        else if (app_RxBuf[2] >= 0xFF - POT_DELTA)
        {                       /* SWITCH OFF command when zero intensity set */
            app_TriacTmrID();   /* disable TRIAC switch ON routine*/
            triacOff();         /* switch off triac */
        }
```

3. Otherwise, T1Ch1 output compare registers are set according to the desired intensity

```
        else                    /* SWITCH to desired intensity */
        {
            app_desAnalogVal = app_RxBuf[2]; /* desired value received in msg */
            app_TriacTmrIE();   /* enable TRIAC switch ON routine */
        }
```

NOTE: *The values of the desired light intensity are not directly written to the T1CH1 output compare register, but they are linearly approximated based on time periods of TBM periphery. Intensity value is incremented / decremented by one each 4.5 ms, until the value is not equal to the desired one. For more information, see the dll_TBModuleISR() routine.*

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.11.3.2 Slave Reception Routine

The implementation of the slave reception routine *app_SlaveReception()* is as follows. It simply checks the type of the received message (see **Table 4-4** for message description) and acts according to the received information.

```
void app_SlaveReception(void)
{
    if ((app_RxBuf[0] == 2) && (app_RxBuf[1] == '1'))
    {                            /* length = 2 and SWITCH ON command */
        if (app_RxBuf[2] <= POT_DELTA)   /* SWITCH ON forever command when full
                                            intensity set */
        {
            app_TriacTmrID();   /* disable TRIAC switch ON routine*/
            app_TriacBurstOnForever(); /* and switch TRIAC ON forever */
        }
        else if (app_RxBuf[2] >= 0xFF - POT_DELTA)
        {                       /* SWITCH OFF command when zero intensity set */
            app_TriacTmrID();   /* disable TRIAC switch ON routine*/
            triacOff();         /* switch off triac */
        }
        else                    /* SWITCH to desired intensity */
        {
            app_desAnalogVal = app_RxBuf[2]; /* desired value received in msg */
            app_TriacTmrIE();   /* enable TRIAC switch ON routine */
        }
        ledIndOn();
    }
    if ((app_RxBuf[0] == 2) && (app_RxBuf[1] == '0'))
                                /* length = 2 and SWITCH OFF command */
    {
        app_TriacTmrID();   /* disable TRIAC switch ON routine */
        triacOff();         /* switch off triac */
        ledIndOff();
    }
    dll_ClearDataFlgs();    /* Flag clearing routine for Data Rx */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

4.11.4 Master Device Implementation Details

Refer to **Figure 4-16** for a flowchart of the Master device application.

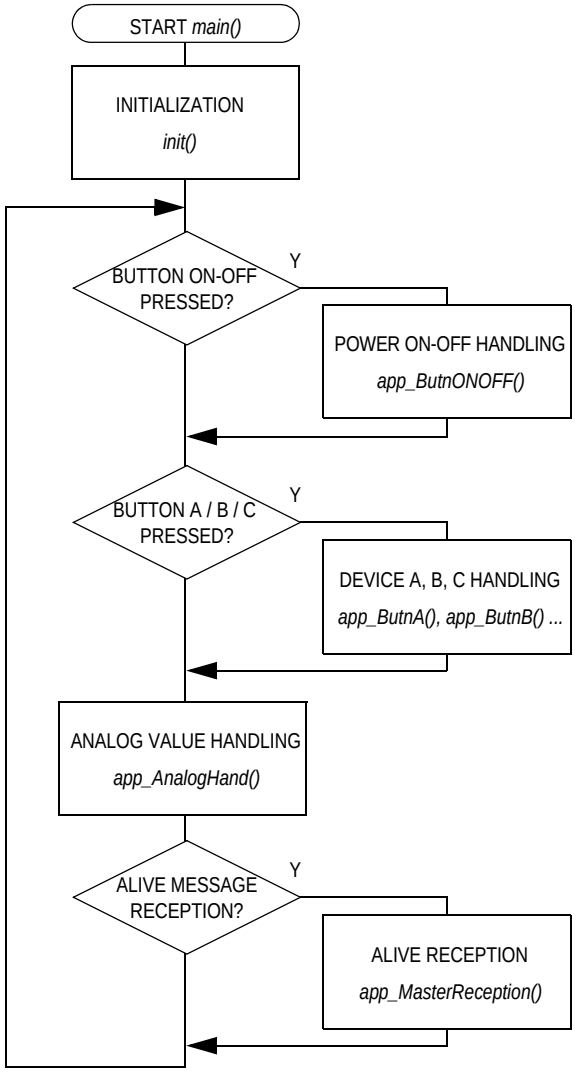


Figure 4-16. Master Device Application Flowchart

There are two important variables describing the state of the connected Slave devices and a currently selected device.

```
unsigned char app_indexOfDev;       /* index for State of devices */
                                    /* index = 0, 1 or 2 */
unsigned char app_deviceState[3];   /* state of devices [lamp is ON/OFF] */
                                    /* state[] = 0 - lamp in device is OFF
                                       state[] = 1 - lamp in device is ON */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.11.4.1 Power ON-OFF Button Handling Routine

This routine is the reaction to the Power ON-OFF button pressed event. It sends an ON or alternatively OFF command message to switch the selected slave device between those two states. Active slave is a device which is stored in the *app_indexOfDev* variable.

This type of message requires an acknowledgement (ACK); this setting is passed in the second function parameter (symbolic constant *ACK_ON)*. So, only when an ACK Datagram is received is the state of the internal *app_deviceState[]* variable modified and the Power ON-OFF LED state updated.

NOTE: *app_Send() function is the user defined transmission routine based on a standard dll_sendKNXData() function, as described in* ***4.6 API Functions of the PL132 Physical and Data Link Layer****.*

Each received ACK message also clears its relevant byte in app_AliveCountInM[], as it was the "Alive" type of message. For more information about this topic see ***4.11.2 Introduction to "Alive" Messages****.*

```
void app_ButnONOFF(void)
{
    dll_tTxStatus status;       /* status of the transmission */

    if (app_deviceState[app_indexOfDev] == 0)   /* if OFF */
    {
        app_TxBuf[0] = '1';                 /* send ON command */
        app_sentAnalogVal = POT_INT_CTRL;   /* store analog value */
        app_TxBuf[1] = POT_INT_CTRL;        /* write analog value into message*/
        status = app_Send(app_indexOfDev, ACK_ON, app_TxBuf);
                                            /* send message with ACKs enabled */
        if (status == DLL_TX_ACK_OK)        /* if message ACKed */
        {
            app_deviceState[app_indexOfDev] = 1;    /* set ON */
            ledPowOn();                     /* LED indication is ON */

            app_AliveCountInM[app_indexOfDev] = 0;  /* clear counter */
            switch (app_indexOfDev)
            {                               /* refresh LED states */
                case 0 :    ledA_Rd();
                            break;
                case 1 :    ledB_Rd();
                            break;
                case 2 :    ledC_Rd();
                            break;
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
            }
        }
    }
    else                                    /* if ON */
    {
        app_TxBuf[0] = '0';                 /* send OFF command */
        app_sentAnalogVal = POT_INT_CTRL;   /* store analog value */
        app_TxBuf[1] = POT_INT_CTRL;        /* write analog value into message*/
        status = app_Send(app_indexOfDev, ACK_ON, app_TxBuf);
                                            /* send message with ACKs enabled */
        if (status == DLL_TX_ACK_OK)        /* if message ACKed */
        {
            app_deviceState[app_indexOfDev] = 0;    /* set OFF */
            ledPowOff();                    /* LED indication is OFF */

            app_AliveCountInM[app_indexOfDev] = 0;  /* clear counter */
            switch (app_indexOfDev)
            {                               /* refresh LED states */
                case 0 :    ledA_Rd();
                            break;
                case 1 :    ledB_Rd();
                            break;
                case 2 :    ledC_Rd();
                            break;
            }
        }
    }
    while (SW_ON_OFF);      /* wait for button release */
    app_delay();            /* button delay routine */
}
```

4.11.4.2 Device Buttons Handling Routine

Pressing the Device A, B, or C buttons only passes respective index information into the *app_indexOfDevice* variable. Index for Device A is equal to 0, 1 for Device B, and 2 for Device C.

Lamp state (switched ON or OFF) of current (last pressed) device is indicated by the LED indication of Power ON-OFF button.

```
void app_ButnA(void)
{
    app_indexOfDev = 0;         /* set index "Which slave device is chosen?" */
                                /* Device "A" is chosen */
    if (app_deviceState[app_indexOfDev] == 1)   /* if device is ON */
        ledPowOn();             /* set LED indication to ON */
    else ledPowOff();           /* else set it to OFF state */
    while (SW_CTRL_DEV_A);      /* wait for button release */
}
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.11.4.3 Potentiometer Analog Value Handling

When the knob value (light intensity analog value) is changed from the last sent value, within certain limits given by *POT_DELTA* symbolic constant, and when the device is in the switched ON state this routine automatically sends an ON command message with actual analog value to the active slave device.

```
void app_AnalogHand(void)
{
    unsigned char hiLimit;      /* for analog value processing */
    unsigned char loLimit;      /* for analog value processing */
    dll_tTxStatus status;       /* status of the transmission */

    if (app_sentAnalogVal < (0xFF - POT_DELTA))  /* set analog high limit val*/
        hiLimit = app_sentAnalogVal + POT_DELTA; /* calculated high limit val*/
    else hiLimit = 0xFF;                         /* limited high limit value */

    if (app_sentAnalogVal > POT_DELTA)           /* set analog low limit */
        loLimit = app_sentAnalogVal - POT_DELTA; /* calculated low limit val */
    else loLimit = 0;                            /* limited low limit value */

    if ((POT_INT_CTRL > hiLimit) || (POT_INT_CTRL < loLimit))
    {
        if (app_deviceState[app_indexOfDev] == 1)  /* only if Device is ON! */
        {
            app_TxBuf[0] = '1';     /* send ON command */
            app_TxBuf[1] = app_sentAnalogVal = POT_INT_CTRL;
                                    /* store analog value */
            status = app_Send(app_indexOfDev, ACK_OFF, app_TxBuf);
                                    /* send message with ACKs disabled */
        }
    }
```

4.11.4.4 Master Reception Routine of “Alive” Messages

The master device has a dedicated array variable called *app_AliveCountInM[3]*, one byte for each slave device. It is a counter used for a time-out generation in the master. The principle is that each successful reception of an “Alive” message from a relevant slave device clears its counter, while some given timing event (based on TBM interrupt) increases the counter value. So when the value reaches a given limit, it means that the respective slave device is no longer connected. This state is indicated in its LED state; red color stands for a non-connected device, while green means the device is on-line.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

The following piece of code is the master reception routine of “Alive” messages. For information about possible types of messages see **Table 4-4**.

```
void app_MasterReception(void)
{
    if ((app_RxBuf[0] == 2) && (app_RxBuf[1] == 0xFF)) /* length = 2 and ALIVE command */
    {
        if (app_RxBuf[2] == 0)
        {
            ledA_Rd();                  /* device A alive */
            app_AliveCountInM[0] = 0;   /* clear counter */
        }
        else if (app_RxBuf[2] == 1)
        {
            ledB_Rd();                  /* device B alive */
            app_AliveCountInM[1] = 0;   /* clear counter */
        }
        else if (app_RxBuf[2] == 2)
        {
            ledC_Rd();                  /* device C alive */
            app_AliveCountInM[2] = 0;   /* clear counter */
        }
    }
    dll_ClearDataFlgs();       /* Flag clearing routine for Data Rx */
}
```

The following code is part of the *dll_TBModuleISR()* TBM interrupt service routine. Counters are incremented approximately each 191 ms. When the counter value reaches a level given by the symbolic constant *APP_ALIVE_MSTR_LIMIT*, the respective device is considered as no longer connected.

```
/* Alive message handling for master side */
    if (dll_nevEndCount == 0xFF)    /* do each 0.75ms * 255 = 191ms */
    {
       for (i = 0; i < 3; i++)
        {
            app_AliveCountInM[i]++;   /* 1 tick in counter is approx. 191ms */
            if (app_AliveCountInM[0] >= APP_ALIVE_MSTR_LIMIT)   /* device A  */
                ledA_NR();                                      /* NOT READY */
            if (app_AliveCountInM[1] >= APP_ALIVE_MSTR_LIMIT)   /* device B  */
                ledB_NR();                                      /* NOT READY */
            if (app_AliveCountInM[2] >= APP_ALIVE_MSTR_LIMIT)   /* device C  */
                ledC_NR();                                      /* NOT READY */
        }
    }
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

4.11.5 Parts of Konnex PL132 Specification Not Implemented

This subsection describes the parts of the Konnex PL132 specification which are not implemented in this generic demo project, mainly due to a lack of PL132 specific information.

- FEC correction when FEC error detected in *phs_RxBitISR()* routine (in **7.3 phs.c**)
- CRC calculation is implemented and fully functional, but because of the vague specification of all its parameters in the PL132 definition, the calculation might be configured improperly. For more details see *dll_CalcCRC()* routine header (in **7.5 dll.c**)
- End of the packet detection is currently based on the detection of a non-active CD signal and it is not obvious if it is the correct method.
- After transmission delay not implemented
- Implementation only supports short frames of the Konnex PL132 specification (up to 15 octets of data)

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Software Development

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 5. Hardware Design Description

5.1 Contents

5.2 Introduction

This hardware design provides a set of devices able to show the key parts of the implementation of a power line modem according to the Konnex PL132 specification. Beyond that, the boards enable the implementation and testing of the user software.The basic kit consists of the Master and Slave device.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.3 Master Device Architecture

The Master device block diagram can be seen in **Figure 2-2. Master Device Block Diagram**. The electrical circuitry can be logically divided into the following basic blocks:

- Power stage and coupling module
- FSK modem
- Microcontroller
- Power module

5.4 Power Stage and Coupling

The basic power stage and coupling network can be seen in **Figure 5-1**. The coupling network is the interface between the power line and the low-voltage transmitter output and receiver input pins of the modem.

The Master device serves as a human interface and insulation from the mains is mandatory. The HF transformer (T1) is used for this purpose. Apart from the insulation with the power line, the transformer also has to perform the appropriate filtering for both the transmission and the reception. The TOKO's TK1903-ND transformer which has two primary windings and one secondary winding is used for this application. The ratios of the windings are 4:1:1 (turns). The primary windings of the transformer are used to create a bandpass filter. The resonance frequency is set at the transmit frequency with C30.

The coupling capacitors C28 and C31 are used to couple the modem with the power line and they must be an X2 type, rated for mains voltage. Resistor R26 serves to discharge C28 and C31 when the device is disconnected from the power line. Varistor D13 provides protection against high-voltage transients on the power line.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

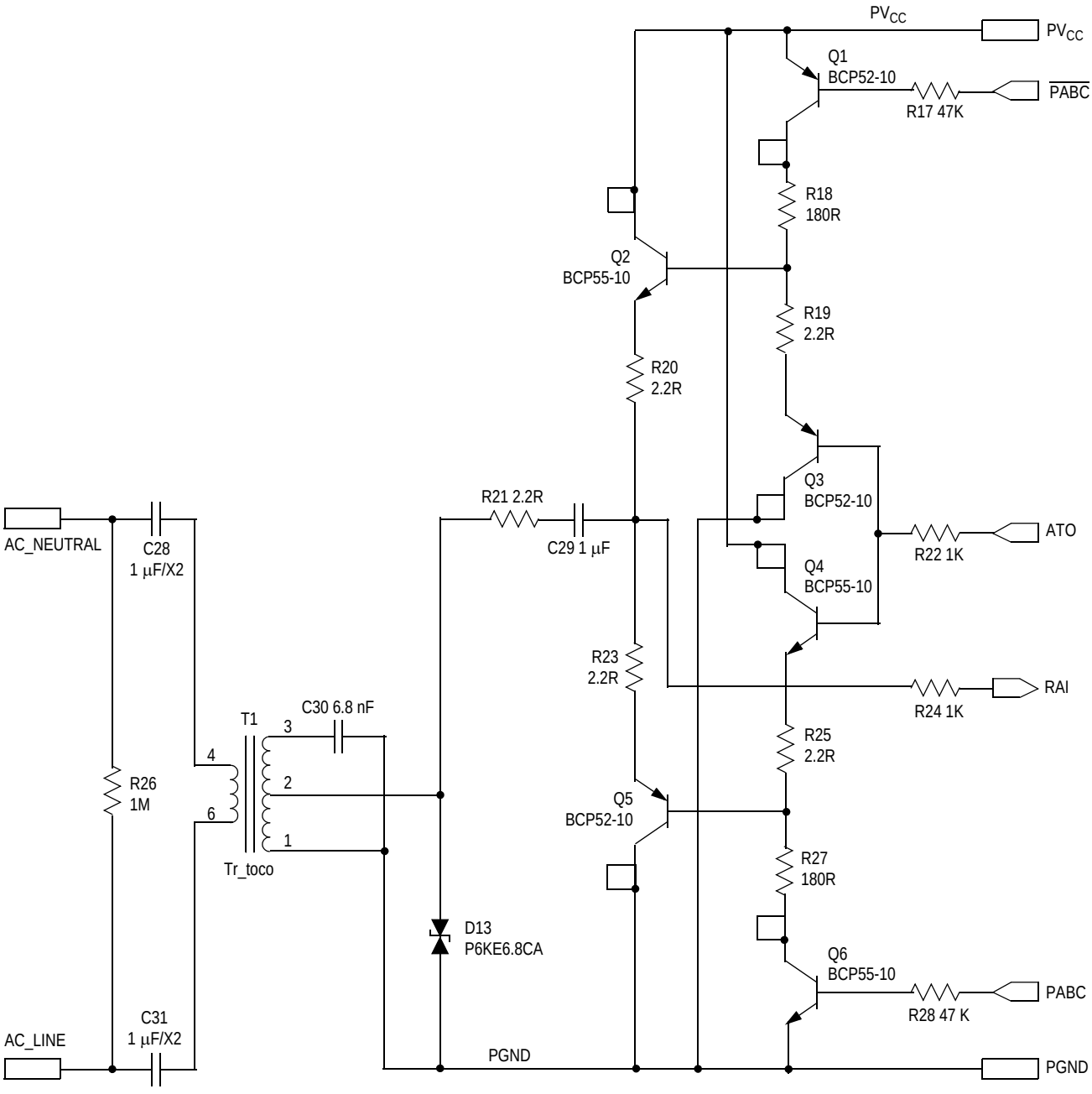


Figure 5-1. Power Stage and Coupling

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The power line interface has two operation modes: transmit and receive. By default the system is in receive mode. In receive mode the transformer extracts the signal from the power line and sends it to the input (RAI) of the FSK modem device. The power amplifier is switched off in receive mode, in order to avoid the low output impedance of the power amplifier attenuating the received signals. Two outputs (PABC and $\overline{\text{PABC}}$) of the FSK modem device, delivering a signal between 0 V and 10 V, are driven low (PABC) and high ($\overline{\text{PABC}}$) respectively, when the circuit is set in the receive mode.

In transmit mode the power stage amplifies the transmit signal (ATO) from the FSK modem device. The power amplifier must drive power lines with impedances from 1 to 100 Ohm, via the transformer.

5.5 FSK Modem

A half-duplex asynchronous FSK modem, ST7537HS1, is used for this application. Its data transmission rate is 2400 bps on a carrier frequency 132.45 kHz. It requires two power supplies 10 V for analog signals and 5 V for digital interface to the microcontroller.

The schematic of the FSK modem can be seen in **Figure 5-2**.

All timing is derived from a crystal oscillator X2 (11.0592 MHz). The output signal MCLK delivers a clock signal for the microcontroller. FSK modem is controlled and monitored by the microcontroller. As described above, the modem can operate in receive or transmit mode. The transmit mode is set when Rx or Tx selection mode signal Rx/$\overline{\text{Tx}}$ = 0. If Rx/$\overline{\text{Tx}}$ is held at 0 longer than 1 second, then the device switches automatically in the Rx mode. In Tx mode the receive data (RxD) signal is set to 1.

The transmit data (TxD) enters asynchronously the FSK modulator with a data rate of 2400 bps from the microcontroller. The modulated signal comes out through the analog transmit output (ATO) pin.

The receive section is active when Rx/Tx = 1. The Rx signal is applied on receive analog input (RAI) pin. The receive data output (RxD) delivers the demodulated signal if the carrier detect ($\overline{\text{CD}}$) signal is low, and is set to high level when $\overline{\text{CD}}$ = 1.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

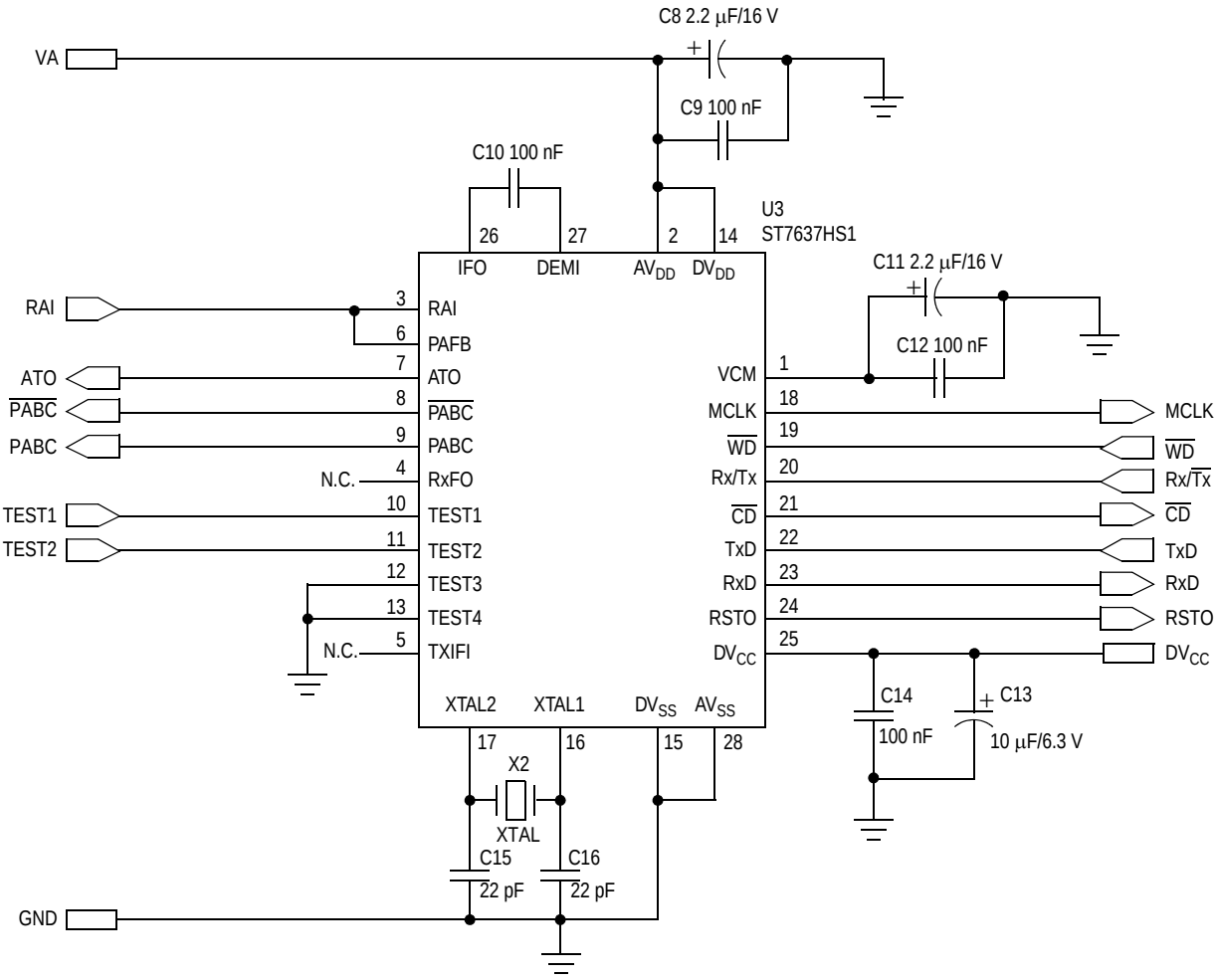


Figure 5-2. FSK Modem

The reset output (RSTO) is driven high when the supply voltage is lower than typically 7.6 V, or when no negative transition occurs on the watchdog input ($\overline{WD}$) for more than 1.5 seconds. When a reset occurs, RSTO is held high for at least 50 ms.

5.6 Microcontroller

Motorola's 8-bit MC68HC908GR8 (U1) microcontroller controls the master PLM board. The microcontroller schematic diagram can be seen in **Figure 5-3**.

For More Information On This Product,
Go to: www.freescale.com
Freescale Semiconductor, Inc.

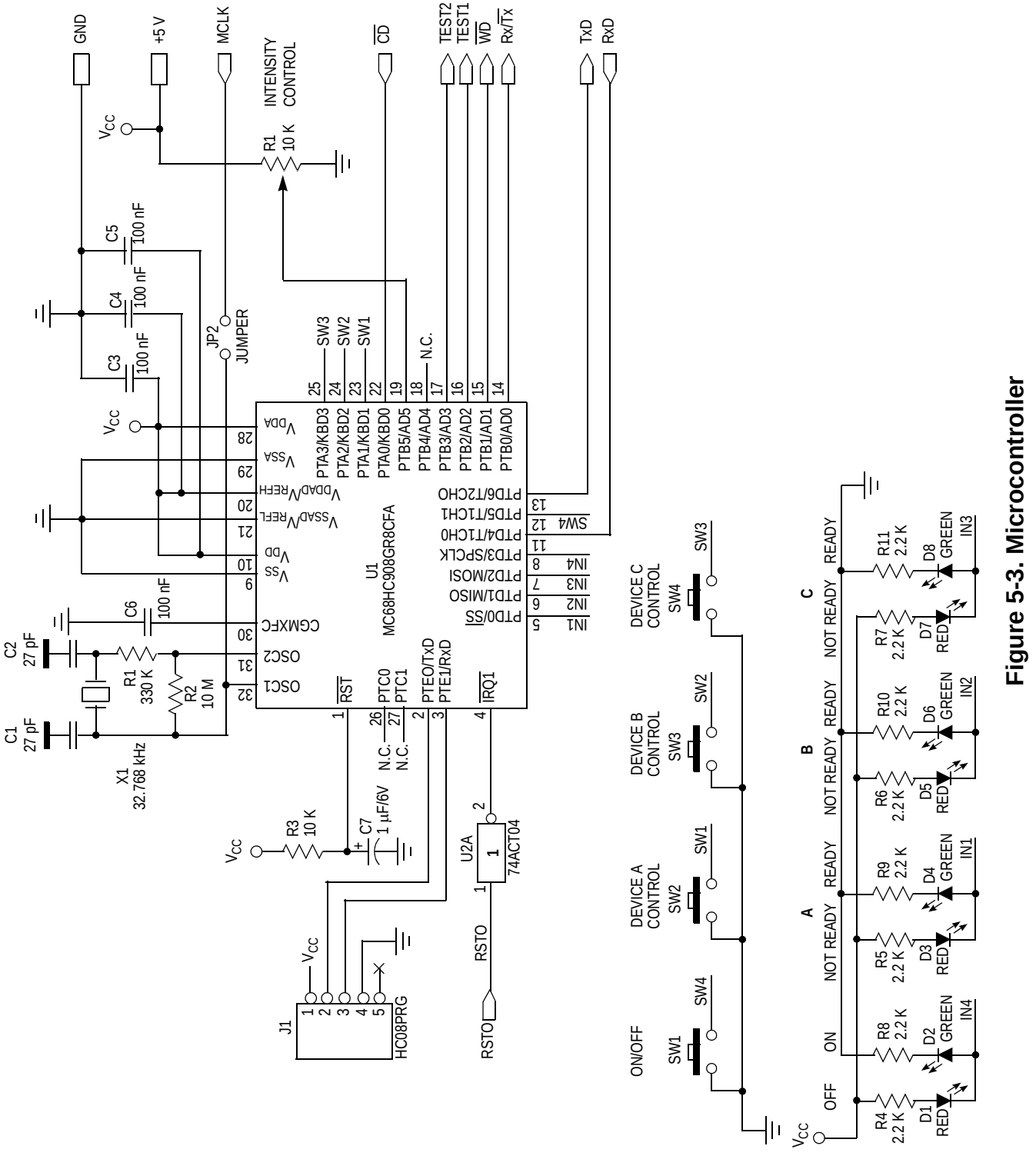


Figure 5-3. Microcontroller

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

The microcontroller's clock signal is provided from the FSK modem device with jumper JP2 closed. An option to use an alternative crystal can be implemented by populating components R1, R2, X1, C1, C2, and open jumper JP2.

The output TxD signal is provided by the timer 2 channel 0 pin, and the input RxD signal is received by the timer 1 channel 0 pin.

A reset (RSTO) from the FSK modem device is provided to the external interrupt request (IRQ1) which generates an interrupt service routine for the microcontroller.

The push buttons (SW1 to SW4) are used by the application for slave control. Pushing the button is an input event that results in the generation of a message from the master to the slave. The button function is done by software.

A variable potentiometer is connected to the analog convertor input channel 5. This is used to provide the dimming function for the slave lamps.

For the status optical signalling, eight LEDs (D1 to D8) are attached to port D. The RxD and TxD interface signals connected to the J1 connector are used for a reprogramming purpose.

5.7 Power Module

The application is supplied from the switch mode power AC-DC converter combined with linear voltage regulators. The power module provides 10-V and 5-V power supply. The power module schematic diagram can be seen in **Figure 5-4**.

The NCP1054 creates the heart of the power supply. This device is designed for direct operation from a rectified 240 Vac line source and requires minimal external components for a complete converter solution.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

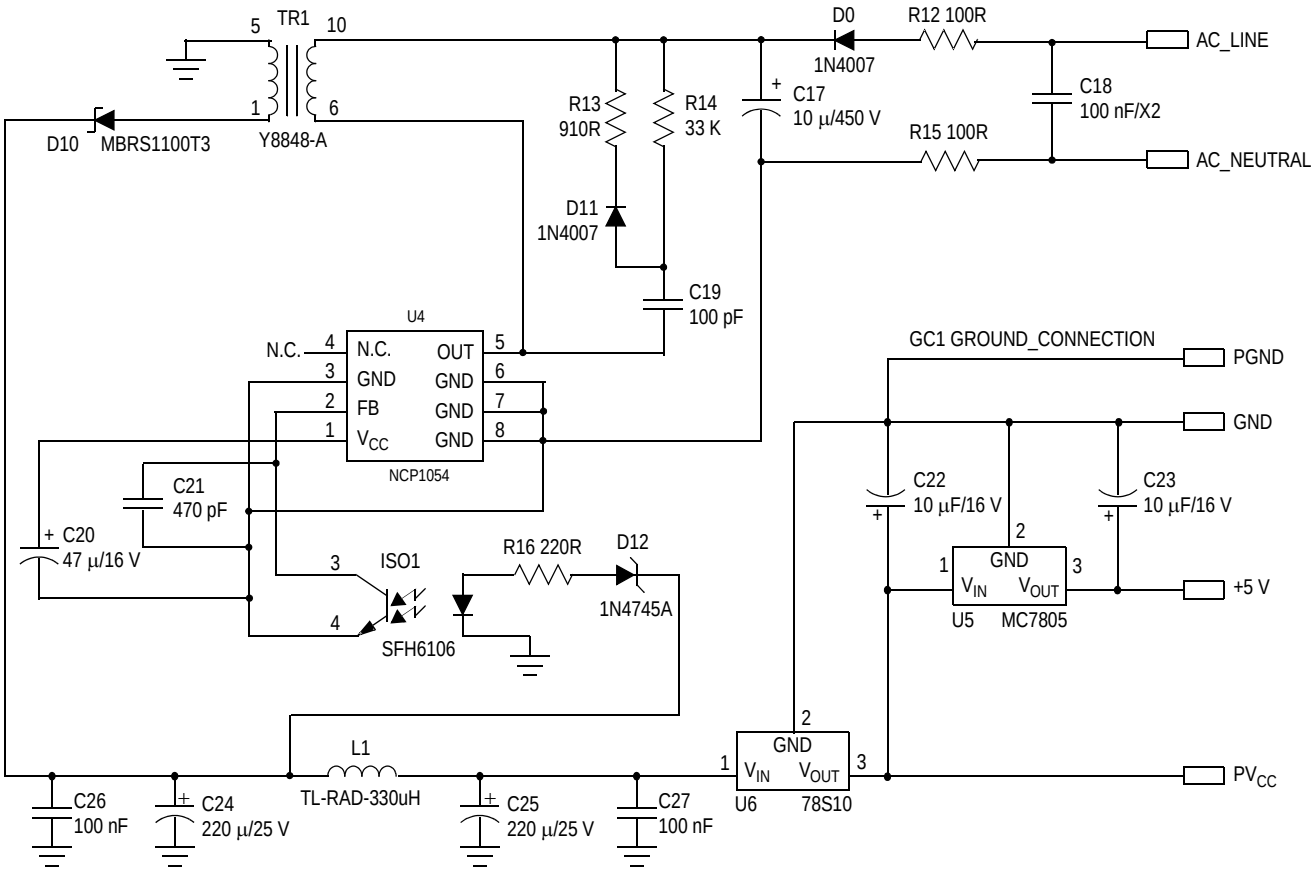


Figure 5-4. Power Module

The timing is controlled by a fixed–frequency, duty–cycle–controlled oscillator. It generates a square wave signal used to pulse width modulate the power switch circuit. The center oscillator frequency is internally programmed for 44-kHz, 100-kHz, or 136-kHz operation. To avoid the interference between the PLM carrier frequency and switch power frequency, the device programmed for 44kHz was selected for this application.

Components C18, R12, and R15 provide EMI filtering for the design. D1 along with C17 provide the AC to bulk DC rectification. The NCP1054 drives the primary side of the transformer.

The capacitor C19, resistors R13, R14, and D11 comprise a snubber to protect the IC from voltage transients greater than 700 volts and reduce radiated noise from the converter. Diode D10 along with C24, C26, L1,

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

C25, and C27 rectify and filter the transformer secondary voltage. The DC output voltage is set by the Zener diode D12. The opto-coupler ISO1 is driven by virtue of the difference in output voltage. The linear voltage regulator U6 provides 10-V output voltage, and voltage regulator U5 provides 5-V output voltage for the application power supply.

5.8 Slave Device Architecture

For this description, refer to **Figure 2-4. Slave Device Block Diagram**.

The electrical circuitry can be logically divided into following basic blocks:

- Power stage and coupling module
- FSK modem
- Microcontroller
- Triac
- Power module

5.8.1 Power Stage and Coupling

The basic power stage and coupling network can be seen in **Figure 5-5**. A non-isolated coupling network was designed for the slave node. Injecting a communication signal into a power mains circuit is accomplished by capacitively coupling a power stage amplifier output to the power mains.

The coupling capacitor C21 and the inductor L3 together act as a high-pass filter when receiving the communications signal. The high-pass filter attenuates the large AC mains signal (at either 50 Hz or 60 Hz), while passing the transceiver's communication signal. The value of the capacitor is chosen to be large enough so that its impedance at the communication frequencies is low, and its impedance at the mains power frequency (50 Hz or 60 Hz) is high. The value of the inductor is chosen to have a relatively high impedance at the transceiver's communication frequencies.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

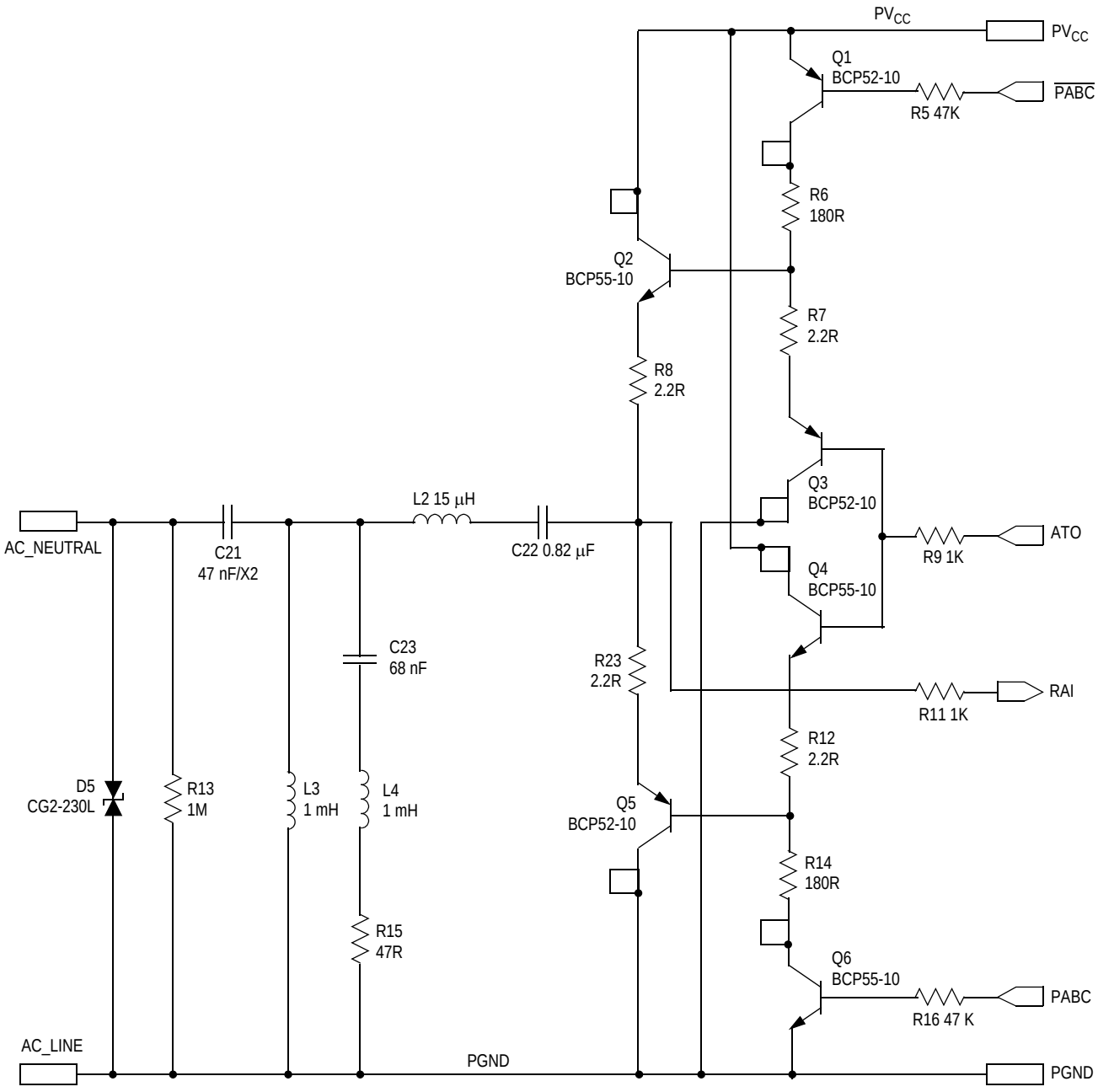


Figure 5-5. Slave Power Stage and Coupling

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Resistor R13 serves to discharge C21 when the device is disconnected from the power line. Varistor D5 provides protection against high-voltage transients on the power line.

The circuit consists of a serial network of inductor L4, capacitor C23, and resistor R15 acting as a notch filter. This circuit improves performance in environments where large (> 50 V) impulses may be present from a triac switching device.

The power stage amplifier is the same as for the Master node, more detailed information can be found in **5.3 Master Device Architecture**.

5.8.2 FSK Modem

The same half-duplex asynchronous FSK modem, ST7537HS1, used by the Master node is used for the Slave node. See **Figure 5-2**.

5.8.3 Microcontroller

Motorola's 8-bit MC68HC908GR8 (U2) microcontroller controls the slave PLM board. The schematic diagram can be seen in **Figure 5-6**.

The microcontroller's clock signal is provided from the FSK modem device when components R21, R22, X2, C25, and C26 are not populated on the board and jumper JP8 is closed.

The output TxD signal is provided by the timer 2 channel 0 pin and the input RxD signal is received by timer 1 channel 0 pin.

The external interrupt request A (IRQ1) input provides the RSTO signal service.

The gate of the triac is directly controlled by the microcontrollers pins PTC0 and PTC1. These pins are connected together and are powerful enough to cover the amount of current needed by the gate. The zero-crossing synchronization signal is connected to the PTA1 pin of the microcontroller.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

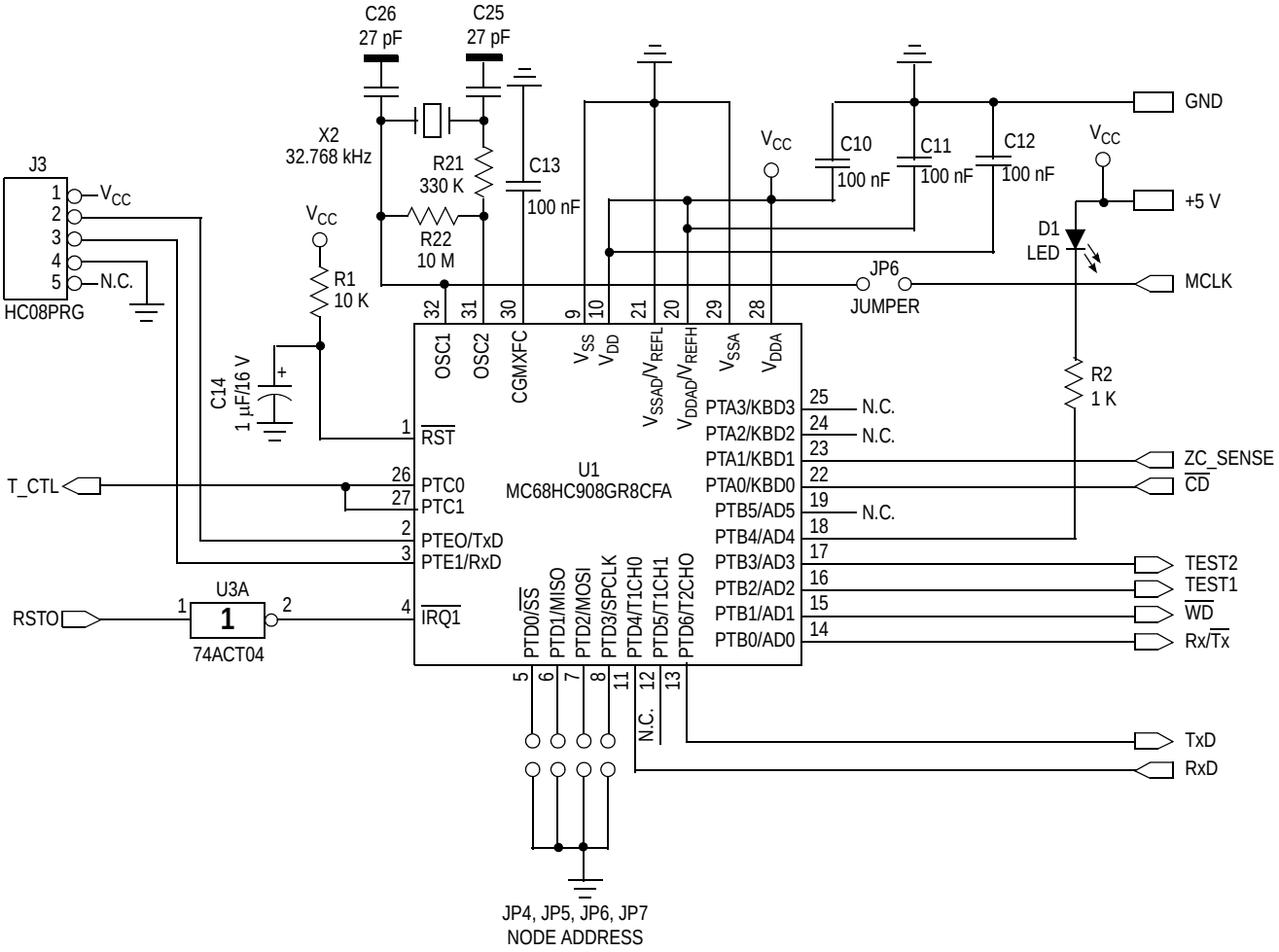


Figure 5-6. Microcontroller

The solder jumpers (JP4 to JP7) are used by the application for the slave node address set up. The address coding is done by software.

For the status optical signalling, LED D1 is attached to the port B4 pin.

The RxD and TxD interface signals connected to the JP3 connector are used for reprogramming purposes.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.8.4 Triac

The sensitive gate triggering triac type 2N6071B (Q7), compatible for direct coupling to a microcontroller, is used to control the load. The schematic diagram can be seen in **Figure 5-7**. The phase angle control technique adjusts the voltage applied to the load. A phase shift of the gate's pulses allows the effective voltage, seen by the load, to be varied.

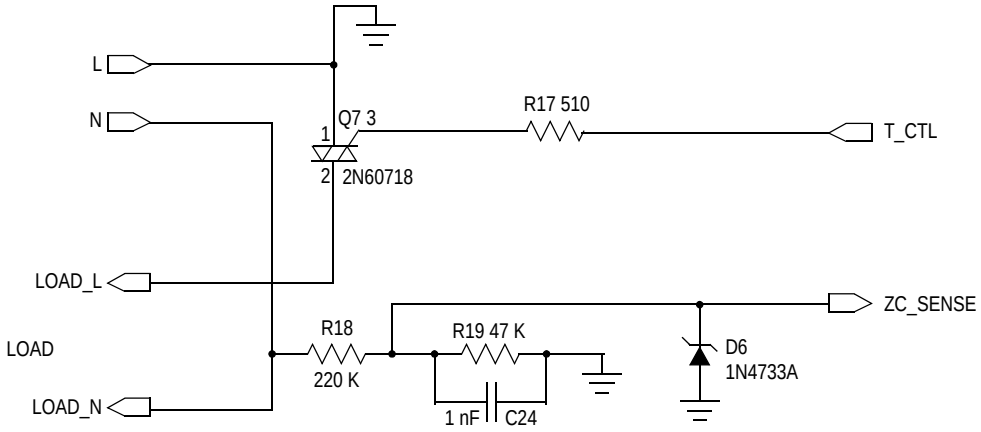


Figure 5-7. Triac Module

The resistors R18 and R19, capacitor C24, and Zener diode D6 create circuitry needed for the acquisition of a synchronization signal. This signal provides the most important information to the microcontroller, which is the zero crossing of the line voltage. The point of the zero crossing is fundamental for the calculation of any triac's action. All actions and the functionality concerning the triac are controlled by software.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

5.8.5 Power Module

The application is supplied from the switch mode power AC-DC converter combined with linear voltage regulators. The power module provides non-isolated 10-V and 5-V power supply for the application. The schematic diagram can be seen in **Figure 5-8**. The power module is based on the NCP1054 device programmed for 44 kHz, as in the power module for the master.

Components C19 and R3 provide EMI filtering for the design. D4 along with C16 provide the AC to bulk DC rectification. The NCP1054 drives the inductor L1. Diode D3 along with C18 rectify and filter the voltage.

The DC output voltage is set by the Zener diode D2. The opto coupler ISO1 is driven by virtue of the difference in output voltage. The linear voltage regulator U6 provides 10-V output voltage and voltage regulator U7 provides 5-V output voltage for the application power supply.

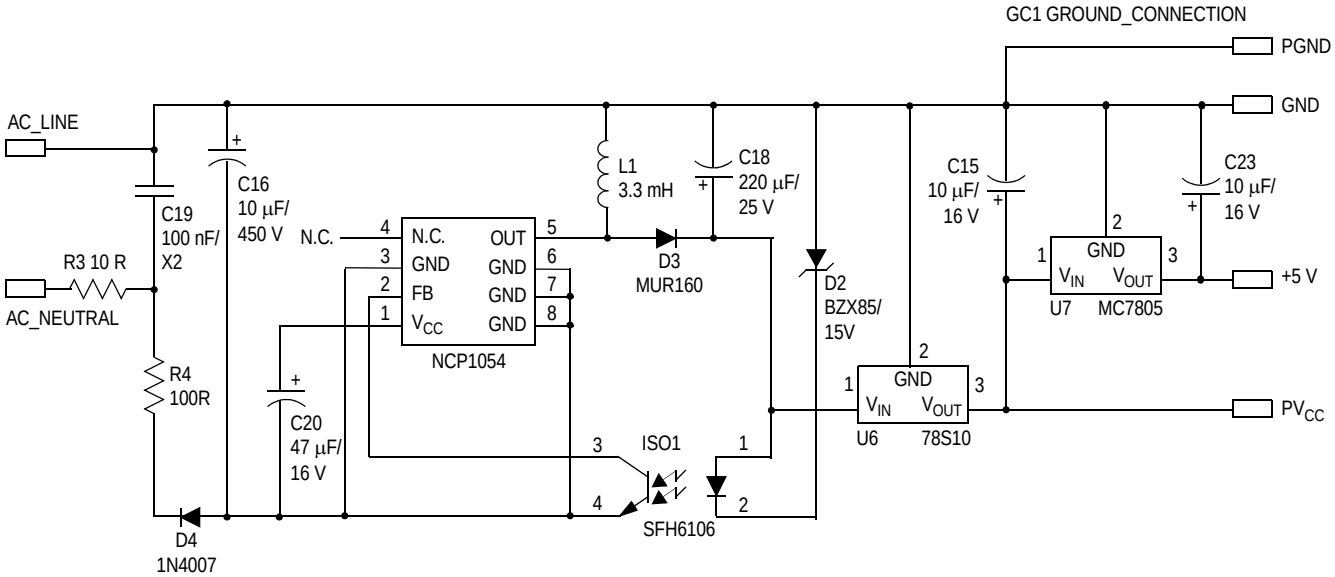


Figure 5-8. Power Module

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Section 6. Conclusions

The following tables provide reference design summary data.

- **Table 6-1** shows the MCU memory allocation of the Konnex PL132 physical and data link layers implementation and both versions of the demo project.
- **Table 6-2** displays master device values (physical and data link layers and master application).
- **Table 6-3** explains the memory consumption of the slave.

Table 6-1. MCU Memory Usage of PL132 Implementation

Type of Memory	MCU Total Size	Used Memory	Free Memory (%)
Data	384B	Roughly 120B	Roughly 69%
Program	7680B	Roughly 2500B	Roughly 67%

Table 6-2. MCU Memory Usage in Master Device

Type of Memory	MCU Total Size	Used Memory	Free Memory (%)
Data	384B	168B	Roughly 56%
Program	7680B	3002B	Roughly 61%

Table 6-3. MCU Memory Usage in Slave Device

Type of Memory	MCU Total Size	Used Memory	Free Memory (%)
Data	384B	144B	Roughly 62%
Program	7680B	2869B	Roughly 63%

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Conclusions

This reference design demonstrates a low-cost connectivity design that can be easily implemented with the following components:

- Microcontroller
- FSK modem device
- Power supply
- Transformer stage for isolated designs
- Minimum discrete devices

The reference design also shows that the Konnex power line PL132 standard can be implemented on any Motorola microcontroller that has at least five input/output pins for physical connections, approximately 2500 bytes of program memory, and 120 bytes of data memory (RAM).

An advantage of using Motorola's MC68HC908GR8 microcontroller in this Konnex reference design is that, by means of the embedded FLASH memory, the system can be re-configured very easily with in-circuit programming to perform additional functions.

Konnex power line communication is aimed at making consumer home applications more intelligent. With Motorola's many embedded FLASH microcontrollers, manufacturers can employ automatic configurability in their products.

"Motorola Making Products Smarter"

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Section 7. Source Code

7.1 Contents

7.2 Introduction

This subsection is comprised of the source code used by this design reference.

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.3 phs.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: phs.c
*
* Description: This file contains physical layer routines for transmission and
*       reception of data frames through powerline using ST7537 power line modem
*
* Modules Included:
*       phs_Init()
*       phs_Send()
*       phs_TxBitISR()
*       phs_CDdetectISR()
*       phs_RxEdgeISR()
*       phs_RxBitISR()
*       phs_IRQ_ISR()
*
* Written by Marek Stricek (R29303)
* Further development by Zdenek Kaspar (R55014)
*
* Revision history:
*       May-30-02- Initial coding
*       Oct-03-02- Coding finished
*
*****************************************************************************/
#include “hc08gp32.h”   /* HC08GP32 header file, suitable for HC08GR08 */
#include “phs.h”        /* KNX physical layer implementation */
#include “dll.h”        /* KNX data link layer implementation */
#include “app.h”        /* demo application layer implementation */

#include “board.h”      /* hardware dependant definitions for 00145_00 board
                           “KNX PL Master” (MCU based) and “KNX PL Slave” */


/*****************************************************************************/
/*              GLOBAL VARIABLES                                             */
/*****************************************************************************/
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
/* transmission / reception */
volatile phs_sFlags phs_Flags;      /* physical layer flags */
volatile phs_tPhase    trx_phase;   /* transmission / reception phase state */
unsigned char * pTxBuf;             /* pointer to Tx data buffer */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
volatile unsigned char phs_RxBufCount;  /* received bytes counter */
volatile unsigned char bitCounter;  /* bit counter */
volatile unsigned char byteCounter; /* byte counter */
volatile unsigned char tmpByte;     /* temp 1 byte long buffer for Rx/Tx */
volatile unsigned char tmpFec;      /* temp forward error correction variable */

#pragma DATA_SEG DEFAULT
unsigned char phs_RxBuf[KNX_BUF_LEN]; /* buffer used during frame reception */


/*****************************************************************************
* Module: void phs_Init()
*
* Description: In this routine the ST connection pins & physical link layer
*   initialization is done.
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
void phs_Init(void)
{
  phs_CD_ID(); /* disable CD interrupt => disable reception */

/* initialize all MCU pins connected to the ST PLM chip */

        /* initial values of output pins */
  PHS_RXTX=PLM_RECEP;   /* PLM will be in reception mode after start */
  PHS_WD=1;
  PHS_TEST1=1;          /* Rx/Tx mode controlled by Rx/Tx pin */
  PHS_TEST2=0;          /* WatchDog Timeout 1.5sec */
  PHS_TXD=1;

        /* direction of pins */
  DDR_PHS_RXTX=P_OUTPUT;
  DDR_PHS_WD=P_OUTPUT;
  DDR_PHS_TEST1=P_OUTPUT;
  DDR_PHS_TEST2=P_OUTPUT;
  DDR_PHS_TXD=P_OUTPUT;
  DDR_PHS_CD=P_INPUT;
  DDR_PHS_RXD=P_INPUT;

  phs_IRQ_CF(); /* clear IRQ flag */
  phs_WDServ();     /* service WD after going out of reset */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
  phs_CD_IE();      /* enable CD interrupt => enable reception */

        /* flags initialization */
  phs_FlgTxInProg = 0;  /* clear trasmission in progress flag */
  phs_FlgRxComp = 0;    /* clear reception completed flag */

  phs_StartRxEdgeTmr();  /* start Timer 1 */
    /* timer has to be switched ON!!! when edge detection (Input capture)
       on Tmr1Ch0 pin (used for Rx pin edge detection) is desired */
    /* Note that Timer 1 is also used for triac control in applic. layer */
}


/*************************************************************************
* Module: void phs_Send()
*
* Description: This is the physical link layer power line transmission routine.
*   It sends the data to the power line modem (ST chip) on the physical layer
*   level. It sends len bytes of data from pBuf buffer, MSB bits first.
*
* Returns: None
*
* Global Data:
*       bitCounter
*       byteCounter
*       trx_phase
*       pTxBuf
*       phs_FlgTxInProg
*
* Arguments: * pBuf - pointer to buffer to send
*            len - number of bytes to send
*
* Range Issues: Note that the implementation supports only short frames format
*   of the Konnex PL132 specification
*
* Special Issues: Note that 1st byte defined in pBuf is transmitted as a 2nd
*    part of header.
*
*************************************************************************/
void phs_Send(unsigned char * pBuf, unsigned char len)
{
  PHS_RXTX=PLM_TRANS;   /* enable transmission on ST chip */
  phs_FlgTxInProg=1;    /* set transmission in progress flag */

  bitCounter=16;        /* set counters */
  byteCounter=len;
  trx_phase=W_PREAMBLE; /* set phase to 16bit preamble */
  pTxBuf=pBuf;          /* set data pointer */
  phs_TmrForTx();       /* set PLM associated timers for transmission */
  phs_StartTxTmr();     /* start PL transmission (Tmr 2) */
  while (phs_FlgTxInProg)   /* wait for finish */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
  {                     /* (we will not service WD during that time) */

  /* ***************************/
  /* START OF APPLICATION PART */

    #ifdef SLAVE
         app_ZcDetect();    /* Zero crossing detection routine */
    #endif

  /* END OF APPLICATION PART   */
  /* ***************************/
  }
  PHS_RXTX=PLM_RECEP;   /* disable transmission (thus enable reception) */
}


/*****************************************************************************
* Module: void phs_TxBitISR()
*
* Description: This is the Timer 2 Overflow ISR. It is used for data fetching
*   to the PL transmission routine on the level of physical link layer.
*
* Returns: None
*
* Global Data:
*       bitCounter
*       byteCounter
*       trx_phase
*       tmpByte
*       pTxBuf
*       tmpFec
*       phs_FlgTxInProg
*
* Arguments: None
*
* Range Issues: Note that the implementation supports only short frames format
*   of the Konnex PL132 specification
*
* Special Issues: Note that 1st byte defined in pBuf is transmitted as a 2nd
*    part of header.
*
*****************************************************************************/
#pragma TRAP_PROC
void phs_TxBitISR(void) /* will be invoked by T2 overflow interrupt */
{
  phs_TxOvrfl_CF();     /* clear interrupt flag */

  switch (trx_phase)    /* possible Tx phases */
  {
    case W_PREAMBLE:
        PHS_TXBIT= bitCounter & 0x01;  /* generate 0xAAAA preamble */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
    bitCounter--;
    if (bitCounter == 0)    /* if finished */
    {
      bitCounter=8;
      trx_phase++;
      tmpByte=KNX_HDR_HIGH; /* set next byte to be send */
    }
  break;

case W_HEADER1:
    PHS_TXBIT=(tmpByte & 0x80) ? 1:0; /* prepare bit for next output compare
                                         (OC) event */
    tmpByte=tmpByte << 1;   /* shift current byte 1 bit left */
    bitCounter--;
    if (bitCounter == 0)    /* if finished */
    {
      bitCounter=8;
      trx_phase++;
      tmpByte=*pTxBuf;  /* 1st byte in data block is 2nd part of header */
      pTxBuf++;
      byteCounter--;
    }
    break;

case W_HEADER2:
    PHS_TXBIT=(tmpByte & 0x80) ? 1:0; /* prepare bit for next OC event */
    tmpByte=tmpByte << 1;   /* shift current byte 1 bit left */
    bitCounter--;
    if (bitCounter == 0)    /* if finished */
    {
      bitCounter=8;
      trx_phase++;
      tmpByte=*pTxBuf;      /* now fetch the first regular data byte */
      pTxBuf++;             /* (second in data buffer) */
      tmpFec=0;             /* prepare FEC variable for calculation */
      byteCounter--;
    }
    break;

case W_DATA:
    PHS_TXBIT=(tmpByte & 0x80) ? 1:0; /* prepare bit for next OC event */
    tmpFec=tmpFec << 1;     /* FEC calculations */
    tmpFec|=PHS_TXBIT;
    if (tmpFec & 0x40)
    {
       tmpFec^=0x39;
    }
    tmpByte=tmpByte << 1;   /* shift current byte 1 bit left */
    bitCounter--;
    if (bitCounter == 0)    /* if finished */
    {
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
          bitCounter=6;
          while (bitCounter!=0) /* FEC calculations */
          {
            tmpFec=tmpFec << 1;
            if (tmpFec & 0x40)
            {
              tmpFec^=0x39;
            }
            bitCounter--;
          }
          bitCounter=6;         /* prepare for sending calculated 6 FEC bits */
          trx_phase++;
        }
        break;

    case W_FEC:
        tmpFec=tmpFec << 1;
        PHS_TXBIT=(tmpFec & 0x40) ? 0 : 1; /* send complemented!! bits of FEC */
        bitCounter--;
        if (bitCounter == 0)        /* if finished */
        {
          if (byteCounter == 0)     /* if whole buffer transmitted */
          {
            trx_phase=W_POSTAMBLE;
            bitCounter=2;           /* postamble consists of 2 bits */
            tmpFec=PHS_TXBIT ^ 0x01;/* which are complement of last FEC bit */
          }
          else
          {
            bitCounter=8;           /* whole buffer is not transmitted yet */
            trx_phase=W_DATA;
            tmpByte=*pTxBuf;        /* fetch next regular data byte */
            pTxBuf++;
            tmpFec=0;           /* prepare FEC variable for next data byte */
            byteCounter--;
          }
        }
        break;

    case W_POSTAMBLE:
        PHS_TXBIT=tmpFec;   /* postamble value is a complement of last FEC bit */
        bitCounter--;
        if (bitCounter == 0)    /* if finished */
        {
          PHS_TXBIT=1;      /* assure that 1 will be on TX pin */
          phs_TxOvrfl_ID;   /* disable further overflow interrupts on Tmr 2 */
          phs_FlgTxInProg=0;/* transmission finished */
        }
        break;
  }
}
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
/*****************************************************************************
* Module: void phs_CDdetectISR()
*
* Description: This is the keyboard interrupt (KBI) ISR. It is used for the
*   falling edge detection of the carrier detection (CD) signal of the ST chip.
*   CD is active in low, only when CD signal is active, the PL reception may
*   be started.
*
* Returns: None
*
* Global Data:
*       bitCounter
*       trx_phase
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
#pragma TRAP_PROC
void phs_CDdetectISR(void) /* CD from powerline occured */
{
  phs_CD_CF();      /* clear flag */

  if (!PHS_CD)      /* only if there is CD signal present start PL reception */
  {
    phs_TmrForRx();       /* configure both Timers for reception */
    trx_phase=W_PREAMBLE;
    bitCounter=0;
  }
}


/*****************************************************************************
* Module: void phs_RxEdgeISR()
*
* Description: This is the Timer 1 Channel 0 input capture (IC) ISR. It is
*   called in order to synchronize the receiver timer with the incoming data bit
*   stream on Rx pin of the ST chip.
*   Each falling edge synchronizes (reset and restart again) the reception timer
*   Tmr 2.
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
#pragma TRAP_PROC
void phs_RxEdgeISR(void)/* falling edge detected on the RX pin of the ST chip */
{
  phs_RxEdge_CF();      /* clear interrupt flag */

  phs_RestartRxTmr();   /* reset the Tmr 2 counter and restart it again in order
                           to synchronize with the incoming bit stream */
}


/*******************************************************************************
* Module: void phs_RxBitISR()
*
* Description: This is the Timer 2 output compare ISR. It is used for the data
*   reception from the PLM on the level of physical link layer.
*   In this interrupt the actual bit value is read using the output compare
*   event in 1/2 of the bit period. Byte value is then checked using 6bit long
*   FEC information and if OK, received byte is stored into phs_RxBuf buffer.
*
* Returns: None
*
* Global Data:
*       phs_FlgRxComp
*       trx_phase
*       bitCounter
*       tmpByte
*       tmpFec
*       phs_RxBufCount
*       phs_RxBuf[]
*
* Arguments: None
*
* Range Issues: Note that the implementation supports only short frames format
*   of the Konnex PL132 specification
*
* Special Issues: None
*
*******************************************************************************/
#pragma TRAP_PROC
void phs_RxBitISR(void)
{
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
  unsigned char tmpInBit;   /* temp variable for current incoming bit */
  static unsigned char phs_Header; /* header information */
#pragma DATA_SEG DEFAULT
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
  phs_RxOC_CF();    /* clear OC interrupt flag */
  tmpInBit=PHS_RXD; /* catch actual incoming bit value */

#ifdef SLAVE
  if (tmpInBit == 0) ledIndOff();   /* reception stream is visible on LED */
  else ledIndOn();
#endif

  if (PHS_CD)       /* carrier signal no longer present -> drop reception */
  {
    phs_RxEdge_ID();    /* disable interrupt from RX pin edge detection event */
    phs_StopRxTmr();    /* stop the Tmr 2 counter => stop PL reception */
    phs_FlgRxComp=1;    /* reception on the phs layer finished - data available
                           in phs_RxBuf[] for dll reception routines */

    if (phs_Header == KNX_HDR_DATA)
    {
        dll_recKNXData();   /* call dll Data reception routine */
    }
    if (phs_Header == KNX_HDR_ACK)
    {
        dll_recKNXACK();   /* call dll ACK reception routine */
    }
  }

  switch (trx_phase)
  {
    case W_PREAMBLE:
        trx_phase++;    /* jump right to the next phase */
        break;

    case W_HEADER1:
        tmpByte= tmpByte << 1;
        tmpByte|=tmpInBit;

        if (tmpByte == KNX_HDR_HIGH)
        {
            phs_Header = 0; /* clear store header information */
            bitCounter=8;
            trx_phase++;    /* first byte of header OK go to next one */
        }
        break;

    case W_HEADER2:
        tmpByte= tmpByte << 1;
        tmpByte|= tmpInBit;
        bitCounter--;
        if (bitCounter == 0)
        {
            if ((tmpByte == KNX_HDR_DATA) || (tmpByte == KNX_HDR_ACK))
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
            {
                phs_Header = tmpByte;   /* store header information */
                bitCounter=8;
                trx_phase++;    /* 2nd byte of header OK go to data part */
                tmpFec=0;       /* reset tmpFec */
                phs_RxBufCount=0;   /* reset reception data counter */
            }
            else
            {
                trx_phase=W_PREAMBLE;   /* header not recognized */
            }                           /* go back to preamble reception */
        }
        break;

    case W_DATA:
        tmpFec=tmpFec << 1;
        tmpFec|=tmpInBit;
        tmpByte= tmpByte << 1;
        tmpByte|=tmpInBit;
        if (tmpFec & 0x40)
        {
            tmpFec^=0x39;
        }
        bitCounter--;
        if (bitCounter == 0)
        {
            bitCounter=6;
            if (phs_RxBufCount < KNX_BUF_LEN)
            {
                phs_RxBuf[phs_RxBufCount]=tmpByte;
                            /* store received byte into the reception buffer */
            }
            trx_phase++;    /* now it is time to check the FEC */
        }
        break;

    case W_FEC:
        tmpFec=tmpFec << 1;
        tmpFec|=tmpInBit ^0x01;  /* FEC bits are received as complements */
        if (tmpFec & 0x40)
        {
            tmpFec^=0x39;
        }
        bitCounter--;
        if (bitCounter == 0)
        {
            bitCounter=8;
            if (tmpFec & 0x3f)
            {
                  /* if we get here -> error during trasmission/reception */
                  /* !!!! do correction (not implemented yet) */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
            }
            tmpFec=0;
            if (phs_RxBufCount < KNX_BUF_LEN)
            {
                phs_RxBufCount++;
                    /* increment counter only if enough space available */
            }
            trx_phase=W_DATA;
        }
        break;

    case W_POSTAMBLE:
        break;
  }
}


/*****************************************************************************
* Module: void phs_IRQ_ISR()
*
* Description: This is the IRQ ISR. It has to be served when WatchDog timeout
*   occured on the ST chip.
*
* Returns: None
*
* Global Data:
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
#pragma TRAP_PROC
void phs_IRQ_ISR(void)  /* PLM WatchDog timeout or Low Voltage interrupt */
{
  phs_IRQ_CF(); /* clear IRQ flag */
  phs_WDServ();     /* service WD */

  illegalOperation(); /* reset the MCU */
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.4 phs.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: phs.h
*
* Description: This is header file for ‘phs.c’
*
* Modules Included: None
*
* Written by Marek Stricek (R29303)
* Further development by Zdenek Kaspar (R55014)
*
* Revision history:
*       May-30-02- Initial coding
*       Oct-03-02- Coding finished
*
*****************************************************************************/
#ifndef _PHS_H
#define _PHS_H


/*****************************************************************************/
/*                P R O T O T Y P E S                                        */
/*****************************************************************************/
void phs_Init (void);
void phs_Send (unsigned char * pBuf, unsigned char len);


/*****************************************************************************/
/*                GLOBAL VARIABLES & DEFINES                                 */
/*****************************************************************************/
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
extern volatile unsigned char phs_RxBufCount;   /* received bytes counter */
#pragma DATA_SEG DEFAULT

#define KNX_BUF_LEN  25        /* maximum length of KNX reception buffer */
extern unsigned char phs_RxBuf[KNX_BUF_LEN];     /* received data buffer */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
/*****************************************************************************/
/*                TYPE DEFINITIONS                                           */
/*****************************************************************************/
/* type definition for both transmission (Tx) / reception (Rx) phases */
typedef enum {
    W_PREAMBLE,  /* waiting for preamble */
    W_HEADER1,   /* waiting for header */
    W_HEADER2,
    W_DATA,      /* waiting for data byte */
    W_FEC,       /* and for associated FEC */
    W_POSTAMBLE  /* wait for end */
    } phs_tPhase;


/*****************************************************************************/
/*                S T R U C T U R E S                                        */
/*****************************************************************************/
/* Physical link layer flags */
typedef struct {
    unsigned char txInProgress : 1;  /* PL transmission in progress flag */
    unsigned char rxCompleted  : 1;  /* PL reception completed flag */
} phs_sFlags;


/*****************************************************************************/
/*                SHORT-CUT DEFINES                                          */
/*****************************************************************************/
#define phs_FlgTxInProg       phs_Flags.txInProgress
#define phs_FlgRxComp         phs_Flags.rxCompleted


/*****************************************************************************/
/*                KONNEX specific constants                                  */
/*****************************************************************************/
#define KNX_BAUD_RATE 2400L  /* comm. speed over powerline according to PL132 */

/* header values */
#define KNX_HDR_HIGH 0x1C
#define KNX_HDR_DATA 0x53
#define KNX_HDR_ACK  0xA1


/*****************************************************************************/
/*                HARDWARE DEPENDENT PIN CONNECTIONS WITH ST chip             */
/*****************************************************************************/
#define PLM_TRANS     0     /* transmission */
#define PLM_RECEP     1     /* reception */

#define P_OUTPUT 1          /* pin as output */
#define P_INPUT  0          /* pin as input */
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
#define PHS_RXTX      PTB0  /* Rx / Tx pin */
#define DDR_PHS_RXTX  DDRB_BIT0
#define PHS_WD        PTB1  /* Watch Dog pin */
#define DDR_PHS_WD    DDRB_BIT1
#define PHS_CD        PTA0  /* Carrier Detection pin */
#define DDR_PHS_CD    DDRA_BIT0
#define PHS_TXD       PTD6  /* TxD pin */
#define DDR_PHS_TXD   DDRD_BIT6
#define PHS_RXD       PTD4  /* RxD pin */
#define DDR_PHS_RXD   DDRD_BIT4
#define PHS_TEST1     PTB2  /* Test 1 pin */
#define DDR_PHS_TEST1 DDRB_BIT2
#define PHS_TEST2     PTB3  /* Test 2 pin */
#define DDR_PHS_TEST2 DDRB_BIT3

/* hardware dependant definitions */
#define TIM_PERIOD   BUS_CLK/KNX_BAUD_RATE
#define TIM_P_HALF   TIM_PERIOD/2


/*****************************************************************************/
/*              M A C R O S                                                  */
/*****************************************************************************/
/* service PLM watchdog */
#define phs_WDServ()  PHS_WD=0; asm nop; asm nop; asm nop; PHS_WD=1

/* IRQ related function style macros for WatchDog ST's output pin */
#define phs_IRQ_CF()  ISCR_ACK1=1
#define phs_IRQ_IE()  ISCR_IMASK1=0
#define phs_IRQ_ID()  ISCR_IMASK1=1


/*****************************************************************************/
/* reception and Rx timers related macros & function style macros            */
/*****************************************************************************/
/* Resources used for the PL reception:
   - Keyboard interrupt on pin PTA0 (ISR is called phs_CDdetectISR) is used for
     falling edge detection of the Carrier Detection (CD) signal.
   - Tmr1Ch0 in the input compare mode with ISR called phs_RxEdgeISR is used for
     the incoming bit stream synchronization
   - Tmr2Ch0 in output compare mode with ISR called phs_RxBitISR is used for
     the Rx data sampling (in 1/2 of the bit period)
   - Tmr2 overflow (with no ISR) keeps the bit period synchronization during
     the reception */

/* CD related function style macros connected with the port A (keyboard intr) */
#define phs_CD_ID() INTKBIER_KBIE0=0    /* CD interrupt disable */
#define phs_CD_IE() INTKBIER_KBIE0=1    /* CD interrupt enable */
#define phs_CD_CF() INTKBSCR_ACKK=1     /* clear CD interrupt flag */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
/* Timer related */
#define phs_RxEdge_CF()   T1SC0_CH0F=0  /* clear interrupt flag of Rx edge
                                           detection on Tmr1Ch0 */
#define phs_RxEdge_IE()   T1SC0_CH0IE=1 /* enable interrupt of Rx edge
                                           detection on Tmr1Ch0 */
#define phs_RxEdge_ID()   T1SC0_CH0IE=0 /* disable interrupt of Rx edge
                                           detection on Tmr1Ch0 */

#define phs_StartRxEdgeTmr()    T1SC=0  /* start Timer 1 */
    /* timer has to be switched ON!!! when edge detection (Input capture)
       on Tmr1Ch0 pin (used for Rx pin edge detection) is desired */
    /* Note that Timer 1 is also used for triac control in applic. layer */

/* phs_TmrForTx() routine prepares both Timers for PL Rx mode:
   - clear possible overflow interrupt flag of Tmr2
   - stop & reset Tmr2 counter
   - enable input capture interrupt from RX pin (T1CH0) on falling edge
   - set period of Timer 2 modulo counter for overflow event to 1 bit length
   - set period of Timer 2 Channel 0 counter to 1/2 bit length
   - finally set the mode T2CH0 to output compare (with interrupt enabled) */
#define phs_TmrForRx() phs_TxOvrfl_CF(); T2SC=0x30; \
                       T1SC0=0x08; phs_RxEdge_IE(); \
                       T2MODH=TIM_PERIOD/256; T2MODL=TIM_PERIOD%256; \
                       T2CH0H=TIM_P_HALF/256; T2CH0L=TIM_P_HALF%256; \
                       T2SC0=0x5C

#define phs_RestartRxTmr()  T2SC|=0x30; T2SC_TSTOP=0
        /* reset the Tmr 2 counter and restart it again in order to synchronize
           with the incoming bit stream during the reception */
#define phs_RxOC_CF()       T2SC0_CH0F=0
        /* clear flag after output compare ISR event during the PL reception */
#define phs_StopRxTmr()     T2SC=0x30
        /* stop the Tmr 2 counter => stop PL reception */


/*****************************************************************************/
/* transmission & Tx timers related macros & function style macros           */
/*****************************************************************************/
/* Resources used for the PL transmission:
   - Tmr2 overflow ISR called phs_TxBitISR loads the new bit value for PL
     transmission
   - Tmr2Ch0 is in output compare mode used for transmission itself */
/* Note that there is no need for Tmr2Ch0 output compare interrupt!!! */

#define PHS_TXBIT   T2SC0_ELS0A      /* this technique enables to set / clear
                                        output pin of timer on compare event */

#define phs_TxOvrfl_IE     T2SC_TOIE = 1    /* enable overfow intr on Tmr 2 */
#define phs_TxOvrfl_ID     T2SC_TOIE = 0    /* disable overfow intr on Tmr 2 */
#define phs_TxOvrfl_CF()   T2SC_TOF = 0     /* clear interrupt flag from
                                               overflow on Tmr 2 counter */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
/* phs_TmrForTx() routine prepares both Timers for PL Tx mode:
   - clear possible overflow interrupt flag of Tmr2
   - stop & reset Tmr 2 counter
   - enable overflow interrupt on Tmr2
   - disable interrupt from RX pin (on T1CH0)
   - set period of Timer 2 modulo counter for overflow event to 1 bit length
   - set period of Tmr 2 Channel 0 counter to 1/2 bit length
   - finally set the mode of T2CH0 to output compare (with intr disabled!) */
#define phs_TmrForTx()  phs_TxOvrfl_CF(); T2SC=0x30; phs_TxOvrfl_IE; \
                        phs_RxEdge_ID(); \
                        T2MODH=TIM_PERIOD/256; T2MODL=TIM_PERIOD%256; \
                        T2CH0H=TIM_P_HALF/256; T2CH0L=TIM_P_HALF%256; \
                        T2SC0=0x1C

#define phs_StartTxTmr() T2SC_TSTOP=0     /* Tmr 2 starts the PL transmission */

#endif
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.5 dll.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: dll.c
*
* Description: This file contains data link layer routines for transmission and
*       reception of data frames through powerline using ST7537 power line modem
*
* Modules Included:
*       dll_Init()
*       dll_TBModuleISR()
*       dll_CalcCRC()
*       dll_sendKNXData()
*       dll_sendKNXACK()
*       dll_recKNXData()
*       dll_recKNXACK()
*
* Written by Zdenek Kaspar (R55014)
*
* Revision history:
*       Aug-22-02- Initial coding
*       Oct-03-02- Coding finished
*
*****************************************************************************/
#include “hc08gp32.h”   /* HC08GP32 header file, suitable for HC08GR08 */
#include “phs.h”        /* KNX physical layer implementation */
#include “dll.h”        /* KNX data link layer implementation */
#include “app.h”        /* demo application layer implementation */

#include “board.h”      /* hardware dependant definitions for 00145_00 board
                           “KNX PL Master” (MCU based) and “KNX PL Slave” */


/*****************************************************************************/
/*              GLOBAL VARIABLES                                             */
/*****************************************************************************/
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
unsigned char i;                        /* index */
extern volatile phs_sFlags phs_Flags;   /* physical layer flags */
volatile dll_sFlags dll_Flags;          /* data link layer flags */
volatile unsigned int dll_RecFCS;       /* received FCS by dll_recKNXACK */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
volatile unsigned int dll_DesiredDelay; /* desired delay value */
volatile unsigned int dll_OneSecCount;  /* 1.1s delay counter, when is this time
                    period over, transmission can start even if line occupied */
volatile unsigned char dll_nevEndCount; /* neverending counter dedicated for
                                           random generation */
#ifdef MASTER
    unsigned int dll_masterAddr;    /* address of the master node */
    unsigned int dll_nodeAddrInM[3];/* addresses of three slave nodes connected
                                       to master */
    extern volatile unsigned char app_AliveCountInM[3]; /* counter dedicated for
            application, used for “are you alive?” message timing in master */
    extern unsigned char app_indexOfDev;    /* index for State of devices */
                                            /* index = 0, 1 or 2 */
#endif
#ifdef SLAVE
    unsigned int dll_nodeAddr;      /* address of the slave node */
    unsigned int dll_masterAddrInS; /* address of the connected master node */

    extern volatile unsigned char app_AliveCount; /* counter dedicated for
            application, used for “are you alive?” message timing in slave */

    extern unsigned char app_desAnalogVal;
                    /* desired analog value received in msg */
    extern unsigned char app_actAnalogVal;
                    /* actual potenciometer analog value */
#endif

#pragma DATA_SEG DEFAULT
unsigned char dll_TxBuf[DLL_TXBUF_LEN];  /* Tx data buffer on data link layer
                                            level */
extern volatile unsigned char app_TxBuf[APP_BUF_LEN];
                                /* appl. buffer used during frame Tx */
extern volatile unsigned char app_RxBuf[APP_BUF_LEN];
                                /* appl. buffer used during frame Rx */


/*****************************************************************************/
/*              Look-up table for CRC calculation                            */
/*****************************************************************************/
/* For information: As a CRC generator polynomial following one is defined in
   KNX PL132:
                 16    15   2
                X   + X  + X  + 1
NOTE: Following is not specified in KNX PL132 specification, but necessary to
    define for precise CRC definition!!!
    - CRC computation algorithm starts with zero
    - it treats the data msb first
    - CRC result is not complemented
Example: the sequence 01 02 03 04 05 06 07 08 has CRC value TBD. */
/*****************************************************************************/
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
#pragma CONST_SEG ROM_CONST
  static const int  tableCRC[] = {
 0x0000, 0x8005, 0x800F, 0x000A, 0x801B, 0x001E, 0x0014, 0x8011,
 0x8033, 0x0036, 0x003C, 0x8039, 0x0028, 0x802D, 0x8027, 0x0022,
 0x8063, 0x0066, 0x006C, 0x8069, 0x0078, 0x807D, 0x8077, 0x0072,
 0x0050, 0x8055, 0x805F, 0x005A, 0x804B, 0x004E, 0x0044, 0x8041,
 0x80C3, 0x00C6, 0x00CC, 0x80C9, 0x00D8, 0x80DD, 0x80D7, 0x00D2,
 0x00F0, 0x80F5, 0x80FF, 0x00FA, 0x80EB, 0x00EE, 0x00E4, 0x80E1,
 0x00A0, 0x80A5, 0x80AF, 0x00AA, 0x80BB, 0x00BE, 0x00B4, 0x80B1,
 0x8093, 0x0096, 0x009C, 0x8099, 0x0088, 0x808D, 0x8087, 0x0082,
 0x8183, 0x0186, 0x018C, 0x8189, 0x0198, 0x819D, 0x8197, 0x0192,
 0x01B0, 0x81B5, 0x81BF, 0x01BA, 0x81AB, 0x01AE, 0x01A4, 0x81A1,
 0x01E0, 0x81E5, 0x81EF, 0x01EA, 0x81FB, 0x01FE, 0x01F4, 0x81F1,
 0x81D3, 0x01D6, 0x01DC, 0x81D9, 0x01C8, 0x81CD, 0x81C7, 0x01C2,
 0x0140, 0x8145, 0x814F, 0x014A, 0x815B, 0x015E, 0x0154, 0x8151,
 0x8173, 0x0176, 0x017C, 0x8179, 0x0168, 0x816D, 0x8167, 0x0162,
 0x8123, 0x0126, 0x012C, 0x8129, 0x0138, 0x813D, 0x8137, 0x0132,
 0x0110, 0x8115, 0x811F, 0x011A, 0x810B, 0x010E, 0x0104, 0x8101,
 0x8303, 0x0306, 0x030C, 0x8309, 0x0318, 0x831D, 0x8317, 0x0312,
 0x0330, 0x8335, 0x833F, 0x033A, 0x832B, 0x032E, 0x0324, 0x8321,
 0x0360, 0x8365, 0x836F, 0x036A, 0x837B, 0x037E, 0x0374, 0x8371,
 0x8353, 0x0356, 0x035C, 0x8359, 0x0348, 0x834D, 0x8347, 0x0342,
 0x03C0, 0x83C5, 0x83CF, 0x03CA, 0x83DB, 0x03DE, 0x03D4, 0x83D1,
 0x83F3, 0x03F6, 0x03FC, 0x83F9, 0x03E8, 0x83ED, 0x83E7, 0x03E2,
 0x83A3, 0x03A6, 0x03AC, 0x83A9, 0x03B8, 0x83BD, 0x83B7, 0x03B2,
 0x0390, 0x8395, 0x839F, 0x039A, 0x838B, 0x038E, 0x0384, 0x8381,
 0x0280, 0x8285, 0x828F, 0x028A, 0x829B, 0x029E, 0x0294, 0x8291,
 0x82B3, 0x02B6, 0x02BC, 0x82B9, 0x02A8, 0x82AD, 0x82A7, 0x02A2,
 0x82E3, 0x02E6, 0x02EC, 0x82E9, 0x02F8, 0x82FD, 0x82F7, 0x02F2,
 0x02D0, 0x82D5, 0x82DF, 0x02DA, 0x82CB, 0x02CE, 0x02C4, 0x82C1,
 0x8243, 0x0246, 0x024C, 0x8249, 0x0258, 0x825D, 0x8257, 0x0252,
 0x0270, 0x8275, 0x827F, 0x027A, 0x826B, 0x026E, 0x0264, 0x8261,
 0x0220, 0x8225, 0x822F, 0x022A, 0x823B, 0x023E, 0x0234, 0x8231,
 0x8213, 0x0216, 0x021C, 0x8219, 0x0208, 0x820D, 0x8207, 0x0202 };
#pragma CONST_SEG DEFAULT


/*****************************************************************************
* Module: void dll_Init()
*
* Description: In this routine the data link layer initialization is done.
*
* Returns: None
*
* Global Data:
*       dll_nodeAddr
*       dll_masterAddrInS
*       dll_masterAddr
*       dll_nodeAddrInM[]
*
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*********************************************************************************/
#pragma INLINE
void dll_Init()     /* data link layer init */
{
#ifdef SLAVE
    dll_nodeAddr = NODE_ADDR;   /* set address of slave node */
        /* address is read from the address configuration on 4 pins of port D */
    dll_masterAddrInS = 0xFF;   /* set address of the master node */
#endif
#ifdef MASTER
    dll_masterAddr = 0xFF;  /* set address of master node */
    dll_nodeAddrInM[0] = 0; /* set address of slave node No. 1 */
    /* button “A” control device with Address = 0 */
    dll_nodeAddrInM[1] = 1; /* set address of slave node No. 2 */
    /* button “B” control device with Address = 1 */
    dll_nodeAddrInM[2] = 2; /* set address of slave node No. 3 */
    /* button “C” control device with Address = 2 */
#endif
    dll_SetStartTBM();  /* set Timebase module for dll functionality */
}


/*********************************************************************************
* Module: void dll_TBModuleISR()
*
* Description: This routine is the Timerbase Module (TBM) interrupt service
*   routine (ISR). TBM interrupts each circa 0.75ms (8192 / 11.0592e6)
*
* Returns: None
*
* Global Data:
*       dll_nevEndCount
*       app_desAnalogVal
*       app_actAnalogVal
*       app_AliveCountInM
*       app_AliveCount
*       dll_FlgFreeLineDet
*       dll_OneSecCount
*       dll_DesiredDelay
*       dll_FlgWaitForACK
*       dll_FlgWaitBefRetr
*
* Arguments: None
*
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
#pragma TRAP_PROC
void dll_TBModuleISR()  /* TBM interrupt service routine */
{
#ifdef SLAVE
   unsigned int tmpIntens; /* temp variable for intensity -> time calculation */
#endif

    dll_TBM_CF();       /* clear flag after TBM interrupt event */
    dll_nevEndCount++;  /* increment counter */


/*****************************/
/* START OF APPLICATION PART */
/*****************************/
/* setting of the analog value for lamp dimmer */
#ifdef SLAVE
    if ((dll_nevEndCount % 6) == 0) /* do increment/decrement each 6 * 0.75ms */
    {
        if (app_desAnalogVal > app_actAnalogVal)     /* increase analog value */
            app_actAnalogVal++;
        else if (app_desAnalogVal < app_actAnalogVal)/* decrease analog value */
            app_actAnalogVal--;

        tmpIntens = (unsigned int) (AC_HALF_PER/0xff) * app_actAnalogVal;
        INTENS_H =  hi(tmpIntens);      /* set intensity */
        INTENS_L =  lo(tmpIntens);
    }
#endif

/* Zerro cross detection routine calling */
#ifdef SLAVE
    app_ZcDetect();    /* Zero crossing detection routine */
#endif

/* Alive message handling for both master and slave side */
    if (dll_nevEndCount == 0xFF)    /* do each 0.75ms * 255 = 191ms */
    {
#ifdef MASTER
        for (i = 0; i < 3; i++)
        {
            app_AliveCountInM[i]++;   /* 1 tick in counter is approx. 191ms */
            if (app_AliveCountInM[0] >= APP_ALIVE_MSTR_LIMIT)   /* device A  */
                ledA_NR();                                      /* NOT READY */
            if (app_AliveCountInM[1] >= APP_ALIVE_MSTR_LIMIT)   /* device B  */
                ledB_NR();                                      /* NOT READY */
            if (app_AliveCountInM[2] >= APP_ALIVE_MSTR_LIMIT)   /* device C  */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
                ledC_NR();                                      /* NOT READY */
        }
#endif
#ifdef SLAVE
        app_AliveCount++;   /* 1 tick in counter is approx. 191ms */
        if (app_AliveCount>=APP_ALIVE_SLV_LIMIT+(NODE_ADDR*APP_ALIVE_SLV_PRIOR))
        {
            app_AliveCount = APP_SEND_ALIVE;
        /* set counter to “Send alive message, message will be send in main() */
        }
#endif
    }
/*****************************/
/* END OF APPLICATION PART   */
/*****************************/

    if(dll_FlgFreeLineDet == 1) /* if Free line detection routine activated */
    {
        dll_OneSecCount++;  /* increment 1.1s long counter */
            /* when 1.1s event occurs (detected in dll_SendKNXData, transmission
               can be started even if not free line were detected */
        dll_DesiredDelay--;         /* dec main delay variable of Free line
                                       detection */
        if (dll_DesiredDelay == 0)  /* delay period is over */
            dll_FlgFreeLineDet = 0; /* finish Free line detection routine */
    }

    if(dll_FlgWaitForACK == 1)  /* if Waiting for ACK routine activated */
    {
        dll_DesiredDelay--;         /* dec main delay variable of Waiting
                                       for ACK */
        if (dll_DesiredDelay == 0)  /* delay period is over */
            dll_FlgWaitForACK = 0;  /* finish Waiting for ACK routine */
    }

    if(dll_FlgWaitBefRetr == 1)  /* if Waiting before retransmit activated */
    {
        dll_DesiredDelay--;         /* dec main delay variable of Waiting before
                                       retransmit */
        if (dll_DesiredDelay == 0)  /* delay period is over */
            dll_FlgWaitBefRetr = 0; /* finish Waiting before retransmit */
    }
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
/*****************************************************************************
* Module: void dll_CalcCRC(char *buf, unsigned char n)
*
* Description:
*       This function generates the 16 bit CRC      16    15   2
*       using the following polynom:               X   + X  + X  + 1
*       Precise CRC computation algorithm definition:
*       - CRC computation algorithm starts with zero
*       - it treats the data msb first
*       - CRC result is not complemented
*
* Returns: calculated CRC value
*
* Global Data:
*       tableCRC[256] - look-up table for 16 bit CRC computation
*
* Arguments:
*       *buffer - pointer to buffer to be calculated
*       n - length of the buffer
*
* Range Issues: None
*
* Special Issues: None
*
* Others: None
*
*****************************************************************************/
unsigned int dll_CalcCRC(unsigned char *buf, unsigned char n)
{
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
  unsigned int crc = 0;
#pragma DATA_SEG DEFAULT

  while (n--)
     crc = tableCRC[((crc>>8) ^ *buf++) & 0xFF] ^ (crc << 8);
  return (crc);
}


/*****************************************************************************
* Module: dll_tTxStatus dll_sendKNXData(unsigned int doa, unsigned char ctrl,
*                          unsigned int sa, unsigned int da, unsigned char npci,
*                          unsigned char * pAppBuf)
*
* Description: This is the data link layer power line transmission routine.
*   It sends the data message of the PL132 format to the power line modem (ST
*   chip) on the application layer level. Control fields of the data message is
*   taken from the parameters of the function.
*
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
* Returns:
*   status with the following declaration of dll_tTxStatus type:
*   DLL_TX_INITIAL -   initial state of the variable describing the transmission
*                      on data link layer
*   DLL_TX_OK -        transmission completed on data link layer,
*                      no ACK signal required
*   DLL_TX_ACK_OK -    transmission completed on data link layer,
*                      ACK required => ACK message received succesfully
*   DLL_TX_ACK_BAD -   transmission not completed on data link layer since
*                      no ACK from 1st initial + 2 retransmit transmissions
*   DLL_TX_BUSOCC -    transmission not completed on data link layer since
*                      unexpected bus occupation during retransmits
*
* Global Data:
*       dll_OneSecCount
*       dll_TxBuf[]
*       dll_nevEndCount
*       dll_DesiredDelay
*       dll_FlgFreeLineDet
*       dll_FlgWaitForACK
*       dll_FlgWaitBefRetr
*
* Arguments: doa - Domain Address (DOA)
*            ctrl - Control Field (CTRL)
*            sa - Source Address (SA)
*            da - Dource Address (DA)
*            npci - Network Protocol Control Information (NPCI)
*            pAppBuf - pointer to application transm. buffer
*
* Range Issues: Note that the implementation supports only short frames format
*   of the Konnex PL132 specification
*
* Special Issues: Only the following bits are taken from the ctrl parameter of
*       the function, rest are set in the routine itself:
*   unsigned char ctrlGroupAddr : 1;     0 - individual frame
*                                        1 - group frame
*   unsigned char ctrlAckReq    : 1;     0 - no Layer 2 ack requested
*                                        1 - Layer 2 ack requested
*   unsigned char ctrlPriority :  2;    11 - low (mandatory for long frames)
*                                       01 - normal (defaul for short frames)
*                                       10 - urgent (reserved for urgent frames)
*                                       00 - system (reserved for high priority
*                                            system config + management)
*
*******************************************************************************/
dll_tTxStatus dll_sendKNXData(unsigned int doa, unsigned char ctrl,
                              unsigned int sa, unsigned int da,
                              unsigned char npci, unsigned char * pAppBuf)
{
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
    dll_tTxStatus status;   /* return values of the function */
    unsigned char retryCount; /* counter of transmission retries */
    unsigned int fcs;       /* calculated frame check sum (CRC value) */
    unsigned char lenLData; /* L_Data length taken from npci information */
    dll_uCTRL tmpCtrl;      /* temp CTRL variable with bitfield operations */
#pragma DATA_SEG DEFAULT

    phs_CD_ID();    /* disable CD interrupt => disable reception */

/* initial variable values */
    status = DLL_TX_INITIAL;    /* set initial state */
    retryCount = 0;             /* clear counter of transmission retries */
    dll_OneSecCount = 0;        /* clear 1.1s counter */

    lenLData = npci & 0xF;      /* length of the L_Data part of message */
    fcs = dll_CalcCRC(pAppBuf, lenLData);     /* CRC calculation */

    tmpCtrl.byte = ctrl; /* CTRL variable with bitfield operations */

/* Transmission preparation */
    dll_TxBuf[0] = KNX_HDR_DATA;    /* 2nd part of the header - L_DATA packet */
    dll_TxBuf[1] = hi(doa);         /* higher part of Domain Address */
    dll_TxBuf[2] = lo(doa);         /* lower part of Domain Address */
    /* default or desired values of some bits of ctrl value */
    tmpCtrl.struc.ctrlLongShort = 1;/* short frame */
    tmpCtrl.struc.ctrlRepeat = 1;   /* first transmission */
    tmpCtrl.struc.ctrlDefault = 3;  /* default values, has to be 11 */
    dll_TxBuf[3] = tmpCtrl.byte;    /* Control Fiels */
    dll_TxBuf[4] = hi(sa);          /* higher part of Source Address */
    dll_TxBuf[5] = lo(sa);          /* lower part of Source Address */
    dll_TxBuf[6] = hi(da);          /* higher part of Destination Address */
    dll_TxBuf[7] = lo(da);          /* lower part of Destination Address */
    dll_TxBuf[8] = npci;            /* Network Protocol Control Information */
    for (i = 0; i < lenLData; i++)  /* store transported L_Data octets */
        dll_TxBuf[9 + i] = *pAppBuf++;
    dll_TxBuf[9 + lenLData] = hi(fcs);      /* store higher byte of CRC */
    dll_TxBuf[9 + lenLData + 1] = lo(fcs);  /* store lower byte of CRC */

/* transmission & retransmission loop */
while(retryCount <= KNX_IMM_RETRIES)
/* number of max retries is given by KNX_IMM_RETRIES, routine tries initial
   transmission + KNX_IMM_RETRIES immediate retransmissions */
{

/************************** Free-line detection ****************************/
    /* it generates delay period 85ms + 0 - 30ms a 5ms; */
    dll_DesiredDelay = (dll_nevEndCount % 7) * DLL_DELAY_5MS + DLL_DELAY_85MS;

    dll_FlgFreeLineDet = 1;             /* initiate delay routine */
    while (dll_FlgFreeLineDet == 1)     /* delay loop uses TBM interrupt */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
    {
    /* *************************/
    /* START OF APPLICATION PART */

        #ifdef SLAVE
             app_ZcDetect();    /* Zero crossing detection routine */
        #endif

    /* END OF APPLICATION PART   */
    /* *************************/

        if (!PHS_CD)      /* if there is CD signal present => bus is occupied */
        {
            /* it generates delay period 85ms + 0 - 30ms a 5ms; */
            dll_DesiredDelay = (dll_nevEndCount%7)*DLL_DELAY_5MS+DLL_DELAY_85MS;
                /* delay counter should be set again */
            if (dll_OneSecCount >= DLL_DELAY_1_1S)    /* if counter > 1.1s */
            {
                dll_FlgFreeLineDet = 0; /* stop delay generation */
                break;
            }
            if (retryCount > 0)     /* CD signal during retransmission */
            {
                status = DLL_TX_BUSOCC;     /* unexpected bus occupation
                                               during retransmits */
                dll_FlgFreeLineDet = 0;     /* stop delay generation */
                break;
            }
        }
    }
    if (status == DLL_TX_BUSOCC)    /* if unexpected bus occupation */
    {
//        break;    /* finish Tx routine without transmission */
                    /* no handling of Bus occupied state in application layer */
    }

/***************************** Transmission *********************************/
    phs_Send(dll_TxBuf, lenLData + 10 + 1);   /* send message */
        /* additional 10 bytes are data link layer frame fields */
        /* additional 1 byte is for second part of the header - data or ACK */

/***************************** Waiting for ACK ******************************/
    if (tmpCtrl.struc.ctrlAckReq == 1)   /* if ACK message required */
    {
        phs_CD_IE();            /* enable CD interrupt => enable reception */
        dll_DesiredDelay = DLL_DELAY_35MS;  /* waiting state 35ms long */
        dll_FlgWaitForACK = 1;              /* initiate delay routine */
        status = DLL_TX_ACK_BAD;            /* set default status value */
        while (dll_FlgWaitForACK == 1)      /* delay loop uses TBM interrupt */
        {
        /* *************************/
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
        /* START OF APPLICATION PART */

            #ifdef SLAVE
                app_ZcDetect();    /* Zero crossing detection routine */
            #endif

        /* END OF APPLICATION PART   */
        /* *************************/

            if(dll_FlgRxACKComp == 1)
            {                        /* if dll flag Reception of ACK completed */
                if (dll_RecFCS == fcs)      /* if message acknowledged OK */
                {
                    status = DLL_TX_ACK_OK; /* set status */
                    dll_FlgWaitForACK = 0;  /* stop Waiting for ACK routine */
                }
                dll_ClearACKFlgs(); /* clear flags of reception */
            }
        }
        phs_CD_ID();          /* disable CD interrupt => disable reception */
    }
    else        /* if ACK is no required */
        status = DLL_TX_OK;     /* transmission completed on data link layer,
                                   no ACK signal required */

    if((status == DLL_TX_OK) || (status == DLL_TX_ACK_OK))  /* if sended */
    {
        break;  /* Tx routine on the ddl layer is finished */
    }
    else        /* not sended yet */
    {
        retryCount++;       /* increment counter of transmission retries */
        /* it generates delay period 0 - 30ms a 4ms; */
        dll_DesiredDelay =  (dll_nevEndCount % 8) * DLL_DELAY_4MS + 1;
        dll_FlgWaitBefRetr = 1;             /* initiate delay routine */
        while (dll_FlgWaitBefRetr == 1);    /* delay loop uses TBM interrupt */
        /* modify packet, set immediate retransmission bit of ctrl octet */
        tmpCtrl.struc.ctrlRepeat = 0;       /* immediate retransmission */
        dll_TxBuf[3] = tmpCtrl.byte;        /* Control Fiels */
    }
}   /* end of while() loop */

/* After transmission delay */ /* TBD */

    phs_CD_IE();            /* enable CD interrupt => enable reception */
    return(status);
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/*****************************************************************************
* Module: void dll_sendKNXACK(uUWord16 fcs)
*
* Description: This is the data link layer power line transmission routine.
*   It sends the ACK message of the PL132 format to the power line modem (ST
*   chip) on the application layer level. Control field of the ACK message
*   (Frame Check Sum) is taken from the parameter of the function.

* Returns: None
*
* Global Data:
*       dll_TxBuf[]
*
* Arguments: fcs - Frame Check Sum (FCS)
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
void dll_sendKNXACK(uUWord16 fcs)
{
    phs_CD_ID();    /* disable CD interrupt => disable reception */
    enableInts();   /* enable interrupts since this routine is called
                       from the Timer 2 output compare ISR! */

/* Transmission preparation */
    dll_TxBuf[0] = KNX_HDR_ACK;     /* 2nd part of the header - ACK packet */
    dll_TxBuf[1] = fcs.byte.msb;    /* higher part of Frame Check Sum */
    dll_TxBuf[2] = fcs.byte.lsb;    /* lower part of Frame Check Sum */

/* Transmission */
    phs_Send(dll_TxBuf, 3);         /* send message */
        /* additional 1 byte is for second part of the header - data or ACK */

/* After transmission delay */  /* TBD */

    disableInts();  /* disable interrupts since this routine is called
                       from the Timer 2 output compare ISR! */
    phs_CD_IE();    /* enable CD interrupt => enable reception */
}


/*****************************************************************************
* Module: void dll_recKNXData(void)
*
* Description: This is the data link layer power line reception routine.
*   It gets the message in physical layer format (phs_RxBuf) and check address,
*   control FCS part of the message; if message requires an acknowledge, it
*   also sends the ACK message.
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
*   If both address and FCS are correct, flag dll_FlgRxDataComp (Reception
*   of Data message completed) is set for the application layer level and data
*   are moved to the application buffer app_RxBuf[].
*   Note that only Destination Address (DA) field is checked, the Source Address
*   (SA) and Domain Addres (DOA) are not checked.
*
* Returns: None
*
* Global Data:
*       phs_RxBuf[]
*       dll_nodeAddr
*       dll_masterAddr
*       app_RxBuf[]
*       dll_FlgRxDataComp
*
* Arguments: None
*
* Range Issues: Note that the implementation supports only short frames format
*   of the Konnex PL132 specification
*
* Special Issues: None
*
*******************************************************************************/
void dll_recKNXData(void)
{
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
    uUWord16 fcs;           /* calculated frame check sum (CRC value) */
    unsigned char lenLData; /* L_Data length taken from npci information */
    unsigned int tmp;       /* temporary variable for received Destination
                               Address */
#pragma DATA_SEG DEFAULT

    tmp = (phs_RxBuf[5] << 8) | phs_RxBuf[6];
                             /* store received Destination Address */
#ifdef SLAVE
    if (tmp == dll_nodeAddr)    /* check the Destination address */
#endif
#ifdef MASTER
    if (tmp == dll_masterAddr)  /* check the Destination address */
#endif
    {
        lenLData = phs_RxBuf[7] & 0xF;
                             /* length of the L_Data part of message */
        fcs.word = dll_CalcCRC(&phs_RxBuf[8], lenLData);  /* CRC calculation */
        if ((fcs.byte.msb == phs_RxBuf[8+lenLData]) &&
            (fcs.byte.lsb == phs_RxBuf[9+lenLData]))   /* check received FCS */
        {
            if (phs_RxBuf[2] & 0x10)    /* if ACK message requested */
            {
                dll_sendKNXACK(fcs);    /* send ACK message */
            }
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
            /* move data to apllication buffer */
            app_RxBuf[0] = lenLData;            /* store length information */
            for (i = 1; i < lenLData + 1; i++)  /* store transported L_Data */
                app_RxBuf[i] = phs_RxBuf[i + 7];/* octets from phs to app buf */

            dll_FlgRxDataComp = 1;  /* dll flag Reception of Data completed */
        }
        else
        {
            phs_FlgRxComp=0;    /* clear phs reception complete flag */
            phs_CD_IE();        /* enable CD interrupt => enable reception */
        }
    }
    else
    {
        phs_FlgRxComp=0;    /* clear phs reception complete flag */
        phs_CD_IE();        /* enable CD interrupt => enable reception */
    }
}


/*****************************************************************************
* Module: void dll_recKNXACK(void)
*
* Description: This is the data link layer power line reception routine for the
*   acknowledge messages.
*   It gets the message in physical layer format (stored in phs_RxBuf) and
*   extract the FCS part of the message to dll_RecFCS buffer for the application
*   while sets dll_FlgRxACKComp flag (Reception of ACK message completed).
*
* Returns: None
*
* Global Data:
*       dll_RecFCS
*       phs_RxBuf
*       dll_FlgRxACKComp
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
void dll_recKNXACK(void)
{
    dll_RecFCS = (phs_RxBuf[0] << 8) | phs_RxBuf[1];
                                /* received FCS information */
    dll_FlgRxACKComp = 1;       /* dll flag Reception of ACK completed */
}
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.6 dll.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: dll.h
*
* Description: This is header file for ‘dll.c’
*
* Modules Included: None
*
* Written by Zdenek Kaspar (R55014)
*
* Revision history:
*       Aug-22-02- Initial coding
*       Oct-03-02- Coding finished
*
*****************************************************************************/
#ifndef _DLL_H
#define _DLL_H


/*****************************************************************************/
/*                TYPE DEFINITIONS                                           */
/*****************************************************************************/
/* type definition of the return value of the dll_sendKNXData function */
typedef enum {
    DLL_TX_INITIAL, /* initial state of the variable describing the transmission
                       on data link layer */
    DLL_TX_OK,      /* transmission completed on data link layer,
                       no ACK signal required */
    DLL_TX_ACK_OK,  /* transmission completed on data link layer,
                       ACK required => ACK message received succesfully */
    DLL_TX_ACK_BAD, /* transmission not completed on data link layer since
                       no ACK from 1st initial + 2 retransmit transmissions */
    DLL_TX_BUSOCC,  /* transmission not completed on data link layer since
                       unexpected bus occupation during retransmits */
} dll_tTxStatus;
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/*****************************************************************************/
/*               S T R U C T U R E S                                         */
/*****************************************************************************/
/* Data link layer structures */

/* Control Field (CTRL) bitfield */
typedef struct {
    unsigned char ctrlPriority :  2;  /*11 - low (mandatory for long frames)
                                        01 - normal (defaul for short frames)
                                        10 - urgent (reserved for urgent frames)
                                        00 - system (reserved for high priority
                                             system config + management) */
    unsigned char ctrlDefault   : 2;  /* default value, has to be 11 */
    unsigned char ctrlAckReq    : 1;  /* 0 - no Layer 2 ack requested
                                         1 - Layer 2 ack requested */
    unsigned char ctrlGroupAddr : 1;  /* 0 - individual frame
                                         1 - group frame */
    unsigned char ctrlRepeat    : 1;  /* 0 - immediate retransmission
                                         1 - first transmission */
    unsigned char ctrlLongShort : 1;  /* 0 - long frame; 1 - short frame */
} dll_sCTRL;
/* Supported settings: flgLongShort = 1 ... short frames supported only */

/* Control Field (CTRL) union */
typedef union {             /* union for Control Field (CTRL) value */
    unsigned charbyte;   /* access as whole byte */
    dll_sCTRL       struc;  /* access in bitfield manner */
} dll_uCTRL;

/* Word => byte conversion structure */
typedef struct {            /* structure of two bytes */
    unsigned charmsb;
    unsigned charlsb;
} sUWord16;

/* Word => byte conversion union */
typedef union {             /*16 bit variable with word and byte access*/
    unsigned intword;   /* access whole word */
    sUWord16        byte;   /* access byte at a time */
} uUWord16;

/* Data link layer flags */
typedef struct {
    unsigned char freeLineDetect : 1;  /* free line detection flag */
    unsigned char waitingForACK  : 1;  /* waiting for ACK message flag */
    unsigned char waitingBefRetr : 1;  /* waiting before retransmission flag */
    unsigned char rxDataComplet  : 1;  /* reception of data packet completed */
    unsigned char rxAckComplet   : 1;  /* reception of ACK packet completed */
} dll_sFlags;
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
/*****************************************************************************/
/*                SHORT-CUT DEFINES                                          */
/*****************************************************************************/
#define dll_FlgFreeLineDet    dll_Flags.freeLineDetect
#define dll_FlgWaitForACK     dll_Flags.waitingForACK
#define dll_FlgWaitBefRetr    dll_Flags.waitingBefRetr
#define dll_FlgRxDataComp     dll_Flags.rxDataComplet
#define dll_FlgRxACKComp      dll_Flags.rxAckComplet


/*****************************************************************************/
/*                P R O T O T Y P E S                                        */
/*****************************************************************************/
void dll_Init(void);
void dll_TBModuleISR();
dll_tTxStatus dll_sendKNXData(unsigned int doa, unsigned char ctrl,
    unsigned int sa,unsigned int da,unsigned char npci,unsigned char * pAppBuf);
void dll_sendKNXACK(uUWord16 fcs);
void dll_recKNXData(void);
void dll_recKNXACK(void);


/*****************************************************************************/
/*                GLOBAL VARIABLES & DEFINES                                 */
/*****************************************************************************/
#define DLL_TXBUF_LEN  15 + 10 + 1  /* maximum length of KNX Tx buffer on data
                                       link layer level  */

#pragma DATA_SEG DEFAULT
extern unsigned char dll_TxBuf[DLL_TXBUF_LEN];  /* Tx data buffer on data link
                                                   layer level */


/*****************************************************************************/
/*                KONNEX specific constants                                  */
/*****************************************************************************/
/* number of immediate retries */
#define KNX_IMM_RETRIES  2

/* Timing defines for data link layer */
/* timing based on TBM interrupts each 0.75 ms */
#define DLL_DELAY_1_1S  1476        /* delay 1.1s long */
#define DLL_DELAY_4MS      5        /* delay  4ms long */
#define DLL_DELAY_5MS      7        /* delay  5ms long */
//#define DLL_DELAY_35MS  47+25     /* delay 35ms long */
#define DLL_DELAY_35MS    47+20     /* delay 35ms long */
                                    /* additional 7.5ms delay long */
#define DLL_DELAY_85MS   113        /* delay 85ms long */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
/*****************************************************************************/
/* Flag clearing routines for reception                                      */
/*****************************************************************************/
/* Flag clearing routine for Data reception */
#define dll_ClearDataFlgs()   dll_FlgRxDataComp = 0; phs_FlgRxComp = 0
/* It clears the dll flag “Reception of Data completed” and phs flag “Reception
   completed” */

/* Flag clearing routine for ACK reception */
#define dll_ClearACKFlgs()    dll_FlgRxACKComp = 0; phs_FlgRxComp = 0
/* It clears the dll flag “Reception of ACK completed” and phs flag “Reception
   completed” */


/*****************************************************************************/
/* Timebase module (TBM) related macros & function style macros              */
/*****************************************************************************/
/* TBM module is used for long-time timing of the data link layer */
#define dll_SetStartTBM()  TBCR = 0x16
        /* set TBM divider to circa 0.75ms interval, enable intr, and start */
#define dll_TBM_CF()       TBCR_TACK=1
        /* clear flag after TBM interrupt event */

#endif
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.7 app.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: app.c
*
* Description: In this file the application layer routines are placed.
*
* Modules Included:
*       app_Init()
*       app_TriacTmrISR()
*       app_ZcDetect()
*       app_SendAlive()
*       app_SlaveReception()
*       app_Send()
*       app_MasterReception()
*       app_ButnONOFF()
*       app_ButnA()
*       app_ButnB()
*       app_ButnC()
*       app_AnalogHand()
*       app_delay()
*       nullHand0()
*
* Written by Zdenek Kaspar (R55014)
*
* Revision history:
*       July-12-02- Initial coding
*       Oct-03-02- Coding finished
*
*****************************************************************************/
#include “hc08gp32.h”   /* HC08GP32 header file, suitable for HC08GR08 */
#include “phs.h”        /* KNX physical layer implementation */
#include “dll.h”        /* KNX data link layer implementation */
#include “app.h”        /* demo application layer implementation */

#include “board.h”      /* hardware dependant definitions for 00145_00 board
                           “KNX PL Master” (MCU based) and “KNX PL Slave” */

#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
extern volatile phs_sFlags phs_Flags;   /* physical layer flags */
extern volatile dll_sFlags dll_Flags;   /* data link layer flags */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
#ifdef MASTER
    extern unsigned int dll_masterAddr;    /* address of the master node */
    extern unsigned int dll_nodeAddrInM[3];/* addresses of three slave nodes
                                              connected to master */

    unsigned char app_indexOfDev;       /* index for State of devices */
                                        /* index = 0, 1 or 2 */
    unsigned char app_deviceState[3];   /* state of devices [lamp is ON/OFF] */
                                        /* state[] = 0 - lamp in device is OFF
                                           state[] = 1 - lamp in device is ON */
    volatile unsigned char app_AliveCountInM[3]; /* counter dedicated for applic
                           used for “are you alive?” message timing in master */

    unsigned char app_sentAnalogVal;  /* last sent potenciometer analog value */
#endif

#ifdef SLAVE
    extern unsigned int dll_nodeAddr;      /* address of the slave node */
    extern unsigned int dll_masterAddrInS; /* address of the connected master */

    volatile unsigned char app_AliveCount; /* counter dedicated for application,
                             used for “are you alive?” message timing in slave */

    unsigned char app_desAnalogVal;   /* desired analog value received in msg */
    unsigned char app_actAnalogVal;   /* actual potenciometer analog value */

#endif

#pragma DATA_SEG DEFAULT
volatile unsigned char app_TxBuf[APP_BUF_LEN];
                        /* appl. buffer used during frame Tx */
volatile unsigned char app_RxBuf[APP_BUF_LEN];
                        /* appl. buffer used during frame Rx */


/*****************************************************************************
* Module: void app_Init()
*
* Description: In this routine all application related initialization is done.
*
* Returns: None
*
* Global Data:
*       app_actAnalogVal
*       app_AliveCountInM[]
*       app_deviceState[]
*       app_indexOfDev
*
* Arguments: None
*
* Range Issues: None
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
*
* Special Issues: None
*
*******************************************************************************/
#pragma INLINE
void app_Init(void)
{
#ifdef SLAVE
  app_TriacInit();          /* setup TRIAC control pins */
  app_TriacTmrInit();
  app_actAnalogVal = 0xFF;  /* default value for analog variable */
#endif

#ifdef MASTER
  app_ADCInit();  /* ADC module init */
  app_AliveCountInM[0] = app_AliveCountInM[1] = app_AliveCountInM[2] = 0;
    /* clear “alive” counters */

  app_deviceState[0] = app_deviceState[1] = app_deviceState[2] = 0;
                       /* all states of lamps are OFF */
  app_indexOfDev = 0;  /* index for State of devices */
                       /* Device “A” is chosen as default */
#endif
}


/*******************************************************************************
* Module: void app_TriacTmrISR()
*
* Description: This is the Timer 1 channel 1 output compare ISR. In this routine
*   the triac is turned on for light intensity generation.
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
#pragma TRAP_PROC
void app_TriacTmrISR(void)  /* triac control timer interrupt service routine */
{
  app_TriacTmrCF();         /* clear triac control timer interrupt flag */

#ifdef SLAVE
  app_TriacBurstOn();       /* switch triac on */
#endif
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
}


#ifdef SLAVE
/*****************************************************************************
* Module: void app_ZcDetect()
*
* Description: This is the zero cross detection routine for the triac control.
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
void app_ZcDetect()
{
    static unsigned char wasZC; /* zero cross temp variable */

    if (wasZC)       /* Zero crossing detection routine */
    {
        if (APP_ZC_INP)     /* from negative to possitive zero cross */
        {
            wasZC=0;        /* set temp zero cross variable */
            app_TriacTmrRst();  /* synchronize Triac control timer */
        }
    }
    else
    {
        if (!APP_ZC_INP)    /* from possitive to negative zero cross */
        {
            wasZC=1;        /* set temp zero cross variable */
            app_TriacTmrRst();  /* synchronize Triac control timer */
        }
    }
}
#endif


#ifdef SLAVE
/*****************************************************************************
* Module: dll_tTxStatus app_SendAlive(void)
*
* Description: This is the data transmission routine of slave on the application
*   layer. Through this messages slaves devices inform the master that they are
*   properly connected and therefor available for communication.
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
*   Message format is fixed to:
*       1st byte = 0xFF - alive message
*       2nd byte = address of the source node
*
* Returns:
*   status with the following declaration of dll_tTxStatus type:
*   DLL_TX_INITIAL -   initial state of the variable describing the transmission
*                      on data link layer
*   DLL_TX_OK -        transmission completed on data link layer,
*                      no ACK signal required
*   DLL_TX_ACK_OK -    transmission completed on data link layer,
*                      ACK required => ACK message received succesfully
*   DLL_TX_ACK_BAD -   transmission not completed on data link layer since
*                      no ACK from 1st initial + 2 retransmit transmissions
*   DLL_TX_BUSOCC -    transmission not completed on data link layer since
*                      unexpected bus occupation during retransmits
*
* Global Data:
*       dll_nodeAddr
*       dll_masterAddrInS
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
#pragma INLINE
dll_tTxStatus app_SendAlive(void)
{
    unsigned char tmpCmd[DEF_VAL_LEN];
    dll_tTxStatus status;   /* status of the Tx operation */

    tmpCmd[0] = 0xFF;                           /* Alive message - 1st byte */
    tmpCmd[1] = (unsigned char) dll_nodeAddr;   /* Alive message - 2nd byte */
    status = dll_sendKNXData(DEF_VAL_DOA, DEF_VAL_CTRL_NA, dll_nodeAddr,
                             dll_masterAddrInS, DEF_VAL_LEN, tmpCmd);
    return (status);
}
#endif


#ifdef SLAVE
/*******************************************************************************
* Module: void app_SlaveReception()
*
* Description: This is the application data reception routine of slave. This
*   kind of messages control the intensity of the lamp and its ON / OFF state.
*
*   NOTE: Higher app_RxBuf[2] byte of message lower the light intensity.
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
*         app_RxBuf[2] is going to    0 => full intensity
*         app_RxBuf[2] is going to 0xFF => zero intensity
*
* Returns: None
*
* Global Data:
*       app_RxBuf[]
*       app_desAnalogVal
*
* Arguments: None
*
* Range Issues:
*
* Special Issues: None
*
*******************************************************************************/
#pragma INLINE
void app_SlaveReception(void)
{
    if ((app_RxBuf[0] == 2) && (app_RxBuf[1] == '1'))
    {                             /* length = 2 and SWITCH ON command */
        if (app_RxBuf[2] <= POT_DELTA)   /* SWITCH ON forever command when full
                                            intensity set */
        {
            app_TriacTmrID();   /* disable TRIAC switch ON routine*/
            app_TriacBurstOnForever(); /* and switch TRIAC ON forewer */
        }
        else if (app_RxBuf[2] >= 0xFF - POT_DELTA)
        {                         /* SWITCH OFF command when zero intensity set */
            app_TriacTmrID();   /* disable TRIAC switch ON routine*/
            triacOff();         /* switch off triac */
        }
        else                      /* SWITCH to desired intensity */
        {
            app_desAnalogVal = app_RxBuf[2]; /* desired value received in msg */
            app_TriacTmrIE();   /* enable TRIAC switch ON routine */
        }
        ledIndOn();
    }
    if ((app_RxBuf[0] == 2) && (app_RxBuf[1] == '0'))
                                  /* length = 2 and SWITCH OFF command */
    {
        app_TriacTmrID();   /* disable TRIAC switch ON routine */
        triacOff();         /* switch off triac */
        ledIndOff();
    }
    dll_ClearDataFlgs();    /* Flag clearing routine for Data Rx */
}
#endif
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
#ifdef MASTER
/*****************************************************************************
* Module: dll_tTxStatus app_Send(unsigned char indexOfDevice,
*                      unsigned char desiredACK, unsigned char cmds[DEF_VAL_LEN])
*
* Description:This is the data transmission routine of master on the application
*   layer. Argument values are incorporated into the message, rest of values
*   are taken as defaults.
*
* Returns:
*   status with the following declaration of dll_tTxStatus type:
*   DLL_TX_INITIAL -   initial state of the variable describing the transmission
*                      on data link layer
*   DLL_TX_OK -        transmission completed on data link layer,
*                      no ACK signal required
*   DLL_TX_ACK_OK -    transmission completed on data link layer,
*                      ACK required => ACK message received succesfully
*   DLL_TX_ACK_BAD -   transmission not completed on data link layer since
*                      no ACK from 1st initial + 2 retransmit transmissions
*   DLL_TX_BUSOCC -    transmission not completed on data link layer since
*                      unexpected bus occupation during retransmits
*
* Global Data:
*       dll_masterAddr
*       dll_nodeAddrInM[]
*
* Arguments:
*       indexOfDevice - index of device to where to send the message
*       desiredACK = ACK_OFF is ACK are not required
*                  = ACK_ON is ACK are required
*       cmds[DEF_VAL_LEN] - message to be send
*
* Range Issues: Note that the implementation supports only short frames format
*   of the Konnex PL132 specification
*
* Special Issues: None
*
*****************************************************************************/
dll_tTxStatus app_Send(unsigned char indexOfDevice, unsigned char desiredACK,
                      unsigned char cmds[DEF_VAL_LEN])
{
    dll_tTxStatus status;   /* status of the Tx operation */
    unsigned char tmpACK;   /* temporary variable of ACK status */

    if (desiredACK == ACK_OFF)      /* if ACK disabled */
        tmpACK = DEF_VAL_CTRL_NA;   /* set no ACKed messages */
    else if (desiredACK == ACK_ON)  /* if ACK enabled */
        tmpACK = DEF_VAL_CTRL_ACK;  /* ACK enabled messages */

    status = dll_sendKNXData(DEF_VAL_DOA, tmpACK, dll_masterAddr,
                             dll_nodeAddrInM[indexOfDevice], DEF_VAL_LEN, cmds);
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
    return (status);
}
#endif


#ifdef MASTER
/*****************************************************************************
* Module: void app_MasterReception(void)
*
* Description: This is the “Alive messages” reception routine of master. Through
*   this kind of messages master detect the availability of the connected slaves.
*
* Returns: None
*
* Global Data:
*       app_RxBuf[]
*       app_AliveCountInM[]
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
#pragma INLINE
void app_MasterReception(void)
{
    if ((app_RxBuf[0] == 2) && (app_RxBuf[1] == 0xFF))
                            /* length = 2 and ALIVE command */
    {
        if (app_RxBuf[2] == 0)
        {
            ledA_Rd();                  /* device A alive */
            app_AliveCountInM[0] = 0;   /* clear counter */
        }
        else if (app_RxBuf[2] == 1)
        {
            ledB_Rd();                  /* device B alive */
            app_AliveCountInM[1] = 0;   /* clear counter */
        }
        else if (app_RxBuf[2] == 2)
        {
            ledC_Rd();                  /* device C alive */
            app_AliveCountInM[2] = 0;   /* clear counter */
        }
    }
    dll_ClearDataFlgs();       /* Flag clearing routine for Data Rx */
}
#endif
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
#ifdef MASTER
/*****************************************************************************
* Module: void app_ButnONOFF()
*
* Description: In this routine the ON/OFF button processing is done of master.
*   It send the ON / OFF message to active (given by app_indexOfDev) slave.
*   Note that this message requires the ACKnowledge. When ACKnowledge received,
*   it serves also as the "Alive message" reception from the slave device.
*
* Returns: None
*
* Global Data:
*       app_deviceState[]
*       app_indexOfDev
*       app_sentAnalogVal
*       app_AliveCountInM[]
*
* Arguments: None
*
* Range Issues:
*
* Special Issues: None
*
*****************************************************************************/
#pragma INLINE
void app_ButnONOFF(void)
{
    dll_tTxStatus status;       /* status of the transmission */

    if (app_deviceState[app_indexOfDev] == 0)   /* if OFF */
    {
           app_TxBuf[0] = '1';                  /* send ON command */
       app_sentAnalogVal = POT_INT_CTRL;   /* store analog value */
        app_TxBuf[1] = POT_INT_CTRL;        /* write analog value into message*/
        status = app_Send(app_indexOfDev, ACK_ON, app_TxBuf);
                                            /* send message with ACKs enabled */
        if (status == DLL_TX_ACK_OK)        /* if message ACKed */
        {
            app_deviceState[app_indexOfDev] = 1;    /* set ON */
           ledPowOn();                     /* LED indication is ON */

            app_AliveCountInM[app_indexOfDev] = 0;  /* clear counter */
            switch (app_indexOfDev)
            {                               /* refresh LED states */
                case 0 :    ledA_Rd();
                            break;
                case 1 :    ledB_Rd();
                            break;
                case 2 :    ledC_Rd();
                            break;
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
            }
        }
       }
    else                                    /* if ON */
    {
           app_TxBuf[0] = '0';                 /* send OFF command */
       app_sentAnalogVal = POT_INT_CTRL;   /* store analog value */
        app_TxBuf[1] = POT_INT_CTRL;        /* write analog value into message*/
        status = app_Send(app_indexOfDev, ACK_ON, app_TxBuf);
                                            /* send message with ACKs enabled */
        if (status == DLL_TX_ACK_OK)        /* if message ACKed */
        {
            app_deviceState[app_indexOfDev] = 0;    /* set OFF */
           ledPowOff();                    /* LED indication is OFF */

            app_AliveCountInM[app_indexOfDev] = 0;  /* clear counter */
            switch (app_indexOfDev)
            {                               /* refresh LED states */
                case 0 :    ledA_Rd();
                            break;
                case 1 :    ledB_Rd();
                            break;
                case 2 :    ledC_Rd();
                            break;
            }
        }
       }
    while (SW_ON_OFF);      /* wait for button release */
    app_delay();            /* button delay routine */
}
#endif


#ifdef MASTER
/*****************************************************************************
* Module: void app_ButnA()
*
* Description: In this routine Device "A" button processing is done of master.
*   It send the ON message to the device and set it as active one (set
*   app_indexOfDev = 0)
*   Note that this message requires the ACKnowledge. When ACKnowledge received,
*   it serves also as the "Alive message" reception from the slave device.
*
* Returns: None
*
* Global Data:
*       app_indexOfDev
*       app_TxBuf[]
*       app_AliveCountInM[]
*
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
#pragma INLINE
void app_ButnA(void)
{
    app_indexOfDev = 0;         /* set index “Which slave device is chosen?” */
                                /* Device “A” is chosen */
    if (app_deviceState[app_indexOfDev] == 1)   /* if device is ON */
           ledPowOn();                          /* set LED indication to ON */
       else ledPowOff();                        /* else set it to OFF state */
       while (SW_CTRL_DEV_A);      /* wait for button release */
}
#endif


#ifdef MASTER
/*****************************************************************************
* Module: void app_ButnB()
*
* Description: In this routine Device “B” button processing is done of master.
*   It send the ON message to the device and set it as active one (set
*   app_indexOfDev = 1)
*   Note that this message requires the ACKnowledge. When ACKnowledge received,
*   it serves also as the “Alive message” reception from the slave device.
*
* Returns: None
*
* Global Data:
*       app_indexOfDev
*       app_TxBuf[]
*       app_AliveCountInM[]
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
#pragma INLINE
void app_ButnB(void)
{
    app_indexOfDev = 1;         /* set index “Which slave device is chosen?” */
                                /* Device “B” is chosen */
    if (app_deviceState[app_indexOfDev] == 1)   /* if device is ON */
           ledPowOn();                          /* set LED indication to ON */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
       else ledPowOff();                          /* else set it to OFF state */
       while (SW_CTRL_DEV_B);      /* wait for button release */
}
#endif


#ifdef MASTER
/*****************************************************************************
* Module: void app_ButnC()
*
* Description: In this routine Device "C" button processing is done of master.
*   It send the ON message to the device and set it as active one (set
*   app_indexOfDev = 2)
*   Note that this message requires the ACKnowledge. When ACKnowledge received,
*   it serves also as the "Alive message" reception from the slave device.
*
* Returns: None
*
* Global Data:
*       app_indexOfDev
*       app_TxBuf[]
*       app_AliveCountInM[]
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
#pragma INLINE
void app_ButnC(void)
{
    app_indexOfDev = 2;         /* set index "Which slave device is chosen?" */
                                /* Device "C" is chosen */
    if (app_deviceState[app_indexOfDev] == 1)   /* if device is ON */
          ledPowOn();                           /* set LED indication to ON */
       else ledPowOff();                        /* else set it to OFF state */
       while (SW_CTRL_DEV_C);      /* wait for button release */
}
#endif


#ifdef MASTER
/*****************************************************************************
* Module: void app_AnalogHand()
*
* Description: In this routine the analog potentiometer value handling of master
*   is done. When analog value exceeds the POT_DELTA hysteresis value, a proper
*   message is sent to the slave with new desired value of the light intensity.
*
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
* Returns: None
*
* Global Data:
*       app_sentAnalogVal
*       app_deviceState
*       app_indexOfDev
*       app_TxBuf
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
#pragma INLINE
void app_AnalogHand(void)
{
    unsigned char hiLimit;      /* for analog value processing */
    unsigned char loLimit;      /* for analog value processing */
    dll_tTxStatus status;       /* status of the transmission */

    if (app_sentAnalogVal < (0xFF - POT_DELTA))  /* set analog high limit val*/
        hiLimit = app_sentAnalogVal + POT_DELTA; /* calculated high limit val*/
    else hiLimit = 0xFF;                         /* limited high limit value */

    if (app_sentAnalogVal > POT_DELTA)           /* set analog low limit */
        loLimit = app_sentAnalogVal - POT_DELTA; /* calculated low limit val */
    else loLimit = 0;                            /* limited low limit value */

    if ((POT_INT_CTRL > hiLimit) || (POT_INT_CTRL < loLimit))
    {
        if (app_deviceState[app_indexOfDev] == 1)  /* only if Device is ON! */
        {
           app_TxBuf[0] = '1';     /* send ON command */
        app_TxBuf[1] = app_sentAnalogVal = POT_INT_CTRL;
                              /* store analog value */
            status = app_Send(app_indexOfDev, ACK_OFF, app_TxBuf);
                                  /* send message with ACKs disabled */
        }
    }
}
#endif


#ifdef MASTER
/*******************************************************************************
* Module: void app_delay()
*
* Description: This is the button delay routine for power ON / OFF button.
*
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*******************************************************************************/
#pragma INLINE
void app_delay(void)
{
    unsigned char i;

    for(i=0; i < 0xFF; i++);
    {
        asm(nop); asm(nop);
    }
}
#endif


/*******************************************************************************
* Module: void nullHand0()
*
* Description: This is the unserviced interrupt ISR routine.
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: For debugging only.
*
* Special Issues: None
*
*******************************************************************************/
#pragma TRAP_PROC
void nullHand0(void) /* all unhandled interrupt requests will be trapped here */
{
  while (1);
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.8 app.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: app.h
*
* Description: This is header file for ‘app.c’
*
* Modules Included: None
*
* Written by Zdenek Kaspar (R55014)
*
* Revision history:
*       July-18-02- Initial coding
*       Oct-03-02- Coding finished
*
*****************************************************************************/
#ifndef _APP_H
#define _APP_H


/*****************************************************************************/
/*               APPLICATION DEFINES FOR TRANSMISSION                        */
/*****************************************************************************/
#define APP_BUF_LEN  16     /* maximum length of buffer on application level */
#define DEF_VAL_LEN   2     /* defined length of the messages on appl level */

/* Default values of the KNX message octets */
#define DEF_VAL_DOA         0x1     /* default value of Domain Address (DOA) */
#define DEF_VAL_CTRL_ACK   0xDD     /* default value of Control Field (CTRL) */
                                    /* with ACKs */
#define DEF_VAL_CTRL_NA    0xCD     /* default value of Control Field (CTRL) */
                                    /* without ACKs */
    /* LSB: ctrlPriority = 01 - normal (defaul for short frames)
       ctrlDefault = 11 - has to be set to 11
       ctrlAckReq = 1 - Layer 2 ack requested
                  = 0 - Layer 2 ack not requested
       ctrlGroupAddr = 0 - individual frame
       ctrlRepeat = 1 - first transmission
       MSB: ctrlLongShort = 1 - short frame  */
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
/*****************************************************************************/
/*                APPLICATION DEFINES FOR ALIVE MESSAGES                     */
/*****************************************************************************/
/* defines are used in TBM ISR routine for timing of “Alive message”
   transmission / reception */
#ifdef MASTER
#define APP_ALIVE_MSTR_LIMIT 25 /* time period of “Go to device is not ready” */
                                /* 1 tick = 191ms */
#define ACK_ON      1   /* ACKed messages desired */
#define ACK_OFF     0   /* non ACKed messages desired */
#endif

#ifdef SLAVE
#define APP_ALIVE_SLV_LIMIT  8   /* time period of “Send alive messages” */
                                 /* 1 tick = 191ms */
#define APP_ALIVE_SLV_PRIOR  2   /* priority part of time period for “Send alive
                                    messages” */
                                 /* 1 tick = 191ms */
#define APP_SEND_ALIVE     0xF0  /* when counter is equal to this value,
                                    alive message will be send in main() */
#endif


/*****************************************************************************/
/*                P R O T O T Y P E S                                        */
/*****************************************************************************/
void app_Init(void);            /* application related initialization */

#ifdef SLAVE
void app_ZcDetect(void);        /* zero cross detection routine */
void app_SlaveReception(void);      /* slave reception routine */
dll_tTxStatus app_SendAlive(void);  /* application message from slave */
#endif

#ifdef MASTER
dll_tTxStatus app_Send(unsigned char indexOfDevice, unsigned char desiredACK,
                       unsigned char cmds[DEF_VAL_LEN]);
                                /* application message from master */
void app_ButnONOFF(void);       /* button ON/OFF handling routine */
void app_ButnA(void);           /* button Device “A” handling routine */
void app_ButnB(void);           /* button Device “B” handling routine */
void app_ButnC(void);           /* button Device “C” handling routine */
void app_AnalogHand(void);      /* potentiometer analog value handling */
void app_MasterReception(void); /* reception routine for “Alive msgs” */
void app_delay(void);           /* button delay routine */
#endif

void nullHand0(void);           /* unserviced ISR */
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
/*****************************************************************************/
/*     APPLICATION I / O RELATED DEFINES                                     */
/*****************************************************************************/
#ifdef MASTER
#define SW_ON_OFF       SW4     /* switch on / off */
#define SW_CTRL_DEV_A   SW1     /* device A control */
#define SW_CTRL_DEV_B   SW2     /* device B control */
#define SW_CTRL_DEV_C   SW3     /* device C control */

#define LED_A           IN1     /* device A status LED */
#define LED_B           IN2     /* device B status LED */
#define LED_C           IN3     /* device C status LED */
#define LED_POW         IN4     /* power ON/OFF LED */

#define ledA_NR()       LED_A=0 /* “not ready” states */
#define ledB_NR()       LED_B=0
#define ledC_NR()       LED_C=0
#define ledPowOff()     LED_POW=0 /* “power OFF” state */

#define ledA_Rd()       LED_A=1 /* “ready” state */
#define ledB_Rd()       LED_B=1
#define ledC_Rd()       LED_C=1
#define ledPowOn()      LED_POW=1 /* “power ON” state */
#endif

#ifdef SLAVE
#define LED_IND         INDICAT /* LED indication of slave */
#define ledIndOn()      LED_IND=0   /* indication on */
#define ledIndOff()     LED_IND=1   /* indication off */
#endif


/*****************************************************************************/
/*     TRIAC CONTROL RELATED DEFINES                                         */
/*****************************************************************************/
#ifdef SLAVE

/* timing */
#define AC_FREQ         50          /* 50Hz in mains for Europe */
#define AC_HALF_PER     BUS_CLK / AC_FREQ / 2

#define INTENS_H        T1CH1H      /* light intensity control registers */
#define INTENS_L        T1CH1L

/* pins related */
#define APP_ZC_INP      ZC_SENSE    /* zerro cross input for triac */
#define TRIAC_CTRL      0x03        /* triac control pins */

#define triacOn()       PTC|=TRIAC_CTRL
#define triacOff()      PTC&=~TRIAC_CTRL
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/* funtion style macros */
#define app_TriacInit()     triacOff(); DDRC|=TRIAC_CTRL; triacOff()

 /* generate burst signal to assure that triak will be really switched ON */
#define app_TriacBurstOn() triacOn(); asm nop; asm nop; triacOff(); asm nop; \
                        asm nop; triacOn(); asm nop; asm nop; triacOff(); \
                        asm nop; asm nop; triacOn(); asm nop; asm nop;triacOff()

 /* generate burst signal to assure that triak will be really switched ON */
#define app_TriacBurstOnForever()  triacOn(); asm nop; asm nop; triacOff(); \
                          asm nop; asm nop; triacOn(); asm nop; asm nop; \
                          triacOff(); asm nop; asm nop; triacOn(); asm nop; \
                          asm nop; triacOff(); asm nop; asm nop; triacOn()

/* Timer 1 - Triac control timer related function style macros */
#define app_TriacTmrRst()   T1SC_TRST=1   /* synchornize (reset) triac Timer 1
                                    to reflect the zero crossing in AC line */

/* app_TriacTmrInit() routine set the Timer1 as the triac control timer:
   - stop & reset Tmr1 counter
   - set period of Timer 2 modulo counter for overflow event to half period of
     AC line
   - finally set the mode T1CH1 to output compare (no interrupt enabled) */
#define app_TriacTmrInit()  T1SC=0x30; \
                            T1MODH=AC_HALF_PER/256; T1MODL=AC_HALF_PER % 256; \
                            T1SC1=0x14
#endif

/* Triac timer (Tmr1Ch1 output compare) interrupt related func style macros */
#define app_TriacTmrCF()    T1SC1_CH1F=0
                    /* clear interrupt flag of triac control timer */
#define app_TriacTmrIE()    T1SC1_CH1IE=1
                    /* triac control timer interrupt enable */
#define app_TriacTmrID()    T1SC1_CH1IE=0
                    /* triac control timer interrupt disable */


/*************************************************************************/
/*     MASTER A/D CONTROL RELATED DEFINES                                */
/*************************************************************************/
#define POT_DELTA       20          /* for analog potentiometer calculation */
#ifdef MASTER
#define POT_INT_CTRL    (ADR)       /* master analog input */
#define app_ADCInit()   ADSCR=0x25; /* ADC in continuos mode,AD5 pin selected*/ \
                        ADCLK=0x30  /* ADC clock settings */
#endif

#endif
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.9 main.c

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: main.c
*
* Description: Main code for the demo Konnex PL132 physical and data link layer
*       implementation project based on the ST power line modem ST7537 and
*       Motorola 8bit MCU HC08GR8.
*
* Modules Included:
*       init()
*       main()
*
* Written by Marek Stricek (R29303)
* Further development by Zdenek Kaspar (R55014)
*
* Revision history:
*       May-28-02- Initial coding
*       Oct-03-02- Coding finished
*
*****************************************************************************/
#include “hc08gp32.h”   /* HC08GP32 header file, suitable for HC08GR08 */
#include “phs.h”        /* KNX physical layer implementation */
#include “dll.h”        /* KNX data link layer implementation */
#include “app.h”        /* demo application layer implementation */

#include “board.h”      /* hardware dependant definitions for 00145_00 board
                           “KNX PL Master” (MCU based) and “KNX PL Slave” */


/*****************************************************************************/
/*              GLOBAL VARIABLES                                             */
/*****************************************************************************/
#pragma DATA_SEG __SHORT_SEG MY_ZEROPAGE
extern volatile phs_sFlags phs_Flags;   /* physical layer flags */
extern volatile dll_sFlags dll_Flags;   /* data link layer flags */

#ifdef SLAVE
    extern volatile unsigned char app_AliveCount; /* counter dedicated for
             application, used for “are you alive?” message timing in slave */
#endif
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

```
#pragma DATA_SEG DEFAULT
extern volatile unsigned char app_TxBuf[APP_BUF_LEN];
                        /* appl. buffer used during frame Tx */
extern volatile unsigned char app_RxBuf[APP_BUF_LEN];
                        /* appl. buffer used during frame Rx */


/*****************************************************************************
* Module: void init()
*
* Description: This is the complete initialization routine of the KNX PL132 phs
*   (physical layer), KNX PL132 dll (data link layer) as well as the KNX demo
*   application.
*
* Returns: None
*
* Global Data: None
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
void init(void)
{
    swchInit();     /* input pins init */
       ledInit();      /* output pins init */
    app_Init();     /* application related initialization */
       phs_Init();     /* ST connection pins & physical link layer init */
       dll_Init();     /* data link layer init */

       phs_IRQ_ID();   /* !!! FOR DEBUG ONLY disable interrupts from PLM WD */
       enableInts();   /* enable interupts */

#ifdef SLAVE
    ledIndOff();        /* init state of Slave LED */
#endif
#ifdef MASTER
    ledPowOff();                        /* init state of ON / OFF LED */
    ledA_NR(); ledB_NR(); ledC_NR();    /* init state of device A, B, C LEDs */
#endif
}
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
/*****************************************************************************
* Module: void main()
*
* Description: This is the main routine of the KNX demo application
*
* Returns: None
*
* Global Data:
*       dll_FlgRxDataComp
*       app_AliveCount
*
* Arguments: None
*
* Range Issues: None
*
* Special Issues: None
*
*****************************************************************************/
void main(void)
{
#ifdef SLAVE
    dll_tTxStatus status;   /* status of the transmission */
#endif

    init();         /* complete initialization of phs, dll as well as appl */

    while (1)
    {
           phs_WDServ();   /* watch-dog serving */

/*****************************************************************************/
/* start of the SLAVE part                                                   */
/*****************************************************************************/
#ifdef SLAVE
        app_ZcDetect(); /* Zero crossing detection routine */

        if (dll_FlgRxDataComp)          /* if frame successfuly received */
            app_SlaveReception();       /* call reception routine */

        if (app_AliveCount >= APP_SEND_ALIVE) /* condition for “device alive” */
        {
            app_AliveCount = 0;         /* clear counter */
            status = app_SendAlive();   /* send alive message */
        }
#endif /* end of the SLAVE part */
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/*****************************************************************************/
/* start of the MASTER part                                                  */
/*****************************************************************************/
#ifdef MASTER
/* ON/OFF button proccess */
         if (SW_ON_OFF)        /* if key is pressed then TRUE */
            app_ButnONOFF();   /* ON/OFF button processing */

/* Device “A” button proccess */
         if (SW_CTRL_DEV_A)
            app_ButnA();       /* Device “A” button processing */

/* Device “B” button proccess */
         if (SW_CTRL_DEV_B)
            app_ButnB();       /* Device “B” button processing */

/* Device “C” button proccess */
         if (SW_CTRL_DEV_C)
            app_ButnC();       /* Device “C” button processing */

/* Analog value handling proccess */
      app_AnalogHand();     /* Potentiometr analog value handling */

/* Alive messages reception */
      if (dll_FlgRxDataComp)    /* if frame successfuly received */
         app_MasterReception(); /* call reception routine for “Alive msg” */
#endif  /* end of the MASTER part */

   }
}   /* end of main() */
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.10 board.h

```
/*****************************************************************************
*
* Motorola Inc.
* (c) Copyright 2002 Motorola, Inc.
* ALL RIGHTS RESERVED.
*
*****************************************************************************
*
* File Name: board.h
*
* Description: This file contains hardware dependant definitions for 00145_00
*       board KNX PL Master (MCU based) and KNX PL Slave
*
* Modules Included:
*
* Written by Marek Stricek (R29303)
* Further development by Zdenek Kaspar (R55014)
*
* Revision history:
*       May-28-02- Initial coding
*       Oct-03-02- Coding finished
*
*****************************************************************************/
#ifndef _BOARD_H
#define _BOARD_H


/*****************************************************************************/
/* HC08 dependent defines                                                    */
/*****************************************************************************/
/* Interrupt Control macros */
#define enableInts()    asm cli
#define disableInts()   asm sei

/* Illegal operation definition used for reset the MCU */
#define illegalOperation()   asm DCB 0x32

/* Conversion from unsigned int to unsigned char */
#define hi(x) ((unsigned char)((x)>>8))
#define lo(x) ((unsigned char)(x))

/*****************************************************************************/
/* Timing related defines                                                    */
/*****************************************************************************/
/* Timing parameters */
#define XTAL_FREQ   11059200L   /* crystal frequency used on board */
#define BUS_CLK     XTAL_FREQ/4
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/*****************************************************************************/
/* Input pins related defines                                                */
/*****************************************************************************/
#ifdef MASTER
#define swchInit()       DDRA&=~0x0e; PTAPUE|=0x0e; DDRD_BIT5=0; PTDPUE_BIT5=1
    /* set SW1 - SW4 as inputs and switch on pull-ups */

/* SWx are name of signals (nets) in schematic, not references to switches
   itself */
/* if key is pressed then it has TRUE value */
#define SW1    !(PTA_BIT1)     /* bit 1 PA */
#define SW2    !(PTA_BIT2)     /* bit 2 PA */
#define SW3    !(PTA_BIT3)     /* bit 3 PA */
#define SW4    !(PTD_BIT5)     /* bit 5 PD */

#endif

#ifdef SLAVE
#define swchInit()       DDRD&=~0x0f; PTDPUE|=0x0f; DDRA&=~0x02
    /* set JP3 - JP6 as inputs and switch on pull-ups */
    /* set ZC_SENCE as input */

/* ZC_SENSE is a name of signal (net) in schematic, not references to dig. input
   itself */
#define ZC_SENSE        PTA1

#define NODE_ADDR    ((~PTD)&0x0f)  /* Node address taken from JP3-6 */

#endif

/*****************************************************************************/
/* Output pins related defines                                               */
/*****************************************************************************/
#ifdef MASTER
#define ledInit()       DDRD|=0x0f; PTD&=~0x0f
    /* set IN1 - IN4 as outputs for LED indicators */

/* INx are name of signals (nets) in schematic, not references to switches
   itself */
#define IN1             PTD_BIT0  /* bit 0 PD */
#define IN2             PTD_BIT1  /* bit 1 PD */
#define IN3             PTD_BIT2  /* bit 2 PD */
#define IN4             PTD_BIT3  /* bit 3 PD */

#endif

#ifdef SLAVE
#define ledInit()       DDRB_BIT4=1; PTB_BIT4=1
    /* set PTB4 as outputs for LED indicator */
/* INDICAT is name of signal (net) in schematic, not references to digit. output
   itself */
#define INDICAT         PTB_BIT4

#endif

#endif
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

7.11 hc08gp32.h

```
/*********************************************************************
* HEADER_START                                                       *
*                                                                    *
*      Name:           hc08gp32.h                                    *
*      Project:        Interconnectivity SRDT                        *
*      Description:    HC08GP32 header file                          *
*      Processor:      HC08                                          *
*      Revision:       1.0                                           *
*      Date:           Feb 26 2002                                   *
*      Compiler:       HI-CROSS+ Compiler for HC08 V-5.0.11 ICG      *
*      Author:         Pavel Lajsner                                 *
*      Based on:       original KX6/8 header file by William Mackay  *
*                      & Ken Berringer                               *
*      Company:        Motorola SPS                                  *
*      Security:       General Business                              *
*                                                                    *
* ================================================================== *
* Copyright (c):      MOTOROLA Inc., 2001, All rights reserved.      *
*                                                                    *
* ================================================================== *
* THIS SOFTWARE IS PROVIDED BY MOTOROLA “AS IS” AND ANY              *
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE *
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  *
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL MOTOROLA OR             *
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,    *
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT        *
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;        *
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)            *
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, *
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)       *
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED *
* OF THE POSSIBILITY OF SUCH DAMAGE.                                 *
* ================================================================== *
*
* HEADER_END
*/
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/*************************************************************************/
/*************** HC08 device mapping for 68HC908GP32            ************/
/*************************************************************************/
/*

Register Definitions

    This header file defines all of the registers using the register name exactly
    as listed in the data book. Each register name is entered in ALL CAPS. The
    registers are all unsigned volatile bytes to ensure that the compiler will
    not remove sequential write commands. This still does not guarentee that
    the compiler will execute each assignment exactly as coded, as some compilers
    may still try to optimize volatile bytes. The compiler should be tested
    with this header file to ensure register assignments are not optimized.

Bitfields

    Each bit is defined using the following format:

        REGISTERNAME_BITNAME

    While this may result in redundant names in some cases, it will
    always prevent duplicate bit names. If a bit has a descriptive name
    such as FLG the descriptive name is used.

    If the bit does not have a descriptive name the generic terms BIT0
    through BIT7 are used.

    In some cases the data book may define bits by adding a digit to the
    register name. In these cases the generic BIT0 names are used
    instead. For eample, DDRA_BIT0 is used instead of the redundant
    DDRA_DDRA0. Short Aliases are defined to provide compatibility for
    these particular cases.


Exceptions

Short Aliases

    Short Aliases are defined for particular cases where the customary usage
    is to use short bit names. For example, it is customary to use the short
    term PTA0 instead of PTA_BIT0. This header permits both usages.

 */

#ifndef __hc08gp32_h
#define __hc08gp32_h      /* if this H file is not included, include */


#define BIT0 0x01
#define BIT1 0x02
#define BIT2 0x04
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
#define BIT3 0x08
#define BIT4 0x10
#define BIT5 0x20
#define BIT6 0x40
#define BIT7 0x80

/*************************************************************************/
/* Register Mapping Structures and Macros                                */
/*************************************************************************/

#define DBLREG(a)  (*((volatile unsigned int *)(a)))
#define REGISTER(a)  (*((volatile unsigned char *)(a)))
#define BIT(a,b)  (((vbitfield *)(a))->bit##b)

/* assumes right to left bit order, highware default */

typedef volatile struct{
  volatile unsigned int bit0   : 1;
  volatile unsigned int bit1   : 1;
  volatile unsigned int bit2   : 1;
  volatile unsigned int bit3   : 1;
  volatile unsigned int bit4   : 1;
  volatile unsigned int bit5   : 1;
  volatile unsigned int bit6   : 1;
  volatile unsigned int bit7   : 1;
} vbitfield;

/*************************************************************************/
/* Input Output Ports                                                    */
/*************************************************************************/

/* Port A Data register */
#define  PTA          REGISTER(0x00)
#define  PTA_BIT0     BIT(0x00,0)
#define  PTA_BIT1     BIT(0x00,1)
#define  PTA_BIT2     BIT(0x00,2)
#define  PTA_BIT3     BIT(0x00,3)
#define  PTA_BIT4     BIT(0x00,4)
#define  PTA_BIT5     BIT(0x00,5)
#define  PTA_BIT6     BIT(0x00,6)
#define  PTA_BIT7     BIT(0x00,7)


/* Port B Data register */
#define  PTB          REGISTER(0x01)
#define  PTB_BIT0     BIT(0x01,0)
#define  PTB_BIT1     BIT(0x01,1)
#define  PTB_BIT2     BIT(0x01,2)
#define  PTB_BIT3     BIT(0x01,3)
#define  PTB_BIT4     BIT(0x01,4)
#define  PTB_BIT5     BIT(0x01,5)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
#define  PTB_BIT6     BIT(0x01,6)
#define  PTB_BIT7     BIT(0x01,7)


/* Port C Data register */
#define  PTC          REGISTER(0x02)
#define  PTC_BIT0     BIT(0x02,0)
#define  PTC_BIT1     BIT(0x02,1)
#define  PTC_BIT2     BIT(0x02,2)
#define  PTC_BIT3     BIT(0x02,3)
#define  PTC_BIT4     BIT(0x02,4)
#define  PTC_BIT5     BIT(0x02,5)
#define  PTC_BIT6     BIT(0x02,6)


/* Port D Data register */
#define  PTD          REGISTER(0x03)
#define  PTD_BIT0     BIT(0x03,0)
#define  PTD_BIT1     BIT(0x03,1)
#define  PTD_BIT2     BIT(0x03,2)
#define  PTD_BIT3     BIT(0x03,3)
#define  PTD_BIT4     BIT(0x03,4)
#define  PTD_BIT5     BIT(0x03,5)
#define  PTD_BIT6     BIT(0x03,6)
#define  PTD_BIT7     BIT(0x03,7)


/* Port E Data register */
#define  PTE          REGISTER(0x08)
#define  PTE_BIT0     BIT(0x08,0)
#define  PTE_BIT1     BIT(0x08,1)


/* Port A Data Direction Register */
#define  DDRA           REGISTER(0x04)
#define  DDRA_BIT0      BIT(0x04,0)
#define  DDRA_BIT1      BIT(0x04,1)
#define  DDRA_BIT2      BIT(0x04,2)
#define  DDRA_BIT3      BIT(0x04,3)
#define  DDRA_BIT4      BIT(0x04,4)
#define  DDRA_BIT5      BIT(0x04,5)
#define  DDRA_BIT6      BIT(0x04,6)
#define  DDRA_BIT7      BIT(0x04,7)


/* Port B Data Direction Register */
#define  DDRB          REGISTER(0x05)
#define  DDRB_BIT0     BIT(0x05,0)
#define  DDRB_BIT1     BIT(0x05,1)
#define  DDRB_BIT2     BIT(0x05,2)
#define  DDRB_BIT3     BIT(0x05,3)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
#define  DDRB_BIT4     BIT(0x05,4)
#define  DDRB_BIT5     BIT(0x05,5)
#define  DDRB_BIT6     BIT(0x05,6)
#define  DDRB_BIT7     BIT(0x05,7)


/* Port C Data Direction Register */
#define  DDRC          REGISTER(0x06)
#define  DDRC_BIT0     BIT(0x06,0)
#define  DDRC_BIT1     BIT(0x06,1)
#define  DDRC_BIT2     BIT(0x06,2)
#define  DDRC_BIT3     BIT(0x06,3)
#define  DDRC_BIT4     BIT(0x06,4)
#define  DDRC_BIT5     BIT(0x06,5)
#define  DDRC_BIT6     BIT(0x06,6)
#define  DDRC_BIT7     BIT(0x06,7)


/* Port D Data Direction Register */
#define  DDRD          REGISTER(0x07)
#define  DDRD_BIT0     BIT(0x07,0)
#define  DDRD_BIT1     BIT(0x07,1)
#define  DDRD_BIT2     BIT(0x07,2)
#define  DDRD_BIT3     BIT(0x07,3)
#define  DDRD_BIT4     BIT(0x07,4)
#define  DDRD_BIT5     BIT(0x07,5)
#define  DDRD_BIT6     BIT(0x07,6)
#define  DDRD_BIT7     BIT(0x07,7)


/* Port E Data Direction Register */
#define  DDRE          REGISTER(0x0c)
#define  DDRE_BIT0     BIT(0x0c,0)
#define  DDRE_BIT1     BIT(0x0c,1)


/* Port A Pull Up Enable Register */
#define  PTAPUE          REGISTER(0x0d)
#define  PTAPUE_BIT0     BIT(0x0d,0)
#define  PTAPUE_BIT1     BIT(0x0d,1)
#define  PTAPUE_BIT2     BIT(0x0d,2)
#define  PTAPUE_BIT3     BIT(0x0d,3)
#define  PTAPUE_BIT4     BIT(0x0d,4)
#define  PTAPUE_BIT5     BIT(0x0d,5)
#define  PTAPUE_BIT6     BIT(0x0d,6)
#define  PTAPUE_BIT7     BIT(0x0d,7)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/* Port C Pull Up Enable Register */
#define  PTCPUE          REGISTER(0x0e)
#define  PTCPUE_BIT0     BIT(0x0e,0)
#define  PTCPUE_BIT1     BIT(0x0e,1)
#define  PTCPUE_BIT2     BIT(0x0e,2)
#define  PTCPUE_BIT3     BIT(0x0e,3)
#define  PTCPUE_BIT4     BIT(0x0e,4)
#define  PTCPUE_BIT5     BIT(0x0e,5)
#define  PTCPUE_BIT6     BIT(0x0e,6)


/* Port D Pull Up Enable Register */
#define  PTDPUE          REGISTER(0x0f)
#define  PTDPUE_BIT0     BIT(0x0f,0)
#define  PTDPUE_BIT1     BIT(0x0f,1)
#define  PTDPUE_BIT2     BIT(0x0f,2)
#define  PTDPUE_BIT3     BIT(0x0f,3)
#define  PTDPUE_BIT4     BIT(0x0f,4)
#define  PTDPUE_BIT5     BIT(0x0f,5)
#define  PTDPUE_BIT6     BIT(0x0f,6)
#define  PTDPUE_BIT7     BIT(0x0f,7)


/*************************************************************************/
/* Serial Peripheral Interface Registers                                 */
/*************************************************************************/

/* SPI Control Register */
#define  SPCR           REGISTER(0x10)
#define  SPCR_SPTIE     BIT(0x10,0)
#define  SPCR_SPE       BIT(0x10,1)
#define  SPCR_SPWOM     BIT(0x10,2)
#define  SPCR_CPHA      BIT(0x10,3)
#define  SPCR_CPOL      BIT(0x10,4)
#define  SPCR_SPMSTR    BIT(0x10,5)
#define  SPCR_DMAS      BIT(0x10,6)
#define  SPCR_SPRIE     BIT(0x10,7)

/* SPI Status and Control Register */
#define  SPSCR          REGISTER(0x11)
#define  SPSCR_SPR0     BIT(0x11,0)
#define  SPSCR_SPR1     BIT(0x11,1)
#define  SPSCR_MODFEN   BIT(0x11,2)
#define  SPSCR_SPTE     BIT(0x11,3)
#define  SPSCR_MODF     BIT(0x11,4)
#define  SPSCR_OVRF     BIT(0x11,5)
#define  SPSCR_ERRIE    BIT(0x11,6)
#define  SPSCR_SPRF     BIT(0x11,7)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
/* SPI Data Register */
#define  SPDR           REGISTER(0x12)

/*************************************************************************/
/* Serial Communications Interface Registers                             */
/*************************************************************************/

/* SCI Control Register 1 */
#define  SCC1           REGISTER(0x13)
#define  SCC1_PTY       BIT(0x13,0)
#define  SCC1_PEN       BIT(0x13,1)
#define  SCC1_ILTY      BIT(0x13,2)
#define  SCC1_WAKE      BIT(0x13,3)
#define  SCC1_M         BIT(0x13,4)
#define  SCC1_TXINV     BIT(0x13,5)
#define  SCC1_ENSCI     BIT(0x13,6)
#define  SCC1_LOOPS     BIT(0x13,7)

/* SCI Control Register 2 */
#define  SCC2           REGISTER(0x14)
#define  SCC2_SBK       BIT(0x14,0)
#define  SCC2_RWU       BIT(0x14,1)
#define  SCC2_RE        BIT(0x14,2)
#define  SCC2_TE        BIT(0x14,3)
#define  SCC2_ILIE      BIT(0x14,4)
#define  SCC2_SCRIE     BIT(0x14,5)
#define  SCC2_TCIE      BIT(0x14,6)
#define  SCC2_SCTIE     BIT(0x14,7)

/* SCI Control Register 3 */
#define  SCC3           REGISTER(0x15)
#define  SCC3_PEIE      BIT(0x15,0)
#define  SCC3_FEIE      BIT(0x15,1)
#define  SCC3_NEIE      BIT(0x15,2)
#define  SCC3_ORIE      BIT(0x15,3)
#define  SCC3_DMATE     BIT(0x15,4)
#define  SCC3_DMARE     BIT(0x15,5)
#define  SCC3_T8        BIT(0x15,6)
#define  SCC3_R8        BIT(0x15,7)

/* SCI Status Register 1 */
#define  SCS1           REGISTER(0x16)
#define  SCS1_PE        BIT(0x16,0)
#define  SCS1_FE        BIT(0x16,1)
#define  SCS1_NF        BIT(0x16,2)
#define  SCS1_OR        BIT(0x16,3)
#define  SCS1_IDLE      BIT(0x16,4)
#define  SCS1_SCRF      BIT(0x16,5)
#define  SCS1_TC        BIT(0x16,6)
#define  SCS1_SCTE      BIT(0x16,7)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/* SCI Status Register 2 */
#define  SCS2           REGISTER(0x17)
#define  SCS2_RPF       BIT(0x17,0)
#define  SCS2_BKF       BIT(0x17,1)

/* SCI Data Register */
/* bit manipulation not recomended */
#define  SCDR           REGISTER(0x18)

/* SCI Baud Rate Register */
#define  SCBR           REGISTER(0x19)
#define  SCBR_SCR0      BIT(0x19,0)
#define  SCBR_SCR1      BIT(0x19,1)
#define  SCBR_SCR2      BIT(0x19,2)
#define  SCBR_SCP0      BIT(0x19,4)
#define  SCBR_SCP1      BIT(0x19,5)

/*************************************************************************/
/* Keyboard Registers                                                    */
/*************************************************************************/

/* Keyboard Status and Control Register */
#define  INTKBSCR          REGISTER(0x1a)
#define  INTKBSCR_MODEK    BIT(0x1a,0)
#define  INTKBSCR_IMASKK   BIT(0x1a,1)
#define  INTKBSCR_ACKK     BIT(0x1a,2)
#define  INTKBSCR_KEYF     BIT(0x1a,3)

/* Keyboard Interrupt Enable Register */
#define  INTKBIER           REGISTER(0x1b)
#define  INTKBIER_KBIE0     BIT(0x1b,0)
#define  INTKBIER_KBIE1     BIT(0x1b,1)
#define  INTKBIER_KBIE2     BIT(0x1b,2)
#define  INTKBIRE_KBIE3     BIT(0x1b,3)
#define  INTKBIER_KBIE4     BIT(0x1b,4)
#define  INTKBIER_KBIE5     BIT(0x1b,5)
#define  INTKBIER_KBIE6     BIT(0x1b,6)
#define  INTKBIER_KBIE7     BIT(0x1b,7)


/*************************************************************************/
/* Time Base Control Register                                            */
/*************************************************************************/

#define  TBCR           REGISTER(0x1c)
#define  TBCR_TBON      BIT(0x1c,1)
#define  TBCR_TBIE      BIT(0x1c,2)
#define  TBCR_TACK      BIT(0x1c,3)
#define  TBCR_TBR0      BIT(0x1c,4)
#define  TBCR_TBR1      BIT(0x1c,5)
#define  TBCR_TBR2      BIT(0x1c,6)
#define  TBCR_TBIF      BIT(0x1c,7)
```


For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Source Code

```
/*************************************************************************/
/* IRQ Status and Control Register                                       */
/*************************************************************************/

#define  ISCR           REGISTER(0x1d)
#define  ISCR_MODE1     BIT(0x1d,0)
#define  ISCR_IMASK1    BIT(0x1d,1)
#define  ISCR_ACK1      BIT(0x1d,2)
#define  ISCR_IRQF1     BIT(0x1d,3)

/*************************************************************************/
/* Configuration Write-Once Registers                                    */
/*                                                                       */
/* note: bit fields or bit manipulation is not permitted on write once reg  */
/*                                                                       */
/*************************************************************************/

#define  CONFIG2    REGISTER(0x1e)
#define  CONFIG1    REGISTER(0x1F)

/*************************************************************************/
/* Timer Registers #1                                                    */
/*************************************************************************/

/* Timer Status and Control Register */
#define  T1SC            REGISTER(0x20)
#define  T1SC_PS0        BIT(0x20,0)
#define  T1SC_PS1        BIT(0x20,1)
#define  T1SC_PS2        BIT(0x20,2)
#define  T1SC_TRST       BIT(0x20,4)
#define  T1SC_TSTOP      BIT(0x20,5)
#define  T1SC_TOIE       BIT(0x20,6)
#define  T1SC_TOF        BIT(0x20,7)

/* Timer Counter Register */
#define  T1CNT  DBLREG(0x21)

#define  T1CNTH  REGISTER(0x21)
#define  T1CNTL  REGISTER(0x22)

/* Timer Modulo Register  */

#define  T1MOD  DBLREG(0x23)

#define  T1MODH  REGISTER(0x23)
#define  T1MODL  REGISTER(0x24)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/* Timer Status and Control Register Channel 0 */
#define  T1SC0   REGISTER(0x25)
#define  T1SC0_CH0MAX  BIT(0x25,0)
#define  T1SC0_TOV0    BIT(0x25,1)
#define  T1SC0_ELS0A   BIT(0x25,2)
#define  T1SC0_ELS0B   BIT(0x25,3)
#define  T1SC0_MS0A    BIT(0x25,4)
#define  T1SC0_MS0B    BIT(0x25,5)
#define  T1SC0_CH0IE   BIT(0x25,6)
#define  T1SC0_CH0F    BIT(0x25,7)

/* Timer Channel 0 Register */
#define  T1CH0   DBLREG(0x26)

#define  T1CH0H  REGISTER(0x26)
#define  T1CH0L  REGISTER(0x27)

/* Timer Status and Control Register Channel 1 */
#define  T1SC1   REGISTER(0x28)
#define  T1SC1_CH1MAX  BIT(0x28,0)
#define  T1SC1_TOV1    BIT(0x28,1)
#define  T1SC1_ELS1A   BIT(0x28,2)
#define  T1SC1_ELS1B   BIT(0x28,3)
#define  T1SC1_MS1A    BIT(0x28,4)
#define  T1SC1_CH1IE   BIT(0x28,6)
#define  T1SC1_CH1F    BIT(0x28,7)

/* Timer Channel 1 Register */
#define  T1CH1   DBLREG(0x29)

#define  T1CH1H  REGISTER(0x29)
#define  T1CH1L  REGISTER(0x2a)


/*****************************************************************************/
/* Timer Registers #2                                                        */
/*****************************************************************************/

/* Timer Status and Control Register */
#define  T2SC            REGISTER(0x2b)
#define  T2SC_PS0        BIT(0x2b,0)
#define  T2SC_PS1        BIT(0x2b,1)
#define  T2SC_PS2        BIT(0x2b,2)
#define  T2SC_TRST       BIT(0x2b,4)
#define  T2SC_TSTOP      BIT(0x2b,5)
#define  T2SC_TOIE       BIT(0x2b,6)
#define  T2SC_TOF        BIT(0x2b,7)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
/* Timer Counter Register */
#define  T2CNT  DBLREG(0x2c)

#define  T2CNTH  REGISTER(0x2c)
#define  T2CNTL  REGISTER(0x2d)

/* Timer Modulo Register  */

#define  T2MOD  DBLREG(0x2e)

#define  T2MODH  REGISTER(0x2e)
#define  T2MODL  REGISTER(0x2f)

/* Timer Status and Control Register Channel 0 */
#define  T2SC0   REGISTER(0x30)
#define  T2SC0_CH0MAX  BIT(0x30,0)
#define  T2SC0_TOV0    BIT(0x30,1)
#define  T2SC0_ELS0A   BIT(0x30,2)
#define  T2SC0_ELS0B   BIT(0x30,3)
#define  T2SC0_MS0A    BIT(0x30,4)
#define  T2SC0_MS0B    BIT(0x30,5)
#define  T2SC0_CH0IE   BIT(0x30,6)
#define  T2SC0_CH0F    BIT(0x30,7)

/* Timer Channel 0 Register */
#define  T2CH0   DBLREG(0x31)

#define  T2CH0H  REGISTER(0x31)
#define  T2CH0L  REGISTER(0x32)

/* Timer Status and Control Register Channel 1 */
#define  T2SC1   REGISTER(0x33)
#define  T2SC1_CH1MAX  BIT(0x33,0)
#define  T2SC1_TOV1    BIT(0x33,1)
#define  T2SC1_ELS1A   BIT(0x33,2)
#define  T2SC1_ELS1B   BIT(0x33,3)
#define  T2SC1_MS1A    BIT(0x33,4)
#define  T2SC1_CH1IE   BIT(0x33,6)
#define  T2SC1_CH1F    BIT(0x33,7)

/* Timer Channel 1 Register */
#define  T2CH1   DBLREG(0x34)

#define  T2CH1H  REGISTER(0x34)
#define  T2CH1L  REGISTER(0x35)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/*****************************************************************************/
/* Phase Locked Loop Module Registers                                        */
/*****************************************************************************/

/* PLL Control Register */
#define  PCTL          REGISTER(0x36)
#define  PCTL_VPR0     BIT(0x36,0)
#define  PCTL_VPR1     BIT(0x36,1)
#define  PCTL_PRE0     BIT(0x36,2)
#define  PCTL_PRE1     BIT(0x36,3)
#define  PCTL_BCS      BIT(0x36,4)
#define  PCTL_PLLON    BIT(0x36,5)
#define  PCTL_PLLF     BIT(0x36,6)
#define  PCTL_PLLIE    BIT(0x36,7)

/* PLL Bandwidth Register */
#define  PBWC          REGISTER(0x37)
#define  PBWC_ACQ      BIT(0x37,5)
#define  PBWC_LOCK     BIT(0x37,6)
#define  PBWC_AUTO     BIT(0x37,7)

/* PLL Multiplier High Register */
#define  PMSH          REGISTER(0x38)

/* PLL Multiplier Low Register */
#define  PMSL          REGISTER(0x39)

/* PLL VCO Select Range Register */
#define  PMRS          REGISTER(0x3a)

/* PLL Reference Divider Select Register */
#define  PMDS          REGISTER(0x3b)


/*****************************************************************************/
/* Analogue To Digital Converter Registers                                   */
/*****************************************************************************/

/* A/D Status and Control Register */
#define  ADSCR          REGISTER(0x3c)
#define  ADSCR_ADCH0    BIT(0x3c,0)
#define  ADSCR_ADCH1    BIT(0x3c,1)
#define  ADSCR_ADCH2    BIT(0x3c,2)
#define  ADSCR_ADCH3    BIT(0x3c,3)
#define  ADSCR_ADCH4    BIT(0x3c,4)
#define  ADSCR_ADCO     BIT(0x3c,5)
#define  ADSCR_AIEN     BIT(0x3c,6)
#define  ADSCR_COCO     BIT(0x3c,7)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
/* A/D-Data Register */
#define  ADR            REGISTER(0x3d)

/* A/D Input Clock Register */
#define  ADCLK          REGISTER(0x3e)
#define  ADCLK_ADICLK   BIT(0x3e,4)
#define  ADCLK_ADIV0    BIT(0x3e,5)
#define  ADCLK_ADIV1    BIT(0x3e,6)
#define  ADCLK_ADIV2    BIT(0x3e,7)



/*****************************************************************************/
/* System Integration Module Registers                                       */
/*****************************************************************************/

/* Break Status Register */
#define  SBSR           REGISTER(0xFE00)
#define  SBSR_SBSW      BIT(0xFE00,1)

/* SIM Reset Status Register */
#define  SRSR          REGISTER(0xFE01)
#define  SRSR_LVI      BIT(0xFE01,1)
#define  SRSR_MODRST   BIT(0xFE01,2)
#define  SRSR_ILAD     BIT(0xFE01,3)
#define  SRSR_ILOP     BIT(0xFE01,4)
#define  SRSR_COP      BIT(0xFE01,5)
#define  SRSR_PIN      BIT(0xFE01,6)
#define  SRSR_POR      BIT(0xFE01,7)

/* SIM Upper Byte Address Register */
#define  SUBAR          REGISTER(0xFE02)

/* Break Flag Control Register */
#define  BFCR          REGISTER(0xFE03)
#define  BFCR_BCFE     BIT(0xFE03,7)

/* Break Address Registers */
#define  BRKH          REGISTER(0xFE09)
#define  BRKL          REGISTER(0xFE0a)

/* Break Status & Control Register */
#define  BRKSCR        REGISTER(0xFE0b)
#define  BRKSCR_BRKA   BIT(0xFE0b,6)
#define  BRKSCR_BRKE   BIT(0xFE0b,7)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
/*****************************************************************************/
/* Interrupt Registers                                                       */
/*****************************************************************************/

/* Interrupt Status Register 1 */
#define  INT1         REGISTER(0xFE04)
#define  INT1_IF1     BIT(0xFE04,2)
#define  INT1_IF2     BIT(0xFE04,3)
#define  INT1_IF3     BIT(0xFE04,4)
#define  INT1_IF4     BIT(0xFE04,5)
#define  INT1_IF5     BIT(0xFE04,6)
#define  INT1_IF6     BIT(0xFE04,7)

/* Interrupt Status Register 2 */
#define  INT2         REGISTER(0xFE05)
#define  INT2_IF7     BIT(0xFE05,0)
#define  INT2_IF8     BIT(0xFE05,1)
#define  INT2_IF9     BIT(0xFE05,2)
#define  INT2_IF10    BIT(0xFE05,3)
#define  INT2_IF11    BIT(0xFE05,4)
#define  INT2_IF12    BIT(0xFE05,5)
#define  INT2_IF13    BIT(0xFE05,6)
#define  INT2_IF14    BIT(0xFE05,7)

/* Interrupt Status Register 3 */
#define  INT3         REGISTER(0xFE06)
#define  INT3_IF15    BIT(0xFE06,0)
#define  INT3_IF16    BIT(0xFE06,1)


/*****************************************************************************/
/* Flash Registers                                                           */
/*****************************************************************************/

/* Flash Control Register 1 */
#define  FLCR         REGISTER(0xFE08)
#define  FLCR_PGM     BIT(0xFE08,0)
#define  FLCR_ERASE   BIT(0xFE08,1)
#define  FLCR_MASS    BIT(0xFE08,2)
#define  FLCR_HVEN    BIT(0xFE08,3)

#define  M_FLCR_PGM     BIT0
#define  M_FLCR_ERASE   BIT1
#define  M_FLCR_MARGIN  BIT2
#define  M_FLCR_HVEN    BIT3

/* Flash Block Protect Register 1 */
#define  FLBPR        REGISTER(0xFF7E)
#define  FLBPR_BPR0   BIT(0xFF7E,0)
#define  FLBPR_BPR1   BIT(0xFF7E,1)
#define  FLBPR_BPR2   BIT(0xFF7E,2)
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Source Code

```
#define  FLBPR_BPR3   BIT(0xFF7E,3)
#define  FLBPR_BPR4   BIT(0xFF7E,4)
#define  FLBPR_BPR5   BIT(0xFF7E,5)
#define  FLBPR_BPR6   BIT(0xFF7E,6)
#define  FLBPR_BPR7   BIT(0xFF7E,7)


/*****************************************************************************/
/* COP Control Registers                                                     */
/*****************************************************************************/

#define  COPCTL REGISTER(0xFFFF)


/*****************************************************************************/
/* Short Aliases                                                             */
/*****************************************************************************/

/* Port A short aliases */
#define PTA0    PTA_BIT0
#define PTA1    PTA_BIT1
#define PTA2    PTA_BIT2
#define PTA3    PTA_BIT3
#define PTA4    PTA_BIT4
#define PTA5    PTA_BIT5
#define PTA6    PTA_BIT6
#define PTA7    PTA_BIT7

/* Port B short aliases */
#define PTB0    PTB_BIT0
#define PTB1    PTB_BIT1
#define PTB2    PTB_BIT2
#define PTB3    PTB_BIT3
#define PTB4    PTB_BIT4
#define PTB5    PTB_BIT5
#define PTB6    PTB_BIT6
#define PTB7    PTB_BIT7

/* Port C short aliases */
#define PTC0    PTC_BIT0
#define PTC1    PTC_BIT1
#define PTC2    PTC_BIT2
#define PTC3    PTC_BIT3
#define PTC4    PTC_BIT4
#define PTC5    PTC_BIT5
#define PTC6    PTC_BIT6

/* Port D short aliases */
#define PTD0    PTD_BIT0
#define PTD1    PTD_BIT1
#define PTD2    PTD_BIT2
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

```
#define PTD3    PTD_BIT3
#define PTD4    PTD_BIT4
#define PTD5    PTD_BIT5
#define PTD6    PTD_BIT6
#define PTD7    PTD_BIT7

/* Port E short aliases */
#define PTE0    PTE_BIT0
#define PTE1    PTE_BIT1

/*****************************************************************************/
/* Interrupt Vectors                                                         */
/*****************************************************************************/

#define IV_TBM        17    /* Time Base Module                */
#define IV_ADC        16    /* Analogue To Digital Converter   */
#define IV_KBRD       15    /* Keyboard                        */
#define IV_SCI_TX     14    /* Serial Communication Transmit   */
#define IV_SCI_RX     13    /* Serial Communication Receive    */
#define IV_SCI_ERROR  12    /* Serial Communication Error      */
#define IV_SPI_TX     11    /* SPI Transmit                    */
#define IV_SPI_TRX    10    /* SPI Receive                     */
#define IV_T2OVF      9     /* Timer B Channel 1               */
#define IV_T2CH1      8     /* Timer B Channel 0               */
#define IV_T2CHO      7     /* Modulo B Timer                  */
#define IV_T1OVF      6     /* Timer A Channel 1               */
#define IV_T1CH1      5     /* Timer A Channel 0               */
#define IV_T1CHO      4     /* Modulo A Timer                  */
#define IV_PLL        3     /* Phase Locked Loop               */
#define IV_IRQ1       2     /* Interrupt Request1              */
#define IV_SWI        1     /* Software Interrupt              */
#define IV_RESET      0     /* Reset                           */

#endif
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

7.12 hc08gr8.prm

```
NAMES END

SECTIONS
    Z_RAM  = READ_WRITE  0x0040 TO 0x00FF;
    RAM    = READ_WRITE  0x0100 TO 0x01BF;
    ROM    = READ_ONLY   0xE000 TO 0xFBFF;
END

PLACEMENT
    DEFAULT_ROM, ROM_VAR, STRINGS, ROM_CONST INTO  ROM;
    DEFAULT_RAM                              INTO  RAM;
    _DATA_ZEROPAGE, MY_ZEROPAGE              INTO  Z_RAM;
END

STACKSIZE 0x30

VECTOR 0    _Startup
VECTOR 1    nullHand0
VECTOR 2    phs_IRQ_ISR    /* Irq1_Int          */
VECTOR 3    nullHand0      /* PLL_Int           */
VECTOR 4    phs_RxEdgeISR     /* Tim1Ch0_Int       */
VECTOR 5    app_TriacTmrISR   /* Tim1Ch1_Int       */
VECTOR 6    nullHand0         /* Tim1Ovrfl_Int     */
VECTOR 7    phs_RxBitISR      /* Tim2Ch0_Int       */
VECTOR 8    nullHand0         /* not used on GR8   */
VECTOR 9    phs_TxBitISR      /* Tim2Ovrfl_Int     */
VECTOR 10   nullHand0         /* Spi_Rec_Full      */
VECTOR 11   nullHand0      /* SpiTx_Int         */
VECTOR 12   nullHand0      /* SciErr_Int        */
VECTOR 13   nullHand0         /* SciRx_Int         */
VECTOR 14   nullHand0         /* SciTx_Int         */
VECTOR 15   phs_CDdetectISR   /* Keyboard_Int      */
VECTOR 16   nullHand0      /* ADC_Int           */
VECTOR 17   dll_TBModuleISR   /* TBM_Int           */
```

Freescale Semiconductor, Inc.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

Information in this document is provided solely to enable system and software implementers to use Motorola products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

MOTOROLA

Motorola and the Stylized M Logo are registered in the U.S. Patent and Trademark Office. digital dna is a trademark of Motorola, Inc. All other product or service names are the property of their respective owners. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

© Motorola, Inc. 2003

DRM009/D
Rev. 0
2/2003

Freescale Semiconductor, Inc.