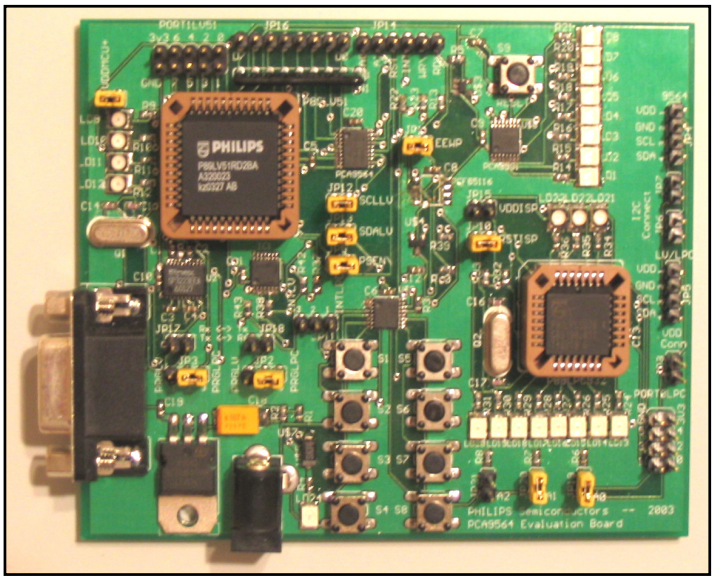
APPLICATION NOTE



Abstract

PCA9564 evaluation board description, features and operation modes are discussed. Source code in C language, containing communication routines between an 80C51-core microcontroller and the PCA9564 is provided.

AN10149

PCA9564 Evaluation Board

Jean-Marc Irazabal — PCA Technical Marketing Manager
Paul Boogaards — Sr. Field Application Engineer
Bauke Siderius — Application Engineer

2004 Aug 19

Philips Semiconductors

PHILIPS

TABLE OF CONTENTS

OVERVIEW

Description

The PCA9564 Evaluation Board demonstrates the Philips PCA9564 I^2C-bus controller's ability to interface between a master (connected to its parallel bus and its control signals) and any master and slave devices connected to its I^2C-bus.
The evaluation board is populated with the following devices and functions:

- **Philips P89LV51RD2** microcontroller connected to the PCA9564 8-bit parallel port and control signals. It is used as the master controlling the other devices on the board with the embedded firmware. It can also be used as a slave device with an appropriate program loaded.
- **Philips PCA9564** I^2C-bus controller interfacing between the P89LV51RD2 and the I^2C-bus.
- **Philips PCA9531** I^2C 8-bit LED dimmer used as an I^2C target slave device for the P89LV51RD2/PCA9564.
- **Philips P89LPC932** microcontroller connected to the I^2C-bus. It can act as either a target slave device with the default P89LV51RD2 firmware programs or as a master connected to the I^2C-bus through some stored user definable routines.
- **Philips PCF85116** 16 kbits (2KB) I^2C EEPROM used to store information that can be used by the evaluation board firmware.
- **Philips PCA9554A** I^2C 8-bit GPIO acting as interface / keyboard between the user and the P89LV51RD2
- **Sipex SP3223** RS-232 transceiver allows the P89LV51RD2 and the P89LPC932 devices to be in-system programmed through a personal computer's serial port.

An external 9 V DC power supply is used to provide power to the 3.3 V on-board voltage regulator. The P89LPC932 and P89LV51 are both limited to a 3.3 V supply voltage.

The evaluation board can be used in different ways:

1. Stand-alone mode: 4 default firmware programs are stored in the P89LV51RD2 (master) and the P89LPC932 (slave). No external hardware or software is required. The firmware allows the user to execute some applications where data and control traffic is automatically generated in both directions between the P89LV51RD2 and the PCA9564 on one side and the PCA9564 and the I^2C devices on the other side (PCA9531, PCF85116, P89LPC932 and PCA9554A). The user, through an 8-switch interface, can control the routines and the execution of the commands. The embedded firmware provides master mode examples (transmitter and receiver). Code is written in C language and can be used with any 80C51-type microcontroller. The embedded firmware can be downloaded from the www.standardproducts.philips.com website which the user can modify as required.
2. Program the microcontroller(s) with compiled files ("Hex" files) through the ISP (In-System Programming) interface. This mode allows a user to program the microcontroller(s) with additional applications and programs. Code programming is not required and the "Hex" file(s) can be loaded to the microcontroller(s) by using Flash Magic, Windows based free software from the Embedded Systems Academy, sponsored by Philips Semiconductors (http://www.esacademy.com/software/flashmagic/). "Hex" files can be downloaded from the www.standardproducts.philips.com website. "Hex" files can be the manufacturing default embedded program (explained above) or any evaluation/demo program that will be developed for this specific board.
3. Use the full flow using 8051 software development tools: C code generation or Assembler code generation, program debugging, compilation and program loading the targeted microcontroller to develop specific applications using the PCA9564 evaluation board and optional I^2C devices daughter cards. Free evaluation software from American Raisonance allowing up to 4 kbits of code can be used.
4. Use any emulator, microcontroller, microprocessor or DSP instead of the Philips P89LV51RD2. To do that, the new master needs to be connected to the 8-bit parallel port and control signals headers and the P89LV51RD2 needs to be removed from its socket.

For more information about program files and software that is required, refer to the paragraphs "Download software, programs and documentation" and "PCA9564 evaluation board web page".

Ordering information

The complete PCA9564 evaluation board Kit consists of the:

- PCA9564 evaluation board
- 9 V DC power supply
- DB-9 connector

Kit can be obtained through your local Philips Semiconductors Sales organization. It can also be obtained via email at i2c.support@philips.com.

TECHNICAL INFORMATION – HARDWARE

Block diagram

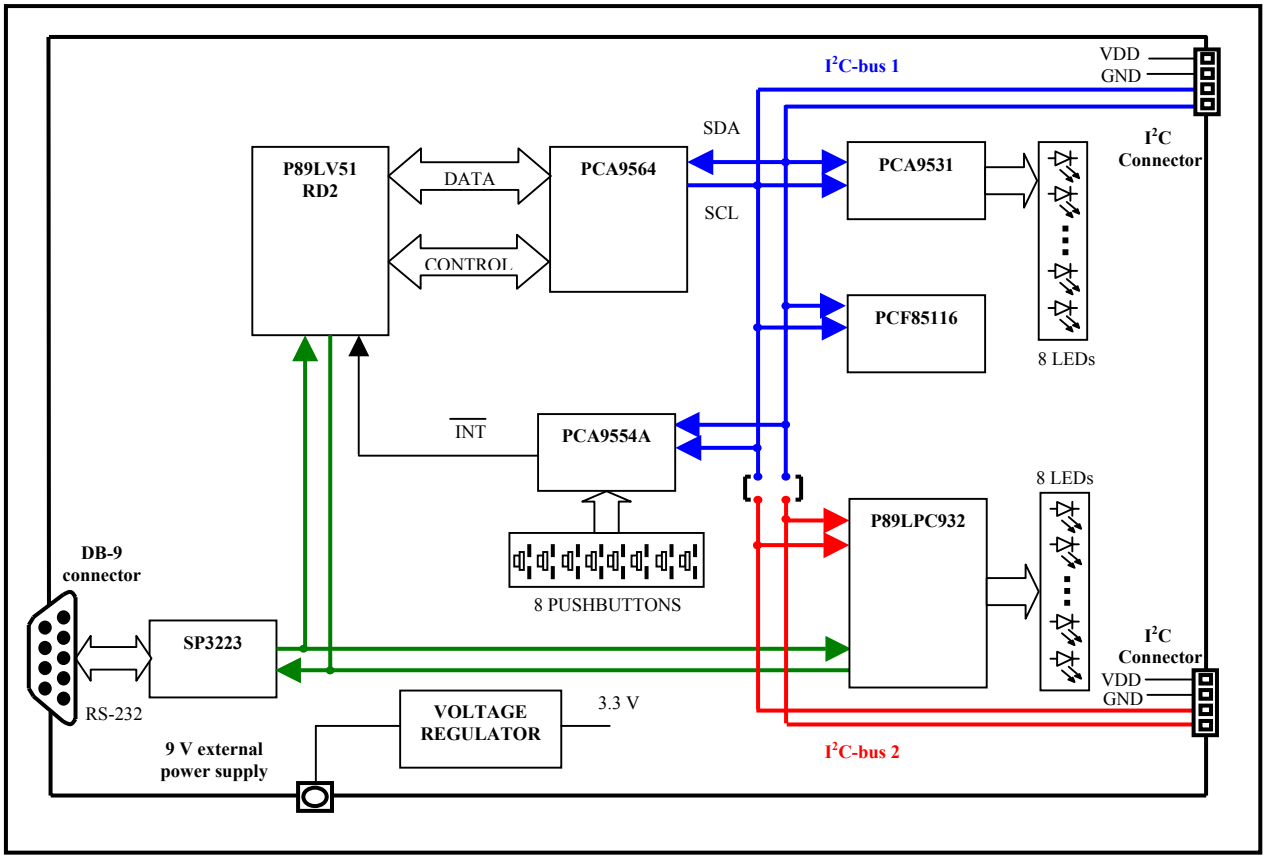


Figure 1. Evaluation board block diagram

I²C device addresses

Device type	Description	I²C Address (Hexadecimal)
P89LV51RD2 / PCA9564	Microcontroller / I²C-bus controller	User definable when microcontroller used as slave
P89LPC932	Microcontroller	User definable when microcontroller used as slave 0xE0 to 0xE8 with the embedded programs
PCA9531	8-bit I²C LED Dimmer	0xC8
PCF85116	16kbits I²C EEPROM	0xA0 to 0xA8 (function of the addressed memory)
PCA9554A	8-bit I²C GPIO	0x7E

Table 1. I²C device addresses

Schematic

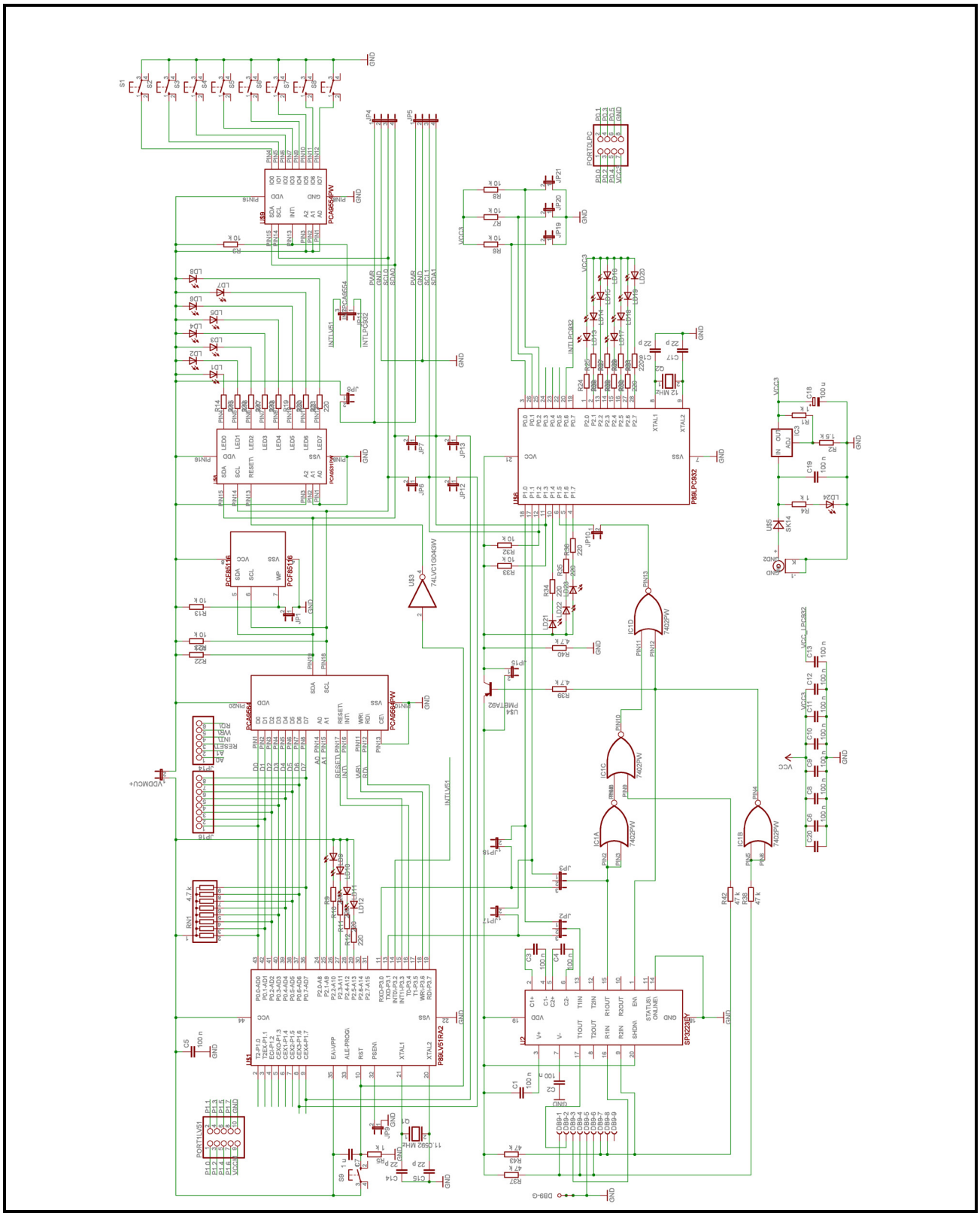


Figure 2. PCA9564 Evaluation Board Schematic

PCA9564 Evaluation Board Top view

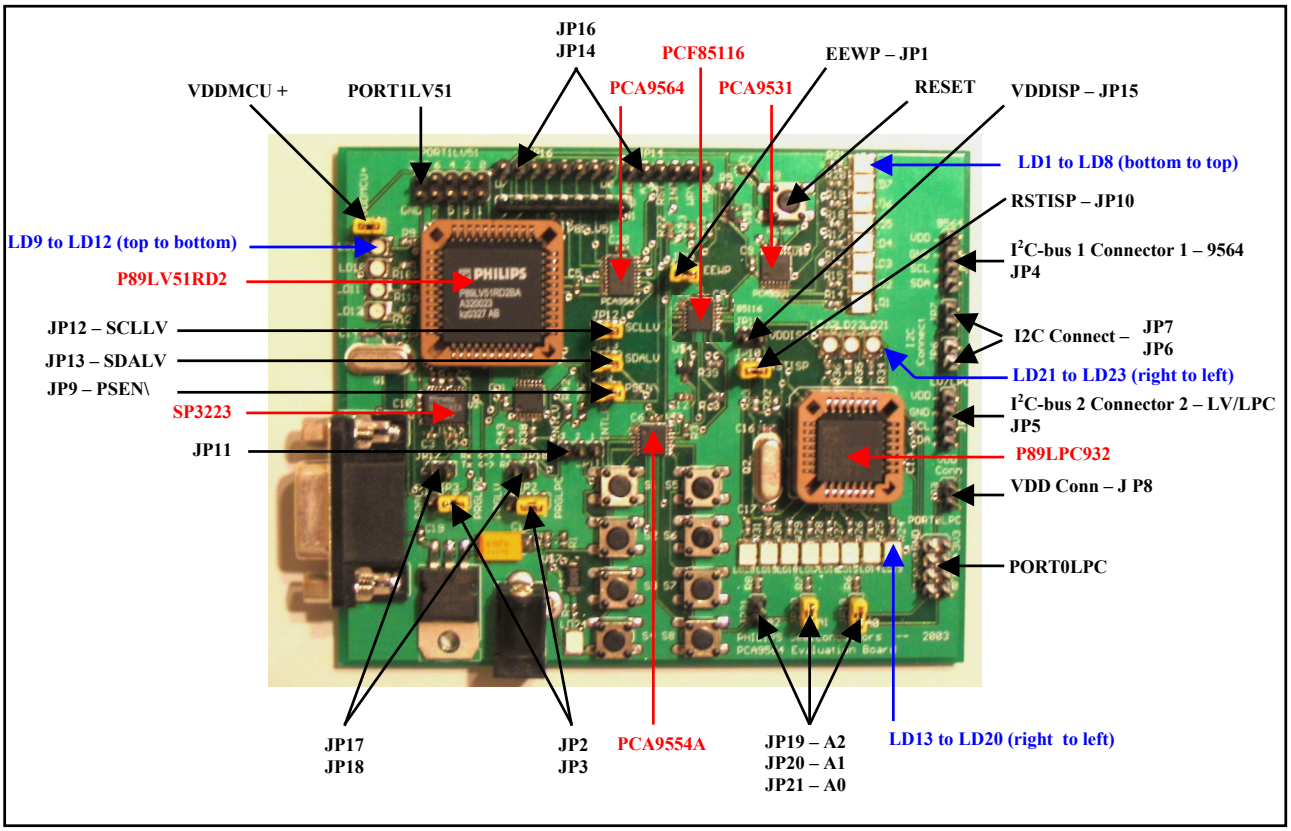


Figure 3. PCA9564 Evaluation Board Top View

Jumpers and Headers

Label	Purpose	Jumper position	Description
JP1 **(EEWP)**	PCF85116 Write Protect	Open	WP pin connected to V_{DD} – Write not permitted
		Closed	WP pin connect to GND – Write permitted
JP2	Selection of the microcontroller to be programmed through ISP (TxD)	Open	No ISP programming can be performed
		Closed between 1 and 2 (**PRGLPC**)	ISP programming of P89LPC932 can be performed TxD pin of P89LPC932 connected to T_1IN of SP3223
		Closed between 2 and 3 (**PRGLV**)	ISP programming of P89LV51RD2 can be performed TxD pin of P89LV51RD2 connected to T_1IN of SP3223
JP3	Selection of the microcontroller to be programmed through ISP (RxD)	Open	No ISP programming can be performed
		Closed between 1 and 2 (**PRGLPC**)	ISP programming of P89LPC932 can be performed RxD pin of P89LPC932 connected to R_1OUT of SP3223
		Closed between 2 and 3 (**PRGLV**)	ISP programming of P89LV51RD2 can be performed TxD pin of P89LV51RD2 connected to R_1OUT of SP3223
JP4 **(9564)**	I^2C-bus connector 1		I^2C-bus 1 – Bus connected to the PCA9564, PCA9531, PCF85116 and PCA9554A **Note:** I^2C-bus 1 and I^2C-bus 2 can be connected together through jumpers JP6 and JP7
JP5 **(LV/LPC)**	I^2C-bus connector 2		I^2C-bus 2 – Bus connected to the P89LPC932. It is also connected to the I^2C-bus of a P89C51Rx+/Rx2/66x with I^2C-bus (SCL = P1.6, SDA = P1.7) when JP12 and JP13 closed **Note:** I^2C-bus 1 and I^2C-bus 2 can be connected together through jumpers JP6 and JP7

JP6 **(I2C Connect)**	Connect I^2C-bus 1 and I^2C-bus 2	Open	SCL I^2C-bus 1 and SCL I^2C-bus 2 are not connected together
		Closed	SCL I^2C-bus 1 and SCL I^2C-bus 2 are connected together
JP7 **(I2C Connect)**	Connect I^2C-bus 1 and I^2C-bus 2	Open	SDA I^2C-bus 1 and SDA I^2C-bus 2 are not connected together
		Closed	SDA I^2C-bus 1 and SDAI^2C-bus 2 are connected together
JP8 **(VDD Conn)**	Power supply for the I^2C-bus connectors	Open	V_{DD} pin of connectors not connected to the internal 3.3 V power supply
		Closed	V_{DD} pin of connectors connected to the internal 3.3 V power supply
JP9 **(PSEN\)**	89C51Rx+/Rx2/66x ISP mode (Not applicable to P89LV51RD2, only to 5 V devices)	Open	ISP mode not entered
		Closed	ISP mode entered **Note:** More information can be found on the Philips Application Notes AN461: "In-circuit and In-application programming of the 89C51Rx+/Rx2/66x microcontrollers"
JP10 **(RSTISP)**	P89LPC932 ISP mode	Open	Normal mode
		Closed	P89LPC932 ISP mode
JP11	PCA9554A Interrupt output monitoring	Open	PCA9554A $\overline{INT}$ pin not monitored
		Closed between 1 and 2 (**INTLPC**)	PCA9554A $\overline{INT}$ pin can be monitored by P89LPC932
		Closed between 2 and 3 (**INTLV**)	PCA9554A $\overline{INT}$ pin can be monitored by P89LV51RD2
JP12 **(SCLLV)**	P89x51 with I^2C-bus connection to I^2C-bus 2	Open	P89C51Rx+/Rx2/66x with I^2C-bus (SCL = P1.6) not connected to SCL I^2C-bus 2
		Closed	P89C51Rx+/Rx2/66x with I^2C-bus (SCL = P1.6) connected to SCL I^2C-bus 2
JP13 **(SDALV)**	P89x51 with I^2C-bus connection to I^2C-bus 2	Open	P89C51Rx+/Rx2/66x with I^2C-bus (SDA = P1.7) not connected to SDA I^2C-bus 2
		Closed	P89C51Rx+/Rx2/66x with I^2C-bus (SDA = P1.7) connected to SDA I^2C-bus 2
JP14	PCA9564 control signals		Probing of PCA9564 control signals
JP15 **(VDDISP)**		Open	P89LPC932 ISP mode
		Closed	Normal mode
JP16	PCA9564 parallel bus		Probing of PCA9564 8-bit parallel bus
JP17 **(Tx ↔Rx)**	Connection TxD P89LV51RD2 to RxD P89LPC932	Open	Pins not connected together
		Closed	Pins connected together **Note:** JP2 and JP3 must be open when JP17 is closed
JP18 **(Rx ↔Tx)**	Connection RxD P89LV51RD2 to TxD P89LPC932	Open	Pins not connected together
		Closed	Pins connected together **Note:** JP2 and JP3 must be open when JP18 is closed
JP19 **(A0)**	P89LPC932 I^2C slave address input 0	Open	Address Input 0 connected to V_{DD} – A0 = 1
		Closed	Address Input 0 connected to GND – A0 = 0
JP20 **(A1)**	P89LPC932 I^2C slave address input 1	Open	Address Input 1 connected to V_{DD} – A1 = 1
		Closed	Address Input 1 connected to GND – A1 = 0
JP21 **(A2)**	P89LPC932 I^2C slave address input 2	Open	Address Input 2 connected to V_{DD} – A2 = 1
		Closed	Address Input 2 connected to GND – A2 = 0
VDDMCU+	P89xx51 Power supply selection	Open	External power supply can be applied to the P89xx51 microcontroller (Voltage applied to pin VDDMCU+, on the left side of the jumper)
		Closed	Internal regulated 3.3 V power supply applied to the P89xx51 microcontroller
PORT1LV51	Port 1 P89LV51		General purpose 8-bit Input/Output port (Port 1 P89LV51RD2)
PORT0LPC	Port 0 P89LPC932		General purpose 8-bit Input/Output port (Port 0 P89LPC932)

Table 2. PCA9564 Evaluation Jumpers and Headers

Pushbuttons – User interface and Reset

- Pushbuttons S1 to S8:
  They are connected to the 8 inputs of the PCA9554A, I^2C General Purpose Input Output device and can be used as an interface between the user and the microcontroller(s) to perform actions such as program selection, user definable events …
  The microcontroller(s) can either:
  - **Poll the PCA9554A** in order to read the input register and the state of the switches.
    Reading of the input port is performed by:
    1. Sending the PCA9554A I^2C address with a Write command followed by 0x00 (Input register pointer).
    2. A Re-Start Command followed by the PCA9554A I^2C address with a Read command.
    3. Reading the input port register byte from the PCA9554A.
  - **Monitor the PCA9554A Interrupt output pin** in order to detect change(s) in the switches.
    When one or more input change states:
    1. The PCA9554A Interrupt output will go LOW, thus indicating to the microcontroller that a switch has been pressed and the Interrupt service routine needs to be initiated.
    2. The microcontroller can then perform the same reading sequence as explained above in order to determine which input changes state. Reading the PCA9554A will automatically clear its interrupt.

  Pushbuttons can be used in 2 different modes with the embedded programs:
  - **Single shot mode:** a single push then release is detected. The action associated with the pushbutton is executed once.
    1. An Interrupt is detected by the master (P89LV51RD2) when a pushbutton is pressed.
    2. P89LV51RD2 initiates a read of the PCA9554A input register (first snapshot).
    3. P89LV51RD2 initiates a second reading of the PCA9554A input register (second snapshot) about 750 ms later.

    If the second reading indicates a pushbutton idle condition, then the action read the first time is performed once.
  - **Permanent push mode:** the user keeps the pushbutton pushed and the master executes the associated command until the pushbutton is released again.
    1. An Interrupt is detected by the master (P89LV51RD2) when a pushbutton is pressed
    2. P89LV51RD2 initiates a read of the PCA9554A input register (first snapshot)
    3. P89LV51RD2 initiates a second read of the PCA9554A input register (second snapshot) about 750 ms after

    If the second read is the same as the first one, then the master will continue to poll the PCA9554A input register and execute the associated command until the user releases the pushbutton.

  Notes:
  - Connection of the PCA9554A Interrupt pin to the P89LV51RD2 or to the P89LPC932 is done through jumper JP11.
    a) JP11 between 1 and 2 connects the PCA9554A Interrupt pin to the P89LPC932 device
    b) JP11 between 2 and 3 connects the PCA9554A Interrupt pin to the P89LV51 device
  - Polling or interrupt monitoring of the PCA9554A by the P89LPC932 microcontroller requires having jumpers JP6 and JP7 closed. I^2C-bus 1 and I^2C-bus 2 need to be connected together since the PCA9554A is located on I^2C-bus 1.
- Pushbutton S9:
  Pushbutton S9 (RESET), when pressed, performs a reset to both P89LV51RD2 and PCA9531 devices to their power up default states. It is also used to enter and exit the P89LV51RD2 ISP mode (for more detail, refer to the paragraph "In-System Programming Mode".

In-System Programming Mode

P89LV51RD2 and P89LPC932 devices have a built-in ISP (In-System Programming) algorithm allowing them to be programmed without the need to remove them from the application. Also, a previously programmed device can be erased and reprogrammed without removal from the circuit board. In order to perform ISP operations, the microcontroller is powered up in a special "ISP mode". ISP mode allows the microcontroller to communicate with an external host device through the serial port, such as a PC or terminal. The microcontroller receives commands and data from the host, erases and reprograms code memory, etc. Once the ISP operations have been completed, the device is reconfigured so that it will operate normally the next time it is either reset or power cycled.

ISP programming for both devices can be done using Flash Magic. Flash Magic is a free, powerful, feature-rich Windows application that allows easy programming of Philips Flash microcontrollers. Flash Magic uses Intel Hex files as input to program the targeted device. For download information, refer to the paragraph "Download software, programs and documentation".

P89LV51RD2 ISP programming

a) Set jumpers JP2 and JP3 to target P89LV51RD2 device: both jumpers connected between 2 and 3
b) Connect the DB-9 cable between the PC serial port and the PCA9564 evaluation board DB-9 connector
c) Enter the P89LV51RD2 ISP mode as requested in the Flash Magic pop up window: This is done by pushing the RESET pushbutton (S9) one time.
d) Open Flash Magic and go through the five following steps:
   Step 1: Set the connection status and the type of microcontroller to be programmed: COM port, Baud Rate (9600), Device = 89LV51RD2
   Step 2: Flash erasing (part or all)
   Step 3: Select the Hex file to be loaded in the microcontroller
   Step 4: Options to be set (Memory verification, Security bits…)
   Step 5: Perform the operations described in the steps above (click on "START" button)
   Programming of the blocks is displayed at the bottom of the Flash Magic window.
e) Exit the P89LV51RD2 ISP mode when programming done ("Finished" displayed at the bottom of the Flash Magic window): This is done by pushing the RESET pushbutton one time again (S9)
f) Once device programming has successfully been executed, the microcontroller can run the new program.

P89LPC932 ISP programming

a) Set jumpers JP2 and JP3 to target P89LPC932 device: both jumpers connected between 1 and 2
b) Connect the DB-9 cable between the PC serial port and the PCA9564 evaluation board DB-9 connector
c) Enter the P89LPC932 ISP mode: This is done by setting the following jumpers:
   - JP10 (RSTISP) closed
   - JP15 (VDDISP) open
   - JP6 and JP7 (I2CConnect) open
   - JP12 (SCLLV) and JP13 (SDALV) open
d) Open Flash Magic and go through the 6 following steps:
   Step 1: Set the connection status and the type of microcontroller to be programmed: COM port, Baud Rate (9600), Device = 89LPC932
   Step 2: Go to: Options → Advanced Options → Hardware Config
   Check the box "Use DTR and RTX to enter ISP mode"
   Step 3: Flash erasing (part or all)
   Step 4: Select the Hex file to be loaded in the microcontroller
   Step 5: Options to be set (Memory verification, Security bits…)
   Step 6: Perform the operations described in the steps above (click on "START" button).
   Programming of the blocks is displayed at the bottom of the Flash Magic window.
e) Exit the P89LV51RD2 ISP mode when programming done ("Finished" displayed at the bottom of the Flash Magic window): This is done by setting:
   - JP10 (RSTISP) open
   - JP15 (VDDISP) closed
   - State of JP6, JP7, JP12 and JP13 are function of the program requirements
f) Once device programming has successfully completed, exit from the ISP. The microcontroller is now ready to run the new program.

Other features

Write Protect PCF85116

JP1 allows data protection in the PCF85116 EEPROM:

- JP1 open: data in the EEPROM is write protected
- JP1 closed: writing to the EEPROM is allowed – memory is not protected

Use of other 80C51 type Philips microcontrollers

Any Philips 80C51 microcontroller pin to pin compatible with the P89LV51Rx2 device can be used as to interface with the PCA9564.

- Power supply:
  It can be chosen from:
  - The internal 3.3 V regulated voltage: Jumper VDDMCU+ closed
  - An external regulated voltage: Jumper VDDMCU+ open, external voltage applied to VCCMCU+

  If an external voltage is applied to the microcontroller, digital signals interfacing with the PCA9564 will be pulled up to this external voltage value.
  Caution: Since the PCA9564 is 5.5 V tolerant, no voltage greater than 5.5 V must be applied to the VDDMCU+ pin.
- Microcontroller with built-in I^2C interface:
  Port P1.6 (SCL) and P1.7 (SDA) can be connected to the internal I^2C-bus 2 (connector JP5) through jumpers JP12 and JP13.
  - JP12 open: P1.6 not connected to SCL2
  - JP12 closed:P1.6 connected to SCL2
  - JP13 open: P1.7 not connected to SDA2
  - JP13 closed:P1.7 connected to SDA2
- ISP mode:
  ISP mode for P89C51Rx+/Rx2/66x devices can also be entered by forcing the /PSEN pin to LOW. This is performed through the jumper JP9.
  - JP9 open: $\overline{\text{PSEN}}$ floating
  - JP9 closed: $\overline{\text{PSEN}}$ forced to ground

Use of any other non 80C51 type master devices

Any other non-80C51 type microprocessor, DSP, ASIC or emulator can be used with the PCA9564 evaluation board.
When an external device is used:

1) Remove the P89LV51RD2 microcontroller from its socket
2) Apply the 8-bit parallel bus data on connector JP16. Built-in pull up resistors can be disconnected by opening the jumper VDDMCU+.
   Note: RESET pushbutton (S9) cannot longer be used when VDDMCU+ is open
3) Apply PCA9564 control signals and monitor Interrupt pin (open drain output) on connector JP14

Caution: Since the PCA9564 is 5.5 V tolerant, no voltage greater than 5.5 V must be applied to the parallel bus data and the control signals

Communication between the 2 microcontrollers

- Communication through the I^2C-bus:
  Jumpers JP6 and JP7 allow to connect or split the I^2C-bus in one same bus or 2 different buses.
  I^2C-bus 1 contains the following devices: P89LV51RD2/ PCA9564, PCA9531, PCF85116 and PCA9554A
  I^2C-bus 2 contains the following devices: P89LPC932, P89xx51 with built-in SCL/SDA (when jumpers JP12 and JP13 are closed).
  - JP6 open: SCL Bus 1 and SCL Bus 2 are not connected together
  - JP6 closed: SCL Bus 1 and SCL Bus 2 are connected together
  - JP7 open: SDA Bus 1 and SDA Bus 2 are not connected together
  - JP7 closed: SDA Bus 1 and SDA Bus 2 are connected together

  Since the PCA9564 is a multi-master capable device, both microcontrollers can be a master in the same bus (when JP6 and JP7 closed). If both masters try to take control of the I^2C-bus at the same time, an arbitration procedure will be performed between the P89LV51RD2/PCA9564 and the P89LPC932.
- Communication through RxD and TxD pins:
  An additional non-I^2C communication channel between the 2 microcontrollers is available through their RxD and TxD pins.
  P89LV51 TxD pin can be connected to the P89LPC932 RxD pin through jumper JP17
  - JP17 open: pins are not connected together
  - JP17 closed: pins are connected together

P89LV51 RxD pin can be connected to the P89LPC932 TxD pin through jumper JP18
- JP18 open: pins are not connected together
- JP18 closed: pins are connected together

Note:
Jumpers JP2 and JP3 must be open when JP17 and JP18 need to be closed.

Miscellaneous

- Power supply for daughter cards connected to the I^2C-bus connectors:
  Jumper JP8 (VDD Conn), when closed, connect the V_{DD} pins in the two I^2C-bus connectors (JP4 and JP5) to the internal 3.3 V regulated voltage, thus allowing daughter cards to be supplied directly by the main board
  - JP8 open: V_{DD} pin in the two I^2C-bus connectors is floating
  - JP8 closed: V_{DD} pin in the two I^2C-bus connectors is connected to the internal 3.3 V regulated voltage
- General purpose LEDs:
  Several LEDs are connected to the P89LV51RD2 and the P89LPC932 for debugging or general-purpose use.
  LD1 to LD8 are accessible by both microcontrollers through I^2C by programming the PCA9531.

LED	Pin	Device	LED	Pin	Device
LD1	LED0	PCA9531	LD13	P2.0	P89LPC932
LD2	LED1	PCA9531	LD14	P2.1	P89LPC932
LD3	LED2	PCA9531	LD15	P2.2	P89LPC932
LD4	LED3	PCA9531	LD16	P2.3	P89LPC932
LD5	LED4	PCA9531	LD17	P2.4	P89LPC932
LD5	LED5	PCA9531	LD18	P2.5	P89LPC932
LD7	LED6	PCA9531	LD19	P2.6	P89LPC932
LD8	LED7	PCA9531	LD20	P2.7	P89LPC932
LD9	P2.2	P89LV51RD2	LD21	P1.4	P89LPC932
LD10	P2.3	P89LV51RD2	LD22	P1.6	P89LPC932
LD11	P2.4	P89LV51RD2	LD23	P1.7	P89LPC932
LD12	P2.5	P89LV51RD2			

Table 3. Evaluation board LEDs

- General Purpose jumpers for P89LPC932:
  Jumpers JP19, JP20 and JP21 allows to force HIGH or LOW logic levels respectively on pins P0.0, P0.1 and P0.2 of the P89LPC932.
  - JPxx open: the corresponding port is set to HIGH
  - JPxx closed: the corresponding input port is set to LOW
- General purpose headers for both microcontrollers:
  PORT1LV51 and PORT0LPC headers allow to easily access to Port 0 of each device for monitoring or external control. V_{DD} and GND pins are also available.
  Note:
  Header labeled “3v3” on PORT0LV51 is actually connected to VDDMCU+ pin. The voltage on this node can be externally supplied and is limited to 5.5 V.

TECHNICAL INFORMATION – EMBEDDED FIRMWARE

Overview

PCA9564 evaluation board is delivered with 4 different embedded firmware programs (Program 1 to Program 4) allowing the user to run simple applications in order to evaluate the PCA9564's capabilities, to monitor data and control signals with the P89LV51RD2 master, and the I^2C slave devices present in the evaluation board. Besides the external power supply, no external hardware or software is required to run those applications. Embedded programs are erased as soon as the microcontroller is reprogrammed with a different code. The embedded programs require programming of both P89LV51RD2 and P89LPC932 and "Hex" files can be downloaded from www.standardproducts.philips.com website. "Hex" files can be loaded to the microcontrollers by using their ISP mode with Flash Magic software. For more information about ISP mode and file downloading, refer to the paragraphs "In-System Programming mode" and "Download software, programs and documentation".

- Pushbuttons S1 to S8 allow program selection (S8) and initiate specific actions for each program (S1 to S7). PCA9554A is used to collect actions performed on the pushbuttons and inform the P89LV51RD2 that a reading routine to determine the nature of the action is requested. Pushing S8 does jump from one program to another (from Program 1 to Program 4, then again Program 1…).
- LD9 and LD10 display the number of the selected program
- LD11 and LD12 display program specific information

- **Program 1 (LD9 = OFF, LD10 = OFF): PCA9531 dynamic programming**
  Program 1 uses the P89LV51RD2/PCA9564 as an I^2C master, the PCA9531 (with LD1 to LD8) as an I^2C slave to dynamically change blinking rates and output states.
  LD1 to LD4 are programmed to blink at Blinking rate 0 (BR0), while LD5 to LD8 are programmed to blink at Blinking Rate 1 (BR1).
  Actions on the pushbuttons:
  - S1: Decrease blinking frequency for both BR0 and BR1 (single shot or permanent push modes)
  - S2: Decrease duty cycle for both BR0 and BR1 (single shot or permanent push modes)
  - S3: Select the Blinking Rate (BR0 or BR1) to be programmed through S1, S2, S5, S6 and S7
  - S4: Reset the programming and program the LEDs to their default blinking frequency
  - S5: Increase blinking frequency for both BR0 and BR1 (single shot or permanent push modes)
  - S6: Increase duty cycle for both BR0 and BR1 (single shot or permanent push modes)
  - S7: Program the LEDs to be OFF or blinking at BR0 or BR1
  - S8: Jump to the next program (Program 2)

  LD11 and LD12 provide the following information:
  - LD11 = OFF → BR0 programming selected (LD1 to LD4)
  - LD11 = ON → BR1 programming selected (LD5 to LD8)
  - LD12 = ON → Default blinking rate set to the PCA9531
  - LD12 = OFF → PCA9531 has been programmed by the user and blinking is different from default values

- **Program 2 (LD9 = ON, LD10 = OFF): Preprogrammed blinking patterns**
  Program 2 uses the P89LV51RD2/PCA9564 as an I^2C master, the PCF85116, the PCA9531 (with LD1 to LD8) and the P89LPC932 (with LD13 to LD20) as I^2C slaves to display preprogrammed blinking patterns stored in the EEPROM.
  For a specific selected pattern:
  a) Data used to program the PCA9531is read from the EEPROM. Data organization is shown in Figure 4.
  b) The PCA9531 is then programmed with the data previously read.

  Action on the pushbuttons:
  - S4: Scans the EEPROM in order to determine location of the different patterns (first and last cell numbers for each programmed pattern).
  - S5: Select the pattern to be read from the EEPROM and to be programmed in the PCA9531. Scan of the EEPROM must be performed first before being able to select between the different patterns.
  - S8: Jump to the next program (Program 3)

LD12 provides the following information:

- LD12 = OFF → Scan of the EEPROM not performed
- LD12 = ON → Scan of the EEPROM performed

LD13 to LD20 display the number of the pattern currently selected.

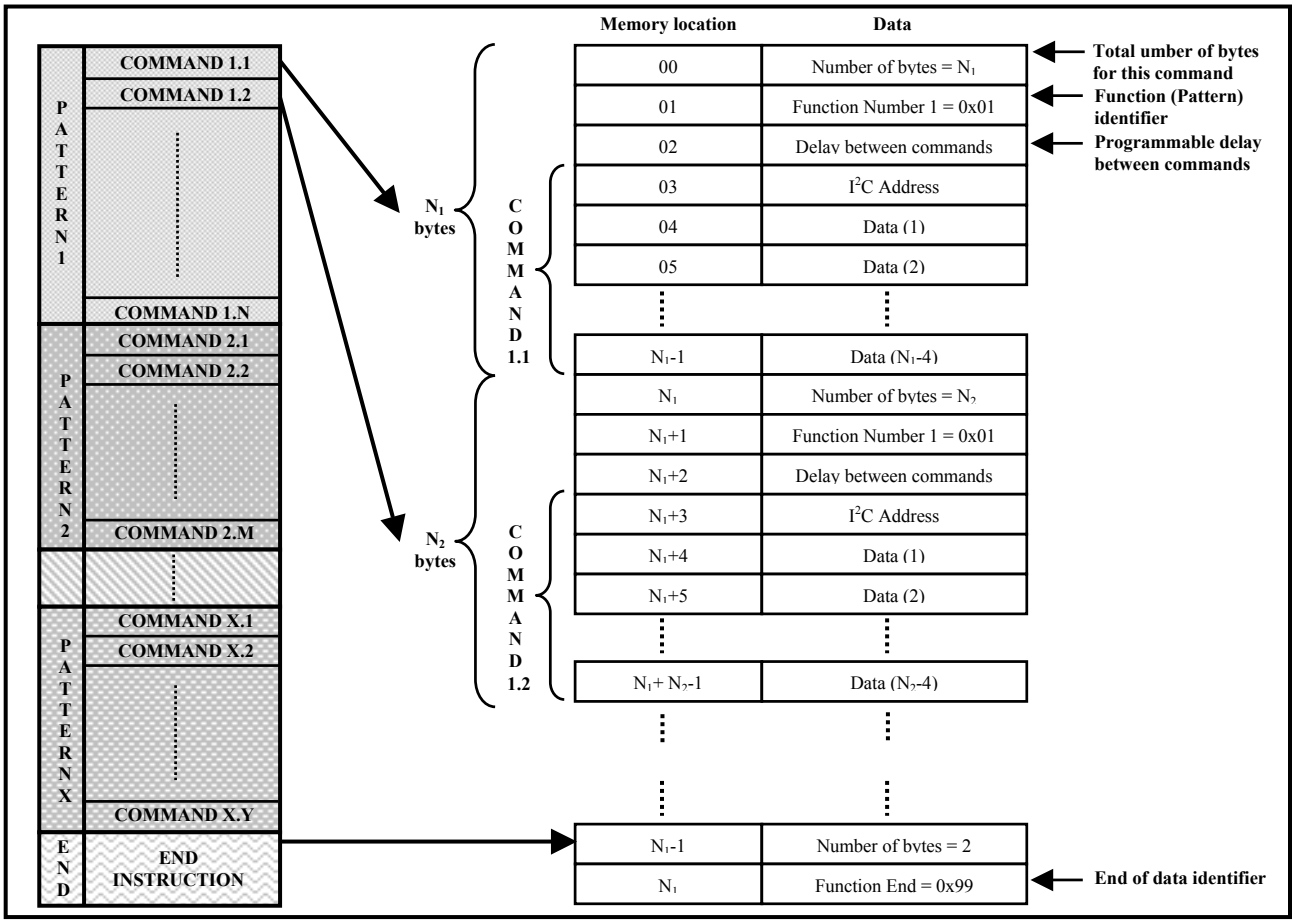


Figure 4. PCF85116 memory organization

- **Program 3 (LD9 = OFF, LD10 = ON): P89LPC932 LED programming**

  Program 3 uses P89LV51RD2/PCA9564 as an I^2C master, the PCA9531 (with LD1 to LD8) and the P89LPC932 (with LD13 to LD20) as I^2C slaves to display a user definable byte on LD13 to LD20.
  Value of the byte to be programmed is displayed with LD1 (bit 0, LSB) to LD8 (bit 7, MSB)
  Once P89LPC932 has been programmed, the value is displayed with LD13 (bit 0, LSB) to LD20 (bit 7, MSB).
  Action on the pushbuttons:
  - S1: Decrease position of the bit to be programmed: 7 → 6 → 5 → 4 → 3 → 2 → 1 → 0 → 7 → …
  - S2: Invert the polarity of the logic value of the current bit, programmed logic value is displayed on LD1 to LD8: 0 → 1 → 0 → 1 …
    0: corresponding LED is OFF
    1: corresponding LED is ON
  - S3: Send the programmed byte to the P89LPC932 when programming has been done. LD13 to LD20 display the programmed byte value when command has been sent
    0: corresponding LED is OFF
    1: corresponding LED is ON
  - S4: Reset the programming and the value sent to the P89LPC932. LD1 to LD8, LD13 to LD20 are OFF.
  - S5: Increase position of the bit to be programmed: 0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 0 → …
  - S8: Jump to the next program (Program 4)

- **Program 4 (LD9 = ON, LD10 = ON): I^2C address search**

  Program 4 uses the P89LV51RD2/PCA9564 as an I^2C master and the P89LPC932 (with jumpers JP19 to JP21) as an I^2C slave. In this mode, the PCA9564 searches for the P89LPC932's I^2C slave address (JP19 to JP21 programs the 3 LSB's of the P89LPC932 I^2C slave address, the 4 MSB's of the address are fixed. The address is unknown to the P89LV51RD2)

  Action on the pushbuttons:
  - S1: Initiates the P89LPC932's I^2C address search routine
  - S2: Resets the P89LV51RD2 search routine algorithm and initiates a P89LPC932 I^2C address scanning and memorization. The P89LPC932 scans its GPIO's in order to memorize logic values associated with jumpers JP19 to JP21.
  - S8: Jump to the next program (Program 1)

  LD11 and LD12 provide the following information:
  - LD11 = OFF → I^2C address not found or search routine not performed yet
  - LD11 = ON → I^2C address search routine successful
  - LD12 = OFF → search routine not performed yet
  - LD12 = ON → search routine performed and I^2C address not found

Embedded programs flowcharts

Program Selection

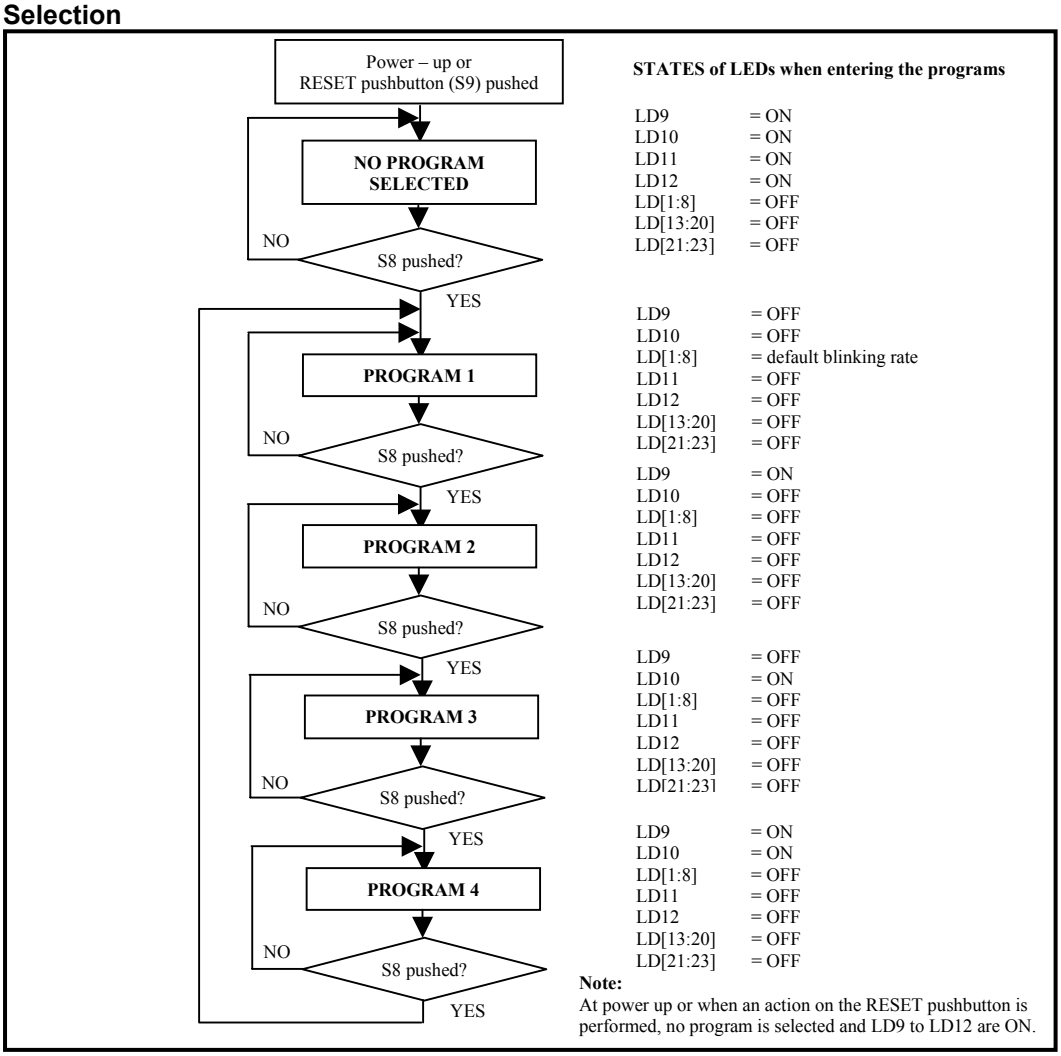


Figure 5. Program selection

Program 1: P89LV51RD2–PCA9564–PCA9531; PCA9531 dynamic programming

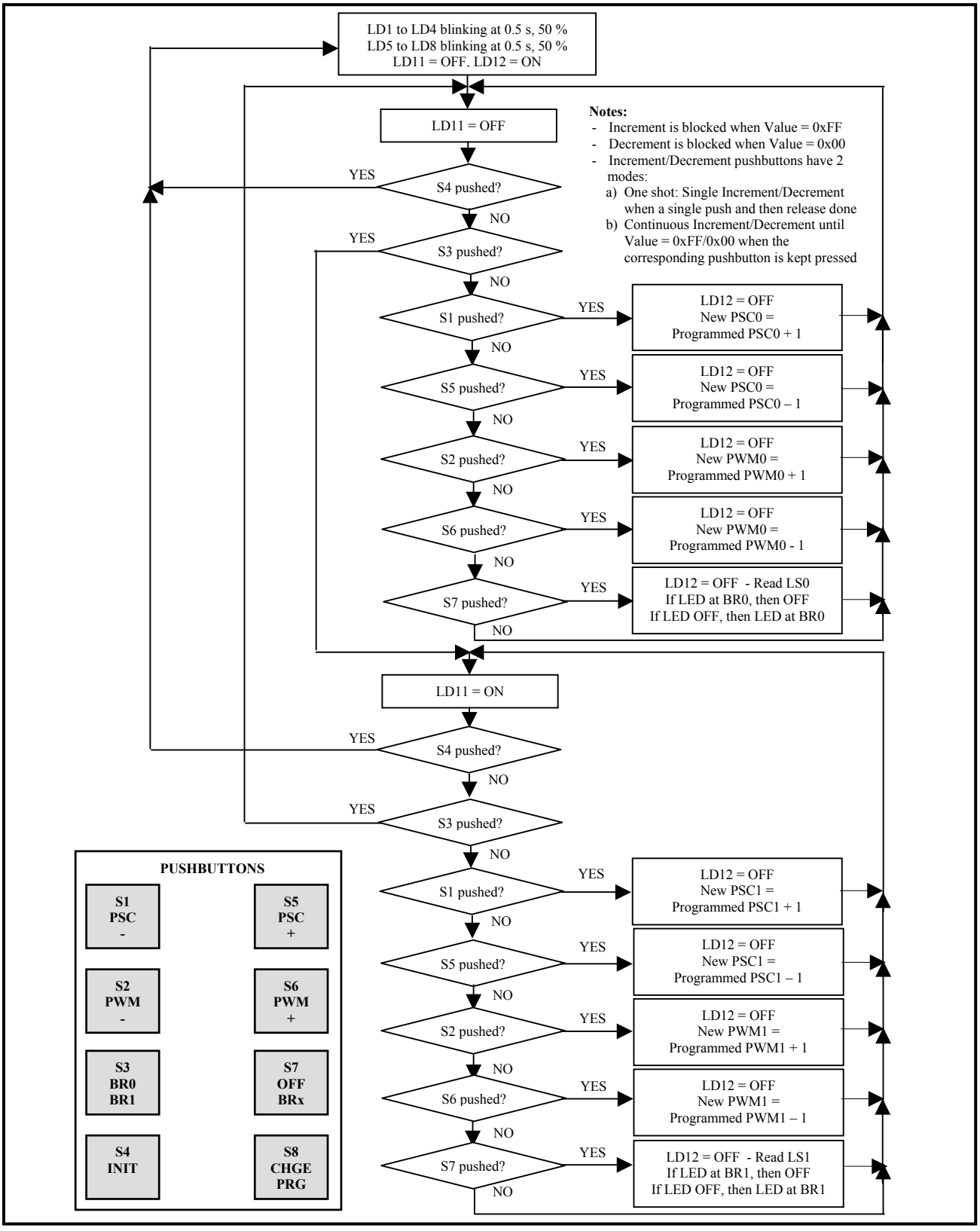


Figure 6. Program 1 – PCA9531 dynamic programming

Program 2: P89LV51RD2–PCA9564–PCA9531–PCF85116–P89LPC932; Predefined blinking patterns

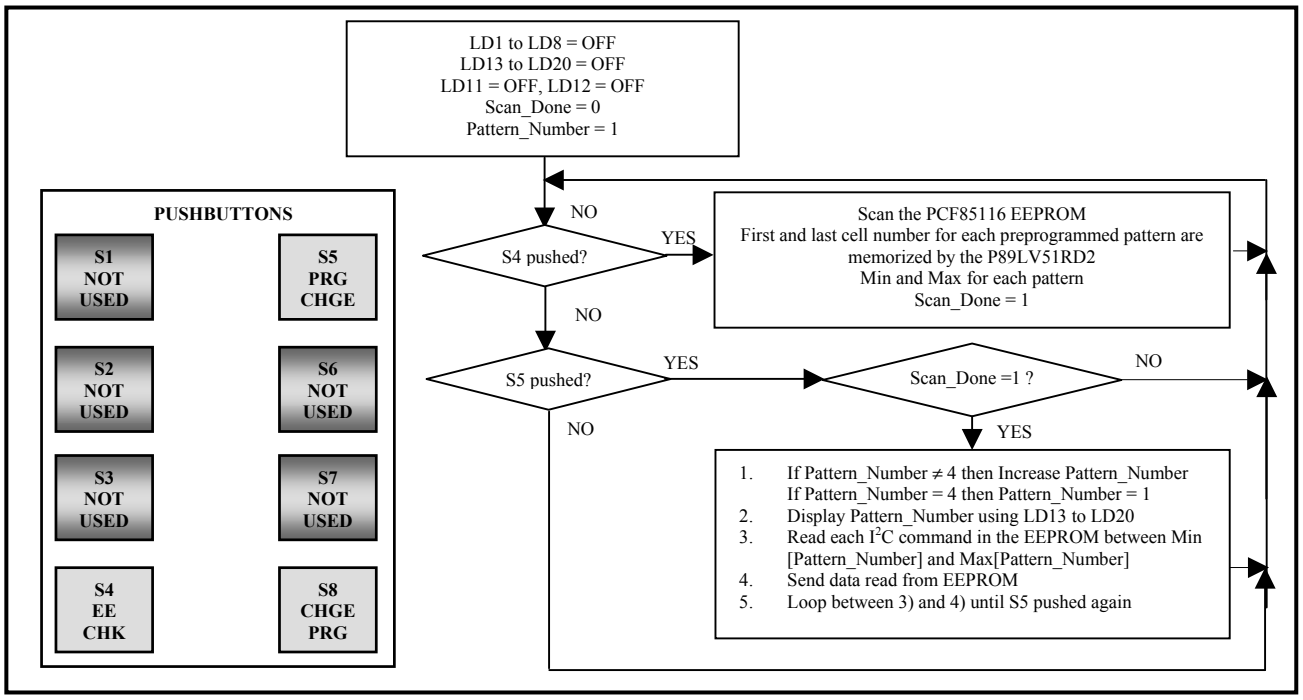


Figure 7. Program 2 – Preprogrammed blinking patterns

Program 3: P89LV51RD2–PCA9564–PCA9531–P89LPC932; P89LPC932 LED programming

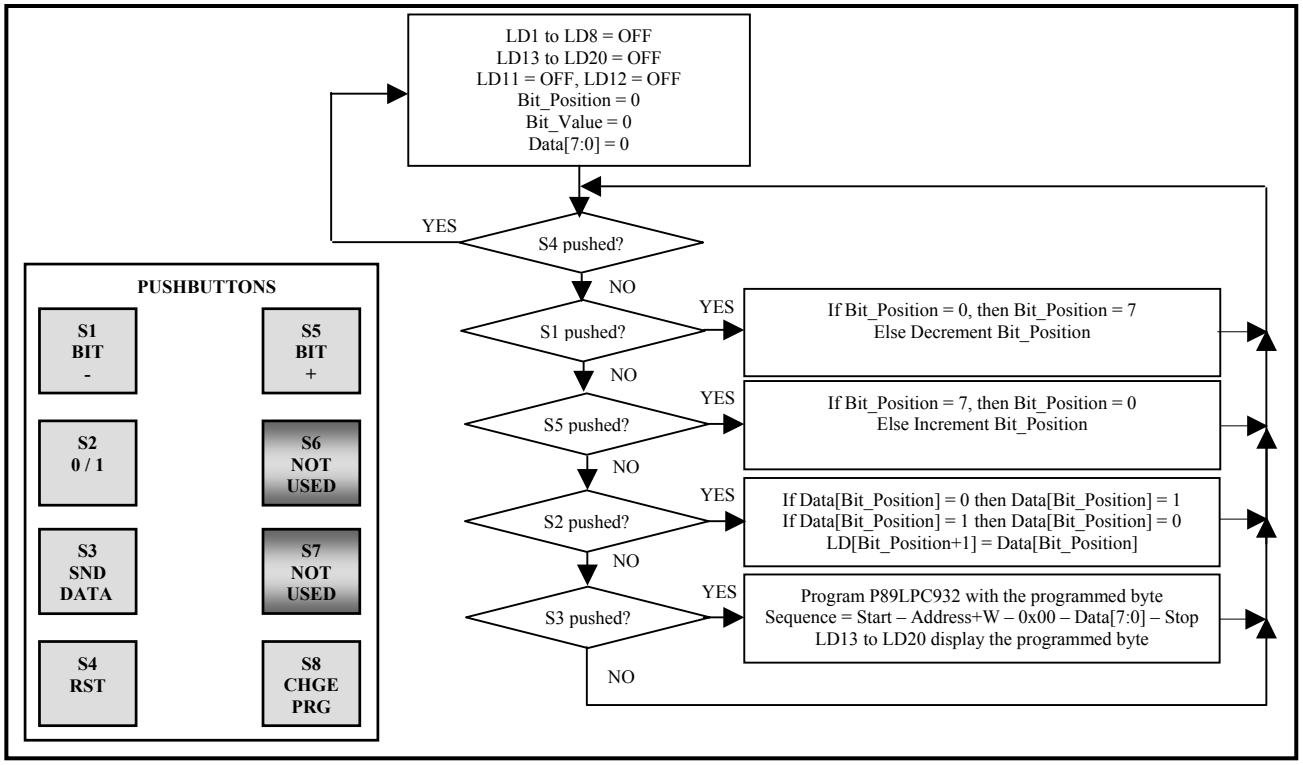


Figure 8. Program 3 – P89LPC932 LED programming

Program 4: P89LV51RD2–PCA9564–PCA9531–P89LPC932; I²C address search

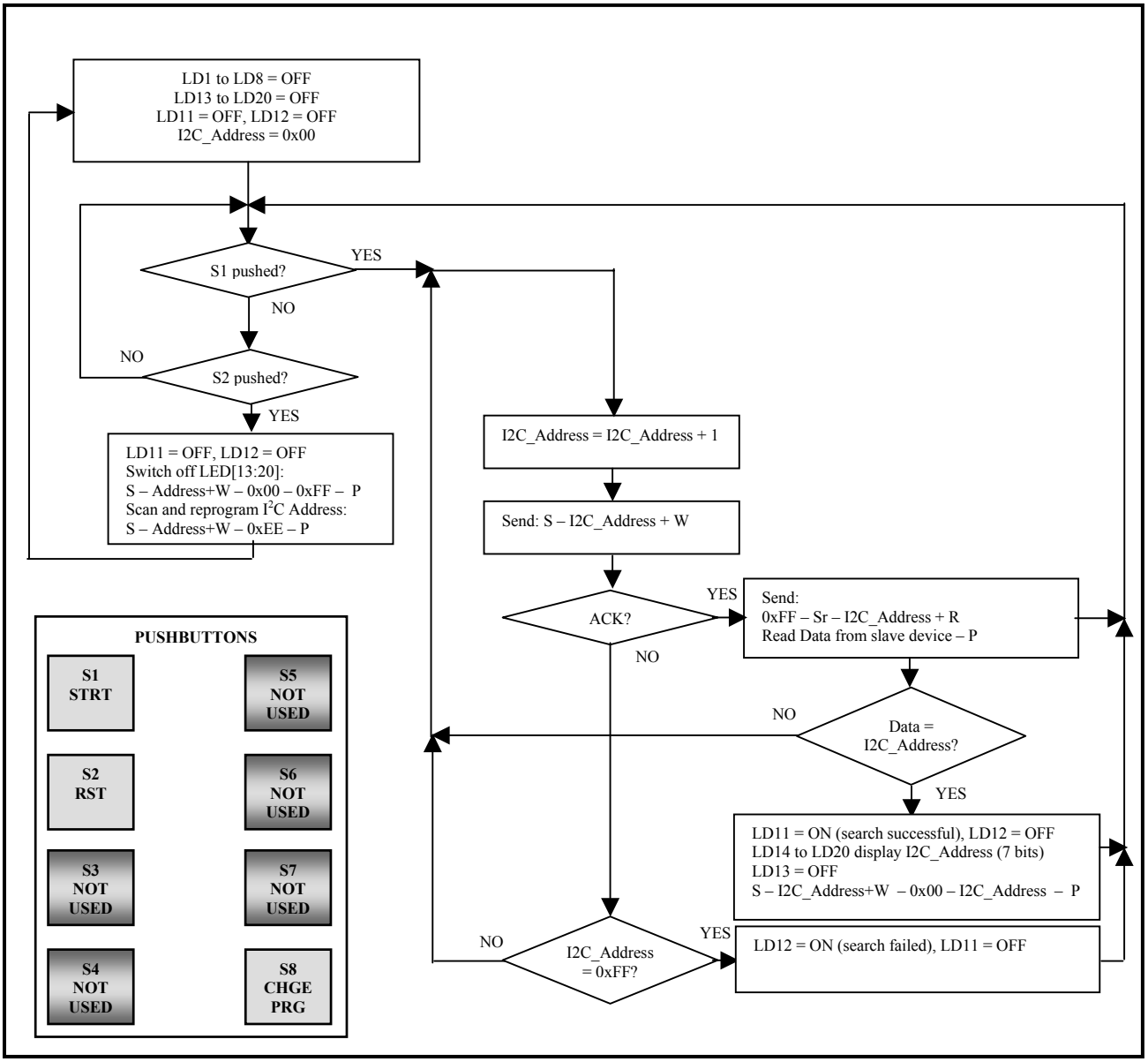


Figure 9. Program 4 – I²C address search

Source Code P89LV51RD2 – Rev 1.0

P89LV51RD2/PCA9564 source code of the embedded software is organized in several files written in C language. Modularity of the files allows building applications using an 8051-core microcontroller and a PCA9564 in an easy and intuitive way. Most of the files are core independent and can be used with different types of microcontrollers. Only the file generating the control signals and receiving/transmitting data is subject to modification depending on the type of microcontroller used.

The code in C language is divided in several files, organized as following:

1. **I2CEXPRT.H:**
2. Contains the definition of the different structures and functions used in the code.
3. **Mainloop.c:**
   Contains the main running loop:
   - Initialization at power up or reset
   - Call to the function handling the program selection
4. **I2C_Routines.c** and **I2C_Routines.h:**
   Contain the different programs selectable by the user. These files are generally those that need to be modified in order to develop specific programs or functions. Main functions are:
   - void **Blinker_Up_Down**(void): Function for Program 1
   - void **ReadEEprom**(short int MinEEPtr, short int MaxEEPtr, int Operation_EEprom, int Operation_Function) and void **Preset_Patterns_PCA9532**(void): Functions for Program 2
   - void **LV51_LPC932**(void): Function for Program 3
   - unsigned char **Search_Routine**(unsigned char min, unsigned char max) and void **I2C_Address_Search**(void): Functions for Program 4
   - void **GPIO_Interrupt_Handler**(void): Function handling actions on pushbuttons S1 to S8
5. **I2CDRIVR.C** and **I2CDRIVR.H:**
   Handle the selection between master and slave mode.
6. **I2CMASTR.C** and **I2CMASTR.h:**
   Contain the functions handling the Master Transmitter and Master Receiver modes. Handle the different states of the state machine and generate the sequencing of the commands based upon the previous command and the status information. Interface directly with the PCA9564 (read and write in a specific register)
7. **I2CINTFC.C:**
   Contains the description of the top functions used to send and receive I^2C messages:
   - Start, Write, Stop
   - Start, Read, Stop
   - Start, Write, Repeated Start, Read, Stop
   - Start, Write, Repeated Start, Write, Stop
8. **PCA9564sys.c** and **PCA9564sys.h:**
   Contain the actual interface between the microcontroller and the PCA9564: control signal generation, data writing and reading. This file is specific to an 8051-type microcontroller and needs to be changed if another type of microcontroller is used to interface with the PCA9564.
9. **Interrupts.c:**
   Contains the definition of the Interrupts – Not used in this program – For future reference

Complete source code can be found in Appendix 1 “P89LV51RD2 Microcontroller Source Code – Rev1.0”.

Source Code P89LPC932 – Rev 1.0

P89LPC932 microcontroller is used as a slave device with the default embedded programs and use only the slave part of the I^2C core.

1. **main.c:**
   Contains the instructions to interface with the P89LV51RD2/PCA9564 default embedded program:
   a) Instruction controlling LD[13:20]: S – Address+W – 0x00 – Data[7:0] – P
      - Data[0] = state LD13
      - Data[7] = state LD20
   b) Instruction controlling the “I^2C address Scan and Memorize” procedure: S – Address+W – 0xEE – P
   c) Instruction allowing reading back the I^2C slave address: S – Address+W – 0xFF – Sr – Address+R – Data – P with Data = I^2C slave address

2. **i2cslave.c:**
   Contains the source code of the I^2C slave core
3. **ua_exprt.h:**
   Contains the definition of variables used in the I^2C slave core

Complete source code can be found in Appendix 2 "P89LPC932 Microcontroller Source Code – Rev1.0".

Download software, programs and documentation

- The **Raisonance free evaluation development kit** can be downloaded from: http://www.amrai.com/amrai.htm
  1. In the "Software" yellow box, select 8051
  2. Fill the form
  3. Download the "kit51.exe" file and the "GettingStartedManual.pdf"
  4. Install the software by running "kit51.exe"

  The Raisonance 8051 Development Kit is a complete solution to creating software for the 8051 family family of microcontroller. The Development Kit comprises many different tools that allow projects ranging from simple to highly complex to be developed with relative ease. The free evaluation version can be used to develop up to 4 kbits of code that can be loaded into the P89LV51 or P89LPC932 by using Flash Magic software.

- **Flash Magic** software from Embedded Systems Academy can be downloaded from:
  http://www.esacademy.com/software/flashmagic/
  1. In the download section (bottom of the page), download the file using http or ftp
  2. Install the software using the downloaded ".exe" file

  Flash Magic is a free, powerful, feature-rich Windows application that allows easy programming of Philips Flash Microcontrollers.

- All the information about Philips microcontrollers (Datasheets, Application Notes, Support Tools…) can be found in the **Philips microcontroller homepage** at:
  http://www.semiconductors.philips.com/markets/mms/products/microcontrollers/

PCA9564 evaluation board web page

PCA9564 evaluation board homepage that can be found at:
http://www.standardproducts.philips.com/support/boards/pca9564
It contains the following:

- Source code in C-language for the manufacturing default firmware used in the P89LV51RD2 and P89LPC932
- Application Note AN10148 and AN10149
- Datasheet of the different I^2C slave devices and µcontrollers used in the PCA9564 evaluation board
- Links to the 3rd party tools (Flash Magic, Raisonance)
- IBIS model
- How to order the PCA9564 Evaluation Board
- …

Appendix 1: P89LV51RD2 Microcontroller Source Code – Rev 1.0

I2CEXPRT.H

```
//***************************************************************************
//
//                 P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name: i2cexpert.h
// Created:   June 2, 2003
// Modified:  June 4, 2003
// Revision:  1.00
//
//***************************************************************************

#include <REG51RX.H>

typedef unsigned char    BYTE;
typedef unsigned short   WORD;
typedef unsigned long    LONG;

typedef struct                  // each message is configured as follows:
{
   BYTE   address;              // slave address to sent/receive message
   BYTE   nrBytes;              // number of bytes in message buffer
   BYTE   *buf;                 // pointer to application message buffer
} I2C_MESSAGE;

typedef struct                  // structure of a complete transfer
{                               // made up of a number of messages and pointers to the messages
  BYTE        nrMessages;       // number of message in one transfer
  I2C_MESSAGE **p_message;      // pointer to pointer to message
} I2C_TRANSFER;

/*****************************************************************/
/*        E X P O R T E D   D A T A   D E C L A R A T I O N S     */
/*****************************************************************/

#define FALSE       0
#define TRUE        1

#define I2C_WR      0
#define I2C_RD      1

#define PCA9531_WR   0xC8               // i2c address LED Dimmer - Write operation
#define PCA9531_RD   0xC9               // i2c address LED Dimmer - Read operation
#define PCA9554_WR   0x7E               // i2c address i/o expander - Write operation
#define PCA9554_RD   0x7F               // i2c address i/o expander - Read operation

/**** Status Errors ****/

#define I2C_OK                    0     // transfer ended No Errors
#define I2C_BUSY                  1     // transfer busy
#define I2C_ERROR                 2     // err: general error
#define I2C_NO_DATA               3     // err: No data in block
#define I2C_NACK_ON_DATA          4     // err: No ack on data
#define I2C_NACK_ON_ADDRESS       5     // err: No ack on address
#define I2C_DEVICE_NOT_PRESENT    6     // err: Device not present
#define I2C_ARBITRATION_LOST      7     // err: Arbitration lost
#define I2C_TIME_OUT              8     // err: Time out occurred
#define I2C_SLAVE_ERROR \         9     // err: Slave mode error
#define I2C_INIT_ERROR            10    // err: Initialization (not done)
#define I2C_RETRIES               11    // err: Initialization (not done)

/*****************************************************************/
/*      I N T E R F A C E   F U N C T I O N   P R O T O T Y P E S */
/*****************************************************************/

extern void I2C_InitializeMaster(BYTE speed);
extern void I2C_InitializeSlave(BYTE slv, BYTE *buf, BYTE size, BYTE speed);
extern void I2C_InstallInterrupt(BYTE vector);
extern void I2C_Interrupt(void);

extern void I2C_Write(I2C_MESSAGE *msg);
extern void I2C_WriteRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2);
```

```
extern void I2C_WriteRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2);
extern void I2C_Read(I2C_MESSAGE *msg);
extern void I2C_ReadRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2);
extern void I2C_ReadRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2);

extern void Blinker_Up_Down(void);
extern void LV51_LPC932(void);
extern void ReadEEprom(short int MinEEPtr, short int MaxEEPtr, int Operation_EEprom, int Operation_Function);
extern void Preset_Patterns_PCA9532(void);
extern void I2C_Address_Search(void);
extern void Init_Slaves(void);
extern void Init_LPC932(void);
extern unsigned char Search_Routine(unsigned char min, unsigned char max);
extern void GPIO_Interrupt_Handler(void);
extern void InsertDelay(unsigned char delayTime);

static sbit LED0        = P2^2;       // LD[9:12] mapped with LV51's P2[2:5]
static sbit LED1        = P2^3;
static sbit LED2        = P2^4;
static sbit LED3        = P2^5;

static sbit PCA9554_Int = P3^2;       // Interrupt PCA9554 mapped with LV51's P3[2]
sbit PCA9564_Reset      = P3^4;       // Reset PCA9564 mapped with LV51's P3[4]
```

Mainloop.c

```
//*************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   mainloop.c
// Created:     June 2, 2003
// Modified:    November 07, 2003
// Revision:    1.00
//
//*************************************************************************

#include <REG51RX.H>
#include "i2cexprt.h"
#include "PCA9564sys.h"
#include "I2C_Routines.h"

idata BYTE  Buffer1[32];
idata BYTE  Buffer2[32];
idata BYTE  Buffer3[16];
idata BYTE  Buffer4[16];

idata I2C_MESSAGEMessage1;
idata I2C_MESSAGEMessage2;
idata I2C_MESSAGEMessage3;
idata I2C_MESSAGEMessage4;

static short int ProgramCounter = 0;

//*************************************************************************
// Initialization Functions at power up, Reset or program change
//*************************************************************************

static void Init_PCA9564(void)
{
  PCA9564_Reset = 1;
  PCA9564_Reset = 0;
  InsertDelay(1);                         // PCA9564 reset time = 1 ms
  PCA9564_Reset = 1;
  AUXR = 2;                               // External memory space
  I2C_InitializeMaster(0x00);             // 330 kHz
}

static void Init_Slaves(void)
{
  Message1.address   = PCA9531_WR;
  Message1.buf       = Buffer1;
  Message1.nrBytes   = 7;
  Buffer1[0]         = 0x11;              // autoincrement + register 1
  Buffer1[1]         = 0x80;              // default prescaler pwm0
  Buffer1[2]         = 0x80;              // default duty cycle for pwm0
  Buffer1[3]         = 0x80;              // default prescaler pwm1
  Buffer1[4]         = 0x80;              // default duty cycle for pwm1
```

```
  Buffer1[5]         = 0x00;                 // LD1 to LD4 off
  Buffer1[6]         = 0x00;                 // LD5 to LD8 off

  I2C_Write(&Message1);                      // LD[1:8] off

  Message2.address   = PCA9554_WR;
  Message2.buf       = Buffer2;
  Message2.nrBytes   = 1;
  Buffer2[0]         = 0;                    // subaddress = 0

  Message3.address   = PCA9554_RD;
  Message3.buf       = Buffer3;
  Message3.nrBytes   = 1;                    // read one byte
}

//*****************************************************************************
// Delay time in milliseconds
// Insert a wait into the program flow
// Use Timer 1
// Do not use an interrupt
// Oscillator running at 11.0592 MHz
// 6 clock cycles per clock tick
// Therefore, we need 1843 cycles for 1msec
//*****************************************************************************

void InsertDelay(unsigned char delayTime)
{
  unsigned char i;

  TMOD = (TMOD & 0x0F) | 0x01;               // 16-bit timer
  TR1 = 0;
  for (i=0;i<delayTime;i++)
  {
    TF1 = 0;
    TH1 = 0xF8;                              // set timer1 to 1843
    TL1 = 0xCD;                              // since it's an up-timer, use  (65536-1843) = 63693 = F8CD
    TR1 = 1;                                 // Start timer
    while(TF1==0);                           // wait until Timer1 overflows
  }
}

//*****************************************************************************************
// Toggles pushbutton S8 in order to determine which  program the user wants to run
//*****************************************************************************************

static void Program_Selection(void)
{
  if (Buffer3[0] == 0x7F)                            // Push on S8 detected
  {
    if (ProgramCounter < 4)
    {
      ProgramCounter++;                              // Program selection incremented
    }
    else
    {
      ProgramCounter = 1;                            // Program selection back to 1
    }
  }
  switch (ProgramCounter)
  {
    case 1 : LED0 = 1;                               // LD9   off
             LED1 = 1;                               // LD10  off
             Buffer3[0]  = 0xFF;
             Blinker_Up_Down();                      // Blinker PSC and PWM Up/down program is selected
             break;
    case 2 : LED0 = 0;                               // LD9   on
             LED1 = 1;                               // LD10  off
             Buffer3[0] = 0xFF;
             Preset_Patterns_PCA9531();              // PCA9531 preset patterns program selected
             break;
    case 3 : LED0 = 1;                               // LD9   off
             LED1 = 0;                               // LD10  on
             Buffer3[0]  = 0xFF;
             LV51_LPC932();                          // LPC932 LED programming program is selected
             break;

    case 4 : LED0 = 0;                               // LD9   on
             LED1 = 0;                               // LD10  on
             Buffer3[0] = 0xFF;
             I2C_Address_Search();                   // LPC932 I2C address search program selected
             break;
```

```
  }
}

//*****************************************************************************
// Main program
//*****************************************************************************

void main(void)
{
  Init_PCA9564();                       // Initialization PCA9564
  Init_Slaves();                        // Initialization slave devices
  Init_LPC932();                        // Initialization LPC932
  LED0 = 0;                             // LD9 on at power up or after reset
  LED1 = 0;                             // LD10 on at power up or after reset
  LED2 = 0;                             // LD11 on at power up or after reset
  LED3 = 0;                             // LD12 on at power up or after reset

  while (1)
  {
    GPIO_Interrupt_Handler();
    Program_Selection();                // Toggles S8 in order to determine which program is selected by the user
  }
}
```

I2C_Routines.h

```
//*****************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   I2C_Routines.c
// Created:     June 2, 2003
// Modified:    November 07, 2003
// Revision:    1.00
//
//*****************************************************************************

unsigned char Search_Routine(unsigned char min, unsigned char max);
void GPIO_Interrupt_Handler(void);
void Blinker_Up_Down(void);
void ReadEEprom(short int MinEEPtr, short int MaxEEPtr, int Operation_EEprom, int Operation_Function);
void Preset_Patterns_PCA9531(void);
void LV51_LPC932(void);
void I2C_Address_Search(void);
```

I2C_Routines.c

```
//*****************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   I2C_Routines.c
// Created:     June 2, 2003
// Modified:    November 07, 2003
// Revision:    1.00
//
//*****************************************************************************

#include <REG51RX.H>
#include "i2cexprt.h"
#include "PCA9564sys.h"

idata BYTE Snapshot_1 = 0x0F;
idata BYTE Snapshot_2 = 0x00;
int Trigger_GPIO_Polling;
int Search_Successful = 0;
unsigned char Data_Received;
unsigned char LPC932_WR;
unsigned char LPC932_RD;

extern unsigned char LPC932_WR;
extern unsigned char LPC932_RD;
```

```
extern unsigned char CRX;

extern idata BYTE  Buffer1[32];
extern idata BYTE  Buffer2[32];
extern idata BYTE  Buffer3[16];
extern idata BYTE  Buffer4[16];

extern idata I2C_MESSAGE  Message1;
extern idata I2C_MESSAGE  Message2;
extern idata I2C_MESSAGE  Message3;
extern idata I2C_MESSAGE  Message4;

//*************************************************************************
// I2C Address Search Routine
// Make the search between min and max
// Return the I2C Address and set the Search_Successful bit
// to 1 when search has been successful
//*************************************************************************

unsigned char Search_Routine(unsigned char min, unsigned char max)
{
  unsigned char I2C_Address_Write;
  unsigned char I2C_Address_Read;
  unsigned char Address_Sent_Status;
  unsigned char Command_Sent_Status;
  unsigned char Counter_I2C_Address_Write = min;
  unsigned char Counter_I2C_Address_Read = min+1;

  int i;

  Search_Successful = 0;

  while (Counter_I2C_Address_Write != max & Search_Successful == 0)  // Search routine starts
  {
    Counter_I2C_Address_Write++;
    Counter_I2C_Address_Write++;                          // Increment I2C Address Write (+2)
    Counter_I2C_Address_Read++;
    Counter_I2C_Address_Read++;                           // Increment I2C Address Read (+2)
    I2C_Address_Write = Counter_I2C_Address_Write;
    I2C_Address_Read = Counter_I2C_Address_Read;
    PCA9564_Write(I2CCON,0xE0 | CRX);                     // 1110 0xxx -> generate Start
    for (i=0; i < 200;i++);
    PCA9564_Write(I2CDAT,I2C_Address_Write);              // Send Address Byte + W
    for (i=0; i < 200;i++);
    PCA9564_Write(I2CCON,0xC0 | CRX);                     // I2CCON=11000xxx
    for (i=0; i < 200;i++);
    Address_Sent_Status = PCA9564_Read(I2CSTA);           // Read status Register
    switch (Address_Sent_Status)
    {
      case 0x18 : //Ack received
                  PCA9564_Write(I2CDAT,0xFF);                     // send Command byte (0xFF)
                  for (i=0; i < 200;i++);
                  PCA9564_Write(I2CCON,0xC0 | CRX);               // I2CCON=11000xxx
                  for (i=0; i < 200;i++);
                  Command_Sent_Status =  PCA9564_Read(I2CSTA);
                  PCA9564_Write(I2CCON,0xD0 | CRX);               // send Stop
                  for (i=0; i < 200;i++);
                  if (Command_Sent_Status == 0x28)                // Command byte has been ack'ed
                  {
                    PCA9564_Write(I2CCON,0xE0 | CRX);             // 1110 0xxx -> generate Start
                    for (i=0; i < 200;i++);
                    Command_Sent_Status =  PCA9564_Read(I2CSTA);
                    if (Command_Sent_Status == 0x08)              // Start = OK
                    {
                      PCA9564_Write(I2CDAT,I2C_Address_Read);     // send Address Byte + R
                      for (i=0; i < 200;i++);
                      PCA9564_Write(I2CCON,0xC0 | CRX);           // I2CCON=11000xxx
                      for (i=0; i < 200;i++);
                      Command_Sent_Status =  PCA9564_Read(I2CSTA);
                      if (Command_Sent_Status == 0x40)            // Addr + R = OK
                      {
                        PCA9564_Write(I2CCON,0x40 | CRX);         // Read Data and NACK
                        for (i=0; i < 200;i++);
                        Data_Received = PCA9564_Read(I2CDAT);
                      }
                    }
                  }
                  PCA9564_Write(I2CCON,0xD0 | CRX);               // send Stop
                  if (Data_Received == I2C_Address_Write)
                  {
                    Search_Successful = 1;                        // Search successful if Read Data = Address
```

```
                }
                else
                {
                  Search_Successful = 0;                        // Search unsuccessful if Read Data != Address
                }
                break;
     case 0x20 : // no Ack received
                PCA9564_Write(I2CCON,0xD0 | CRX);              // I2CCON=11010xxx -> Stop condition
                break;
     }

    Address_Sent_Status = 0x00;
    Command_Sent_Status = 0x00;
  }
  return I2C_Address_Write;
}

//*****************************************************************************
// GPIO Interrupt Handling function
// One shot mode (through /INT) or
// permanent action detection (then Input PCA9554 Reg# polling)
//*****************************************************************************

 void GPIO_Interrupt_Handler(void)
{
  Message2.address   = PCA9554_WR;
  Message2.buf       = Buffer2;
  Message2.nrBytes   = 1;
  Buffer2[0]         = 0;                  // subaddress = 0

  Message3.address   = PCA9554_RD;
  Message3.buf       = Buffer3;
  Message3.nrBytes   = 1;                  // read one byte

  if (PCA9554_Int==0)                      // Action on pushbutton detected
  {
    I2C_WriteRepRead(&Message2,&Message3); // 1st read the PCA9554
    if (Buffer3[0] != 0xFF)
    {
      Snapshot_1 = Buffer3[0];             // load the 1st read data in a temp memory
    }
    InsertDelay(255);                      // Delay between 2 snapshots to detect if pushbutton is
    InsertDelay(255);                      // still pressed or has been released
    InsertDelay(255);
    I2C_WriteRepRead(&Message2,&Message3); // 2nd read the PCA9554
    Snapshot_2 = Buffer3[0];               // load the 2nd read data in a temp memory
    if (Snapshot_1 == Snapshot_2)          // Compare the 2 read data in the temp memories
    {
      Trigger_GPIO_Polling = 1;            // permanent push detected when 1st and 2nd readings equal
    }
    else
    {
      Trigger_GPIO_Polling = 0;            // single shot action when 1st and 2nd readings different
      Buffer3[0] = Snapshot_1;             // Buffer loaded again with the initial push value
    }
  }
  if (Trigger_GPIO_Polling == 1)           // Start Polling PCA9554 when permanent push detected
  {
    I2C_WriteRepRead(&Message2,&Message3);
  }
}

//*****************************************************************************
// Program 1: P89LV51 <--> PCA9564 <--> PCA9531
// Through Pushbuttons, BR0 and BR1 can be selected
// Once BR selected, PSC and PWM registers
// can be incremented / decremented
//*****************************************************************************

static  int BR_Select = 0;

void Blinker_Up_Down(void)
{
  idata BYTE Frequency_0;
  idata BYTE DutyCycle_0;
  idata BYTE Frequency_1;
  idata BYTE DutyCycle_1;

  LED2 = 1;                          // LD11     off
  LED3 = 0;                          // LD12     on --> PCA9531 programmed with default blinking rate
```

```
Message1.nrBytes = 7;             // Reset the PCA9531 to its default programmed values
Buffer1[0]       = 0x11;          // subaddress = 0x01
Buffer1[1]       = 0x80;          // default prescaler pwm0
Buffer1[2]       = 0x80;          // default duty cycle for pwm0
Buffer1[3]       = 0x80;          // default prescaler pwm1
Buffer1[4]       = 0x80;          // default duty cycle for pwm1
Buffer1[5]       = 0xAA;          // LD1 to LD4 blinking at BR0
Buffer1[6]       = 0xFF;          // LD5 to LD8 blinking at BR1
I2C_Write(&Message1);             // Program PCA9531 with default values  (7 bytes)

Frequency_0 = Buffer1[1];
DutyCycle_0 = Buffer1[2];
Frequency_1 = Buffer1[3];
DutyCycle_1 = Buffer1[4];

while (Buffer3[0]!=0x7F)          // Main loop as long as S8 (exit Program) has not been pushed
 {
 GPIO_Interrupt_Handler();        // Check if an action on pushbutton happened
 InsertDelay(100);                // Small delay for LED dimmer visual purpose (wait between device programming)

 Message1.nrBytes = 2;            // 2 bytes will be sent to the PCA9531 (pointer + register to be modified)
 if (Buffer3[0]!=0xFF)            // Execute the command associated with the action on pushbutton
 {
   switch (Buffer3[0])
   {
     case 0x7F : break;               // Exit Program 1- Push on S8 detected
     case 0xFB : if (BR_Select == 0)  // Action on pushbutton selecting Blinking Rate to be programmed
                 {
                   BR_Select = 1;     // Blinking Rate 1 selected to be modified - LD[4:8]
                   LED2 = 0;          // LD11 on
                   Buffer3[0] = 0xFF;
                   break;
                 }
                 else
                 {
                   BR_Select = 0      // Blinking Rate 0 selected to be modified - LD[1:4]
                   LED2 = 1;          // LD11 off
                   Buffer3[0] = 0xFF;
                 }
                 break;
     case 0xBF : LED3 = 1;                     // LD12 = off --> Default programming overwritten
                 Message2.address = PCA9531_WR; // Action on pushbutton - outputs to be either off or blinking
                 Message2.buf       = Buffer2;
                 Message2.nrBytes = 1;
                 Buffer2[0]         = 0x15;     // subaddress = 15
                 Message3.address = PCA9531_RD;
                 Message3.buf       = Buffer3;
                 Message3.nrBytes = 2;          // read 2 bytes
                 I2C_WriteRepRead(&Message2,&Message3);   // read output states of the PCA9531
                 if (BR_Select == 0)
                 {
                   if (Buffer3[0] == 0x00)
                   {
                     Buffer1[0]= 0x05;                    // subaddress = 0x05
                     Buffer1[1] = 0xAA;                   // LD[1:4] blinking at BR0
                     I2C_Write(&Message1);                // send new data to PCA9531 (2 bytes)
                   }
                   if (Buffer3[0] == 0xAA)
                   {
                     Buffer1[0]= 0x05;                    // subaddress = 0x05
                     Buffer1[1] = 0x00;                   // LD[1:4] off
                     I2C_Write(&Message1);                // send new data to PCA9531 (2 bytes)
                   }
                 }
                 if (BR_Select == 1)
                 {
                   if (Buffer3[1] == 0x00)
                   {
                     Buffer1[0]= 0x06;                    // subaddress = 0x05
                     Buffer1[1] = 0xFF;                   // LD[4:8] blinking at BR1
                     I2C_Write(&Message1);                // send new data to PCA9531 (2 bytes)
                   }
                   if (Buffer3[1] == 0xFF)
                   {
                     Buffer1[0]= 0x06;                    // subaddress = 0x05
                     Buffer1[1] = 0x00;                   // LD[4:8] off
                     I2C_Write(&Message1);                // send new data to PCA9531 (2 bytes)
                   }
                 }
                 break;
```

```
case 0xFE : LED3 = 1;                                  // LD12 = off --> Default programming overwritten
            if (BR_Select == 0 & Frequency_0 < 0xFF)
            {
              Buffer1[0] = 0x01;                       // subaddress = 0x01
              Frequency_0++;                           // increment prescaler 0
              Buffer1[1] = Frequency_0;
              I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
              Buffer3[0] = 0xFF;
            }
            if (BR_Select == 1 & Frequency_1 < 0xFF)
            {
              Buffer1[0] = 0x03;                       // subaddress = 0x03
              Frequency_1++;                           // increment prescaler 1
              Buffer1[1] = Frequency_1;
              I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
              Buffer3[0] = 0xFF;
            }
            break;
case 0xEF : LED3 = 1;                                  // LD12 = off --> Default programming overwritten
            if (BR_Select == 0 & Frequency_0 > 0x00)
            {
              Buffer1[0] = 0x01;                       // subaddress = 0x01
              Frequency_0--;                           // decrement prescaler 0
              Buffer1[1] = Frequency_0;
              I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
              Buffer3[0] = 0xFF;
            }
            if (BR_Select == 1 & Frequency_1 > 0x00)
            {
              Buffer1[0] = 0x03;                       // subaddress = 0x03
              Frequency_1--;                           // decrement prescaler 1
              Buffer1[1] = Frequency_1;
              I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
              Buffer3[0] = 0xFF;
            }
            break;
case 0xDF : LED3 = 1;                                  // LD12 = off --> Default programming overwritten
            if (BR_Select == 0 & DutyCycle_0 < 0xFF)
            {
              Buffer1[0] = 0x02;                       // subaddress = 0x02
              DutyCycle_0++;                           // increment pwm 0
              Buffer1[1] = DutyCycle_0;
              I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
              Buffer3[0] = 0xFF;
            }
            if (BR_Select == 1 & DutyCycle_1 < 0xFF)
            {
              Buffer1[0] = 0x04;                       // subaddress = 0x04
              DutyCycle_1++;                           // increment pwm 1
              Buffer1[1] = DutyCycle_1;
              I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
              Buffer3[0] = 0xFF;
            }
            break;
case 0xFD : LED3 = 1;                                  // LD12 = off --> Default programming overwritten
            if (BR_Select == 0 & DutyCycle_0 > 0x00)
            {
              Buffer1[0] = 0x02;                       // subaddress = 0x02
              DutyCycle_0--;                           // decrement pwm 0
              Buffer1[1] = DutyCycle_0;
              I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
              Buffer3[0] = 0xFF;
            }
            if (BR_Select == 1 & DutyCycle_1 > 0x00)
            {
              Buffer1[0] = 0x04;                   // subaddress = 0x04
              DutyCycle_1--;                       // decrement pwm 1
              Buffer1[1] = DutyCycle_1;
              I2C_Write(&Message1);                // send new data to PCA9531 (2 bytes)
              Buffer3[0] = 0xFF;
            }
            break;
case 0xF7 : LED3 = 0;                                  // LD12 = on --> PCA9531 with default blinking rate
            Message1.nrBytes = 7;                      // Reset the PCA9531 to its default programmed values
            Buffer1[0] = 0x11;                         // subaddress = 0x01
            Buffer1[1] = 0x80;                         // default prescaler pwm0
            Buffer1[2] = 0x80;                         // default duty cycle for pwm0
            Buffer1[3] = 0x80;                         // default prescaler pwm1
            Buffer1[4] = 0x80;                         // default duty cycle for pwm1
            Buffer1[5] = 0xAA;                         // LD1 to LD4 blinking at BR0
            Buffer1[6] = 0xFF;                         // LD5 to LD8 blinking at BR1
```

```
                I2C_Write(&Message1);                   // send new data to PCA9531 (7 bytes)
                Buffer3[0] = 0xFF;
                BR_Select  = 0;
                LED2 = 1;
                break;
    }
   }
  }
  Message1.nrBytes = 3;
  Buffer1[0]       = 0x15;    // subaddress = 0x15
  Buffer1[1]       = 0x00;    // PCA9531 all LEDs off when leaving Program 1
  Buffer1[2]       = 0x00;
  I2C_Write(&Message1);       // send new data to PCA9531 (3 bytes)
}

//*************************************************************************************************************
// Program 2 : P89LV51 <--> PCA9564 <--> PCF85116 <--> PCA9531
// Predefined blinking patterns are stored in the EEPROM (3 for this specific code)
// For each of them, command to be written are read from the EEPROM by the P89LV51 and PCA9531 is programmed
//
// Read the EEprom
// In order to get the data from the EE, read five bytes.
//   Buffer4[0]: Length
//   Buffer4[1]: Function
//   if Function=0 (function 0):
//     Buffer4[2]: Delay
//     Buffer4[3]: Address
//     Buffer4[4]: Data0
//     Buffer4[n]: Data...
//   if Function=1 (function 1):
//     Buffer4[2]: Delay
//     Buffer4[3]: Address
//     Buffer4[4]: Data0
//     Buffer4[n]: Data...
//   if Function=0x99 then end
//
// ReadEEprom function can be used im 2 different ways:
// Operation_EEprom = 0: determine the different functions by reading  the full memory and storing
// the 1st and last memory cell for each function
// Operation_EEprom = 1: for a specified function stored in the EEPROM,  a specific operation is performed
//     - Operation_Function = 0: a write operation is performed
//     - Operation_Function = 1: not defined
//     - Operation_Function = 2: not defined
//*************************************************************************************************************

short int EEPROM_Scan_Done;
short int MinPtrFound_0 = 0;
short int MinPtr_0;
short int MaxPtr_0;
short int MinPtrFound_1 = 0;
short int MinPtr_1;
short int MaxPtr_1;
short int MinPtrFound_2 = 0;
short int MinPtr_2;
short int MaxPtr_2;
short int MinPtrFound_3 = 0;
short int MinPtr_3;
short int MaxPtr_3;  // If more than 3 patterns in EEPROM, declare the additional PtrFound, MinPtr and  MaxPtr here

void ReadEEprom(short int MinEEPtr, short int MaxEEPtr,  int Operation_EEprom,  int Operation_Function)
{
  short int NextEEaddress;
  short int EndEEaddress;
  bit EndofMessages;
  int ExitLoop = 0;

  EndofMessages    = 0;
  NextEEaddress   = MinEEPtr;           // initialization of the min subaddress within the 2kB eeprom
  EndEEaddress    = MaxEEPtr;           // initialization of the max subaddress within the 2kB eeprom

  while (EndofMessages==0 & ExitLoop==0 & NextEEaddress <= MaxEEPtr)
  {
                                        // First we need to retrieve the data from the eeprom
    Message3.address = 0xA0 | ((NextEEaddress & 0x0700)>>7);  // Upper byte of NextEEaddress
    Buffer3[0] = NextEEaddress & 0xFF;                        // Lower byte of NextEEaddress = subaddress
    Message4.address = Message3.address | 0x01;               // Message 4 address is a read so set lsb
    Message3.nrBytes = 1;
```

```
                                // We're going to write one byte (subaddress)
if (Operation_EEprom == 0)          // EEPROM reading - Function (search min and max / function)
{
  EEPROM_Scan_Done = 0;
  Message4.nrBytes = 2;             // We're going to read 2 bytes (length of descriptor + Function)
  I2C_Write(&Message3);
  InsertDelay(2);
  I2C_Read(&Message4);
  switch (Buffer4[1])
  {
  case 0x00 : if (MinPtrFound_0 == 0)          // Min and Max search for function 0
              {
                MinPtr_0 = NextEEaddress;
                MinPtrFound_0 = 1;
              }
              if (MinPtrFound_0 == 1)
              {
                MaxPtr_0 = NextEEaddress;
              }
              break;
  case 0x01 : if (MinPtrFound_1 == 0)          // Min and Max search for function 1
              {
                MinPtr_1 = NextEEaddress;
                MinPtrFound_1 = 1;
              }
              if (MinPtrFound_1 == 1)
              {
                MaxPtr_1 = NextEEaddress;
              }
              break;
  case 0x02 : if (MinPtrFound_2 == 0)          // Min and Max search for function 2
              {
                MinPtr_2 = NextEEaddress;
                MinPtrFound_2 = 1;
              }
              if (MinPtrFound_2 == 1)
              {
                MaxPtr_2 = NextEEaddress;
              }
              break;
  case 0x03 : if (MinPtrFound_3 == 0)          // Min and Max search for function 3 (not defined in the EEPROM)
              {
                MinPtr_3 = NextEEaddress;
                MinPtrFound_3 = 1;
              }
              if (MinPtrFound_3 == 1)
              {
                MaxPtr_3 = NextEEaddress;
              }
              break;
  // If more patterns, add the additional "case" here
  case 0x99 : EndofMessages = 1;               // End of data in the EEPROM
              break;
  }
  EEPROM_Scan_Done = 1;
}
if (Operation_EEprom == 1)                     // EEPROM reading - Data in a specified function
{
  Message4.nrBytes = 1;                        // We're going to read one byte (length of descriptor)
  //I2C_WriteRepRead(&Message3,&Message4);
  I2C_Write(&Message3);                        // Read the EEPROM
  InsertDelay(2);
  I2C_Read(&Message4);
                                               // Buffer4[0] contains the length of the descriptor
  InsertDelay(2);
  Message4.nrBytes = Buffer4[0];               // First byte of data is the descriptor length
  //I2C_WriteRepRead(&Message3,&Message4);
  I2C_Write(&Message3);                        // Read the entire descriptor from the eeprom
  InsertDelay(2);
  I2C_Read(&Message4);
  InsertDelay(2);
                                               // Buffer4 contains the data from the eeprom
                                               // ------------------------------------------------------
                                               // At this point we have the data from the eeprom in Buffer4
```

```
    switch(Operation_Function)
    {
      case 0x00 : Message1.address = Buffer4[3];     // Write operation performed with the data read from EEPROM
                  Message1.nrBytes = Buffer4[0] - 4;
                  Message1.buf    = Buffer4 + 4;
                  I2C_Write(&Message1);
                  InsertDelay(Buffer4[2]);          // Programmable delay in ms (up to 255 ms i.e. 0xFF)
                  break;

      case 0x01 : break;                            // Not defined
      case 0x02 : break;                            // Not defined
    }
  }
  NextEEaddress = NextEEaddress + Buffer4[0];       // The length of the descriptor is in Buffer4[0]
                                                    // so we can calculate the next eeprom address+subaddress
  GPIO_Interrupt_Handler();  // Check if an action on S8 happened (to exit the program)or S5 (to change program)
  if (Buffer3[0]==0x7F | Buffer3[0]==0xEF)          // Leave the loop when S5 or S8  pushed
  {
    ExitLoop = 1;
  }
                                                    // switch
 }                                                  // while(EndofMessage or exit program)
}


void Preset_Patterns_PCA9531(void)
{

 short int PatternCounter = 0;

 LED2 = 1;                                          // LD11     off
 LED3 = 1;                                          // LD12     off
 EEPROM_Scan_Done = 0;                              // EEPROM scan not done when entering the program

 while (Buffer3[0]!=0x7F)                           // Main loop as long as S8 (exit Program) has not been pushed
 {
  GPIO_Interrupt_Handler();                         // Check if an action on pushbutton happened

  if (Buffer3[0]!=0xFF)                             // execute the command associated with the action on pushbutton
  {
    switch (Buffer3[0])
    {
      case 0x7F : break;                            // Exit Program 3- S8 pushed
      case 0xF7 : ReadEEprom(0x000,0x7FF,0,0);      // Read EEPROM structure (function organization)
                  LED3 = 0;                         // LD12 on when the EEPROM structure has been analyzed
                  Buffer3[0]  = 0xFF;
                  break;
      case 0xEF : if ( EEPROM_Scan_Done == 0) break; // Nothing can be done until the EEPROM SCAN has been done
                  if (PatternCounter < 3)           // Pattern change pushbutton activated
                  {             // If more than 3 patterns, increase the number above to the adequate number
                    PatternCounter++;               // Program selection incremented
                  }
                  else
                  {
                    PatternCounter = 1;             // Program selection back to 1
                  }
                  Message4.address = LPC932_WR;
                  Message4.buf     = Buffer4;
                  Message4.nrBytes = 2;
                  Buffer4[0] = 0x00;                            // Command byte to program LEDs
                  if (PatternCounter ==1) Buffer4[1] = 0xFE;    // LD13 on (Pattern 1)
                  if (PatternCounter ==2) Buffer4[1] = 0xFD;    // LD14 on (Pattern 2)
                  if (PatternCounter ==3) Buffer4[1] = 0xFB;    // LD15 on (Pattern 3)
                  // If more than 3 patterns, add the required "if"
                  I2C_Write(&Message4);                         // Program LPC932 to display the pattern number
                  Buffer3[0]  = 0xFF;
                  break;
    }
    GPIO_Interrupt_Handler();                                   // Check if an action on pushbutton happened
    if (PatternCounter == 1 & EEPROM_Scan_Done == 1)ReadEEprom(MinPtr_0,MaxPtr_0,1,0);
    if (PatternCounter == 2 & EEPROM_Scan_Done == 1)ReadEEprom(MinPtr_1,MaxPtr_1,1,0);
    if (PatternCounter == 3 & EEPROM_Scan_Done == 1)ReadEEprom(MinPtr_2,MaxPtr_2,1,0);
    // If more than 3 patterns, add the required "if"
  }
 }
 Message1.address = PCA9531_WR;
 Message1.nrBytes = 3;
 Buffer1[0] = 0x15;                                 // subaddress = 0x15
 Buffer1[1] = 0x00;                                 // PCA9531 all LEDs off when leaving Program 1
 Buffer1[2] = 0x00;
```

```
  I2C_Write(&Message1);                              // send new data to PCA9531 (3 bytes)
  Buffer4[0] = 0x00;                                 // Command to program  LEDs
  Buffer4[1] = 0xFF;                                 // LPC932 all LEDs off when leaving Program 3
  I2C_Write(&Message4);
}

//*************************************************************************
// Program 3 : P89LV51 <--> PCA9564 <--> P89LPC932
// Through Pushbuttons, Byte to be sent to LPC932 can be programmed
// Once done, a pushbutton sends the LPC932 I2C address + programmed byte
//*************************************************************************

static bdata BYTE LS0 = 0;
sbit LS0_0 = LS0^0;
sbit LS0_1 = LS0^1;
sbit LS0_2 = LS0^2;
sbit LS0_3 = LS0^3;
sbit LS0_4 = LS0^4;
sbit LS0_5 = LS0^5;
sbit LS0_6 = LS0^6;
sbit LS0_7 = LS0^7;

static bdata BYTE LS1 = 0;
sbit LS1_0 = LS1^0;
sbit LS1_1 = LS1^1;
sbit LS1_2 = LS1^2;
sbit LS1_3 = LS1^3;
sbit LS1_4 = LS1^4;
sbit LS1_5 = LS1^5;
sbit LS1_6 = LS1^6;
sbit LS1_7 = LS1^7;

static bdata BYTE DataByteLPC932 = 0xFF;
sbit DataByteLPC932_0 = DataByteLPC932^0;
sbit DataByteLPC932_1 = DataByteLPC932^1;
sbit DataByteLPC932_2 = DataByteLPC932^2;
sbit DataByteLPC932_3 = DataByteLPC932^3;
sbit DataByteLPC932_4 = DataByteLPC932^4;
sbit DataByteLPC932_5 = DataByteLPC932^5;
sbit DataByteLPC932_6 = DataByteLPC932^6;
sbit DataByteLPC932_7 = DataByteLPC932^7;

void LV51_LPC932(void)
{
  int BitCounter = 1;
  int ValueToBeChanged = 0;

  Init_LPC932();                                 // Initialization of LPC932 (narrowed search address + LEDs off
  Message1.address = PCA9531_WR;                 // Initialization of PCA9531
  Message1.buf     = Buffer1;
  Message1.nrBytes = 7;
  Buffer1[0]       = 0x11;                       // autoincrement + register 1
  Buffer1[1]       = 0x80;                       // default prescaler pwm0
  Buffer1[2]       = 0x80;                       // default duty cycle for pwm0
  Buffer1[3]       = 0x80;                       // default prescaler pwm1
  Buffer1[4]       = 0x80;                       // default duty cycle for pwm1
  Buffer1[5]       = 0x00;                       // LD1 to LD4 off
  Buffer1[6]       = 0x00;                       // LD5 to LD8 off
  I2C_Write(&Message1);                          // LD[1:8] off
  LED2 = 1;                                      // LD11 is    off
  LED3 = 1;                                      // LD12 is    off
  DataByteLPC932 = 0xFF;
  LS0 = 0;
  LS1 = 0;

  while (Buffer3[0]!=0x7F)                         // Main loop as long as S8 (exit Program) has not been pushed
  {
    GPIO_Interrupt_Handler();                      // Check if an action on pushbutton happened

    if (Buffer3[0]!=0xFF)                          // execute the command associated with the action on pushbutton
    {
      switch (Buffer3[0])
      {
        case 0x7F : break;                         // Exit Program 3- S8 pushed
        case 0xF7 : BitCounter = 1;                // Reset programming (all LEDs are off) - S4 pushed
                    LS0 = 0x00;
                    LS1 = 0x00;
                    Message1.nrBytes = 3;
                    Buffer1[0] = 0x15;             // subaddress = 0x05
                    Buffer1[1] = 0x00;             // LS0 = 0x00 - All LEDs off
                    Buffer1[2] = 0x00;             // LS1 = 0x00 - All LEDs off
```

```
            I2C_Write(&Message1);          // Program PCA9531 (3 bytes)
            DataByteLPC932 = 0xFF;
            Buffer4[0] = 0x00;             // Command byte to program LEDs
            Buffer4[1] = DataByteLPC932;   // LPC932 all LEDs off
            I2C_Write(&Message4);          // Program LPC932 (2 bytes)
            Buffer3[0] = 0xFF;
            break;
case 0xEF : if (BitCounter < 8)            // increment programming position - S1 pushed
            {
              BitCounter++;
            }
            else
            {
              BitCounter = 1;
            }
            Buffer3[0] = 0xFF;
            break;
case 0xFE : if (BitCounter != 1)           // decrement programming position - S5 pushed
            {
              BitCounter--;
            }
            else
            {
              BitCounter = 8;
            }
            Buffer3[0] = 0xFF;
            break;
case 0xFD : ValueToBeChanged = 1;          // S2 pushed - Change polarity of the current position
            switch (BitCounter)
            {
              Message1.nrBytes = 2;
              case 1 : if (ValueToBeChanged == 1)     // Bit 0
                       {
                         if (DataByteLPC932_0 == 0)
                         {
                           LS0_0 = 0;
                           LS0_1 = 0;
                           DataByteLPC932_0 = 1;
                           ValueToBeChanged = 0;
                         }
                         else
                         {
                           LS0_0 = 1;
                           LS0_1 = 0;
                           DataByteLPC932_0 = 0;
                           ValueToBeChanged = 0;
                         }
                         Message1.nrBytes = 2;
                         Buffer1[0] = 0x15;           // subaddress = 0x05
                         Buffer1[1] = LS0;            // LED Selector programming
                         I2C_Write(&Message1);
                       }
                       break;
              case 2 : if (ValueToBeChanged == 1)     // Bit 1
                       {
                         if (DataByteLPC932_1 == 0)
                         {
                           LS0_2 = 0;
                           LS0_3 = 0;
                           DataByteLPC932_1 = 1;
                           ValueToBeChanged = 0;
                         }
                         else
                         {
                           LS0_2 = 1;
                           LS0_3 = 0;
                           DataByteLPC932_1 = 0;
                           ValueToBeChanged = 0;
                         }
                         Message1.nrBytes = 2;
                         Buffer1[0] = 0x15;           // subaddress = 0x05
                         Buffer1[1] = LS0;            // LED Selector programming
                         I2C_Write(&Message1);
                       }
                       break;
```

```
case 3 : if (ValueToBeChanged == 1)     // Bit 2
         {
          if (DataByteLPC932_2 == 0)
          {
           LS0_4 = 0;
           LS0_5 = 0;
           DataByteLPC932_2 = 1;
           ValueToBeChanged = 0;
          }
          else
          {
           LS0_4 = 1;
           LS0_5 = 0;
           DataByteLPC932_2 = 0;
           ValueToBeChanged = 0;
          }
          Message1.nrBytes = 2;
          Buffer1[0] = 0x15;            // subaddress = 0x05
          Buffer1[1] = LS0;             // LED Selector programming
          I2C_Write(&Message1);
         }
         break;
case 4 : if (ValueToBeChanged == 1)     // Bit 3
         {
          if (DataByteLPC932_3 == 0)
          {
           LS0_6 = 0;
           LS0_7 = 0;
           DataByteLPC932_3 = 1;
           ValueToBeChanged = 0;
          }
          else
          {
           LS0_6 = 1;
           LS0_7 = 0;
           DataByteLPC932_3 = 0;
           ValueToBeChanged = 0;
          }
          Message1.nrBytes = 2;
          Buffer1[0] = 0x15;            // subaddress = 0x05
          Buffer1[1] = LS0;             // LED Selector programming
          I2C_Write(&Message1);
         }
         break;
case 5 : if (ValueToBeChanged == 1)     // Bit 4
         {
          if (DataByteLPC932_4 == 0)
          {
           LS1_0 = 0;
           LS1_1 = 0;
           DataByteLPC932_4 = 1;
           ValueToBeChanged = 0;
          }
          else
          {
           LS1_0 = 1;
           LS1_1 = 0;
           DataByteLPC932_4 = 0;
           ValueToBeChanged = 0;
          }
          Message1.nrBytes = 2;
          Buffer1[0] = 0x16;            // subaddress = 0x06
          Buffer1[1] = LS1;             // LED Selector programming
          I2C_Write(&Message1);
         }
         break;
case 6 : if (ValueToBeChanged == 1)     // Bit 5
         {
          if (DataByteLPC932_5 == 0)
          {
           LS1_2 = 0;
           LS1_3 = 0;
           DataByteLPC932_5 = 1;
           ValueToBeChanged = 0;
          }
          else
          {
           LS1_2 = 1;
           LS1_3 = 0;
           DataByteLPC932_5 = 0;
```

```
                        ValueToBeChanged = 0;
                      }
                      Message1.nrBytes = 2;
                      Buffer1[0] = 0x16;          // subaddress = 0x06
                      Buffer1[1] = LS1;           // LED Selector programming
                      I2C_Write(&Message1);
                    }
                    break;
            case 7 : if (ValueToBeChanged == 1)   // Bit 6
                    {
                      if (DataByteLPC932_6 == 0)
                      {
                        LS1_4 = 0;
                        LS1_5 = 0;
                        DataByteLPC932_6 = 1;
                        ValueToBeChanged = 0;
                      }
                      else
                      {
                        LS1_4 = 1;
                        LS1_5 = 0;
                        DataByteLPC932_6 = 0;
                        ValueToBeChanged = 0;
                      }
                      Message1.nrBytes = 2;
                      Buffer1[0] = 0x16;          // subaddress = 0x06
                      Buffer1[1] = LS1;           // LED Selector programming
                      I2C_Write(&Message1);
                    }
                    break;
            case 8 : if (ValueToBeChanged == 1)   // Bit 7
                    {
                      if (DataByteLPC932_7 == 0)
                        {
                        LS1_6 = 0;
                        LS1_7 = 0;
                        DataByteLPC932_7 = 1;
                        ValueToBeChanged = 0;
                      }
                      else
                      {
                        LS1_6 = 1;
                        LS1_7 = 0;
                        DataByteLPC932_7 = 0;
                        ValueToBeChanged = 0;
                      }
                      Message1.nrBytes = 2;
                      Buffer1[0] = 0x16;          // subaddress = 0x06
                      Buffer1[1] = LS1;           // LED Selector programming
                      I2C_Write(&Message1);
                    }
                    break;
          }
          Buffer3[0] = 0xFF;
          break;
    case 0xFB : Buffer4[0] = 0x00;                // Command byte to program LEDS
          Buffer4[1] = DataByteLPC932;            // Program LPC932 with the programmed byte - S3 pushed
          I2C_Write(&Message4);                   // LPC932 programmed (2 bytes)
          Buffer3[0] = 0xFF;
          break;
   }
  }
 }
 Message1.nrBytes = 3;
 Buffer1[0]       = 0x15;          // subaddress = 0x15
 Buffer1[1]       = 0x00;          // PCA9531 all LEDs off when leaving Program 3
 Buffer1[2]       = 0x00;
 I2C_Write(&Message1);             // Program PCA9531 (3 bytes)
 Buffer4[0]       = 0x00;          // Command to program  LEDs
 Buffer4[1]       = 0xFF;          // LPC932 all LEDs off when leaving Program 3
 I2C_Write(&Message4);
}
```

```
//*****************************************************************************
// Program 4:P89LV51 <--> PCA9564 <--> P89LPC932
// Initiates an automatic I2C slave address search in order
// to find P89LPC932's I2C slave address
//*****************************************************************************

void I2C_Address_Search(void)
{
  bdata BYTE I2C_Address;

  LED2 = 1;                                    // LD11 is    off
  LED3 = 1;                                    // LD12 is    off

  while (Buffer3[0]!=0x7F)                     // Main loop as long as S8 (exit Program) has not been pushed
  {
    GPIO_Interrupt_Handler();                  // Check if an action on pushbutton happened

    if (Buffer3[0]!=0xFF)                      // execute the command associated with the action on pushbutton
    {
      switch (Buffer3[0])
      {
        case 0x7F : break;                     // Exit Program 3- S8 pushed
        case 0xFE : I2C_Address = Search_Routine(0x00,0xFE);
                    if (Search_Successful == 0)
                    {
                      LED2 = 1;
                      LED3 = 0;                        // Search failed (the all I2C address range has been checked
                    }
                    Buffer3[0] = 0xFF;
                    if (Search_Successful == 1)
                    {
                      LED2 = 0;                        // Search successful
                      LED3 = 1;
                      LPC932_WR = I2C_Address;
                      LPC932_RD = LPC932_WR + 1;
                      Message4.address = LPC932_WR;
                      Message4.buf     = Buffer4;
                      Message4.nrBytes = 2;
                      Buffer4[0] = 0x00;               // Command byte to program the LEDs
                      Buffer4[1] = ~LPC932_WR;         // LD[13:20] display the found address
                      I2C_Write(&Message4);            // Program LPC932 (2 bytes)
                    }
                    break;
        case 0xFD : Search_Successful = 0;  // Reset of the previous search, LPC932 scans and stores its I2C address
                    LED2 = 1;
                    LED3 = 1;
                    Message4.address = LPC932_WR;
                    Message4.buf     = Buffer4;
                    Message4.nrBytes = 2;
                    Buffer4[0] = 0x00;               // Command byte to program the LEDs
                    Buffer4[1] = 0xFF;               // LD[13:20] off
                    I2C_Write(&Message4);            // Program LPC932 (2 bytes)
                    Message4.nrBytes = 1;
                    Buffer4[0] = 0xEE;               // Command byte to scan and store the new I2C address
                    I2C_Write(&Message4);            // Program LPC932 (1 byte)
                    Buffer3[0] = 0xFF;
                    break;
      }
    }
  }
  Message4.nrBytes = 2;
  Buffer4[0] = 0x00;           // Command to program  LEDs
  Buffer4[1] = 0xFF;           // LPC932 all LEDs off when leaving Program 3
  I2C_Write(&Message4);
}

void Init_LPC932(void)
{
  LPC932_WR = Search_Routine(0xDE,0xEE);
  LPC932_RD = LPC932_WR + 1;
  Message4.address = LPC932_WR;
  Message4.buf     = Buffer4;
  Message4.nrBytes = 2;
  Buffer4[0] = 0x00;         // Command byte to program LEDs
  Buffer4[1] = 0xFF;         // LPC932 all LEDs off
  I2C_Write(&Message4);      // LD[13:20] off
}
```

I2CDRIVR.H

```
//*****************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   i2cdriver.h
// Created:     June 2, 2003
// Modified:    June 9, 2003
// Revision:    1.00
//
//*****************************************************************************

#define ST_IDLE       0
#define ST_SENDING    1
#define ST_AWAIT_ACK  2
#define ST_RECEIVING  3
#define ST_RECV_LAST  4

//#define CR0_MASK    0x01
//#define CR1_MASK    0x02
//#define CR2_MASK    0x04
//#define SI_MASK     0x08
//#define STO_MASK    0x10
//#define STA_MASK    0x20
//#define ENSIO_MASK  0x40
//#define AA_MASK    0x80

extern BYTE master;
extern BYTE intMask;

void MainStateHandler(void);
```

I2CDRIVR.C

```
//*****************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   i2cdriver.c
// Created:     June 2, 2003
// Modified:    June 10, 2003
// Revision:    1.00
//
//*****************************************************************************

#include "i2cexprt.h"
#include "i2cdrivr.h"
#include <REG51RX.H>
#include "PCA9564sys.h"

static void NoInitErrorProc(void);

void (*masterProc)(void) = NoInitErrorProc;
void (*slaveProc)(void)  = NoInitErrorProc;

BYTE master;
BYTE intMask;
```

```
/* ************************************************************************
* Input(s):       none.
* Output(s):      none.
* Returns:        none.
* Description:    ERROR: Master or slave handler called while not initialized
************************************************************************/

static void NoInitErrorProc(void)
{
    PCA9564_Write(I2CCON, 0x40);
}

/************************************************************************
 * Input(s):       none.
 * Output(s):      none.
 * Returns:        none.
 * Description:    Main event handler for I2C.
 ************************************************************************/

void MainStateHandler(void)
{
    if (master)
        masterProc();               // Master Mode
    else
        slaveProc();                // Slave Mode
}

/* ************************************************************************
* Input(s):        none.
* Output(s):      none.
* Returns:        none.
* Description:    Interrupt handler for I2C.
************************************************************************/

void I2C_Interrupt(void) interrupt 5
{
    MainStateHandler();
}
```

I2CMASTR.h

```
//************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   i2cmaster.h
// Created:     June 2, 2003
// Modified:    June 9, 2003
// Revision:    1.00
//
//************************************************************************

void I2C_Transfer(I2C_TRANSFER *p, void (*proc)(BYTE, BYTE));
```

I2CMASTR.C

```
//***********************************************************************
//
//                 P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   i2cmaster.c
// Created:     June 2, 2003
// Modified:    June 10, 2003
// Revision:    1.00
//
//***********************************************************************

#include <REG51RX.H>
#include "i2cexprt.h"
#include "i2cdrivr.h"
#include "i2cmastr.h"
#include "PCA9564sys.h"

extern void (*masterProc)();                        // Handle Master Transfer action
//extern void (*slaveProc)();                       // Handle Slave Transfer action

static I2C_TRANSFER *tfr;                           // Pointer to active transfer block
static I2C_MESSAGE  *msg;                           // Pointer to active message block

static void (code *readyProc)(BYTE status,BYTE);    //  proc. to call if transfer ended
static BYTE msgCount;                               // Number of messages sent
static BYTE dataCount;                              // nr of bytes of current message
static BYTE state;                                  // state of the I2C driver
extern BYTE drvStatus;
unsigned char CRX;                                  // I2C frequency selector


/*************************************************************************
* Input(s):      status -> status of the driver.
* Output(s):   None.
* Returns:    None.
* Description: Generate a stop condition.
*************************************************************************/

/*static void GenerateStop(BYTE status)
{
    PCA9564_Write(I2CCON,0xD0 | CR);
    master  = FALSE;
    state   = ST_IDLE;
    readyProc(status, msgCount);                    // Signal driver is finished
}*/

/*************************************************************************
* Input(s):      None.
* Output(s):   None.
* Returns:     None.
* Description: Master mode state handler for I2C-bus.
*************************************************************************/

//     +-------------------------------------------------------+
//     | I2CCON  | AA | ENSIO | STA | STO | SI | CR2 | CR1 | CR0 |
//     +-------------------------------------------------------+

static void HandleMasterState(void)
{
  unsigned char PCA9564_Status;

  PCA9564_Status = PCA9564_Read(I2CSTA);

  switch (PCA9564_Status)
  {
    case 0x08 : // (re)start condition
    case 0x10 : PCA9564_Write(I2CDAT,msg->address);
                PCA9564_Write(I2CCON,0xC0 | CRX);             // clear SI bit to send address
                break;
    case 0x18 : PCA9564_Write(I2CDAT,msg->buf[dataCount++]);  // send next data byte
                PCA9564_Write(I2CCON,0xC0 | CRX);             // I2CCON=11000xxx
                break;
```

```
    case 0x20 : // no Ack received
                PCA9564_Write(I2CCON,0xD0 | CRX);             // I2CCON=11010xxx -> Stop condition
                drvStatus = I2C_ERROR;
                break;
    case 0x28 : // ack received
                if (dataCount < msg->nrBytes)
                {
                  PCA9564_Write(I2CDAT,msg->buf[dataCount++]);
                  PCA9564_Write(I2CCON,0xC0 | CRX);           // I2CCON=11000xxx -> release interrupt
                }
                else
                {
                  if (msgCount < tfr->nrMessages)
                  {
                    dataCount = 0;
                    msg = tfr->p_message[msgCount++];
                    PCA9564_Write(I2CDAT,msg->address);
                    PCA9564_Write(I2CCON,0xE0 | CRX);         // I2CCON=11100xxx = start
                  }
                  else
                  {
                    PCA9564_Write(I2CCON,0xD0 | CRX);         // I2CCON=11010xxx
                    drvStatus = I2C_OK;
                  }
                }  // if
                break;
    case 0x30 : // no ACK for data byte
                PCA9564_Write(I2CCON,0xD0 | CRX);             // I2CCON=11010xxx -> stop condition
                drvStatus = I2C_ERROR;
                break;
    case 0x38 : // arbitration lost -> not addressed as slave
                PCA9564_Write(I2CCON,0xE0 | CRX);             // I2CCON=11100xxx -> send start again
                drvStatus = I2C_ARBITRATION_LOST;
                break;

    // MASTER RECEIVER FUNCTIONS
    case 0x40 : // ACK for slave address + R
                if (msg->nrBytes>1)
                {
                  PCA9564_Write(I2CCON,0xC0 | CRX);           // I2CCON=11000xxx -> acknowledge byte
                }
                else
                {
                  PCA9564_Write(I2CCON,0x40 | CRX);           // I2CCON=01000xxx -> return NACK
                }  // if
                break;
    case 0x48 : // no ACK for slave address + R
                PCA9564_Write(I2CCON,0xD0 | CRX);             // I2CCON=11010xxx -> send stop
                drvStatus = I2C_ERROR;
                break;
    case 0x50 : // ACK for data byte
                msg->buf[dataCount++] = PCA9564_Read(I2CDAT);
                if (dataCount + 1 < msg->nrBytes)
                {
                  PCA9564_Write(I2CCON,0xC0 | CRX);           // I2CCON=11000xxx
                }
                else
                {
                  PCA9564_Write(I2CCON,0x40 | CRX);           // I2CCON=01000xxx
                }
                break;
    case 0x58 : // no ACK for data byte
                msg->buf[dataCount++] = PCA9564_Read(I2CDAT);
                PCA9564_Write(I2CCON,0xD0 | CRX);             // I2CCON=11010xxx -> send Stop
                drvStatus = I2C_OK;
                break;
    default :   // undefined error
                PCA9564_Write(I2CCON,0xD0 | CRX);             // I2CCON=11000xxx -> send stop
                drvStatus = I2C_ERROR;
                break;
  }    // switch - PCA9564_Status
}      // i2c_isr
```

```
/*****************************************************************************
* Input(s):        p      address of I2C transfer parameter block.
*                  proc   procedure to call when transfer completed,
*                         with the driver status passed as parameter.
* Output(s):     None.
* Returns:       None.
* Description:   Start an I2C transfer, containing 1 or more messages. The application must
*                leave the transfer parameter block untouched until the ready procedure is called.
*****************************************************************************/

void I2C_Transfer(I2C_TRANSFER *p, void (*proc)(BYTE status, BYTE msgsDone))
{
    tfr       = p;
    readyProc = proc;
    msgCount  = 0;
    dataCount = 0;
    master    = TRUE;
    msg = tfr->p_message[msgCount++];
    state     = (msg->address & 1) ? ST_AWAIT_ACK : ST_SENDING;
    PCA9564_Write(I2CCON,0xE0 | CRX);        // 1110 0xxx -> generate Start
}

/* *********************************************************************
* Input(s):      speed      clock register value for bus speed.
* Output(s):     None.
* Returns:       None.
* Description:   Initialize the PCA9564 as I2C-bus master.
*********************************************************************/

void I2C_InitializeMaster(BYTE speed)
{
  int i;

    state     = ST_IDLE;
    readyProc = 0;                      // Null pointer
    masterProc = HandleMasterState;     // Set pointer to correct proc.
    PCA9564_Write(I2CADR,0xFE);         // own slave address
    CRX = speed;                        // I2C Frequency
    PCA9564_Write(I2CCON,0xC0 | CRX);   // 1100 0xxx -> Set to slave receiver
    for (i=0;i<1000;i++)  ;
    master  = FALSE;
}
```

I2CINTFC.C

```
//***************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   i2cintfc.c
// Created:     June 2, 2003
// Modified:    June 10, 2003
// Revision:    1.00
//
//***************************************************************************

#include <REG51RX.H>
#include "i2cexprt.h"
#include "i2cdrivr.h"
#include "i2cmastr.h"
#include "PCA9564sys.h"

BYTE drvStatus;                                 // Status returned by driver

static I2C_MESSAGE   *p_iicMsg[2];              // pointer to an array of (2) I2C messages
static I2C_TRANSFER  iicTfr;

/***************************************************************************
* Input(s):     status      Status of the driver at completion time
*               msgsDone    Number of messages completed by the driver
* Output(s):    None.
* Returns:      None.
* Description:  Signal the completion of an I2C transfer. This function is
*               passed (as parameter) to the driver and called by the
*               drivers state handler (!).
***************************************************************************/

static void I2cReady(BYTE status, BYTE msgsDone)
{
    drvStatus = status;
}

/* ************************************************************************
* Input(s):     None.
* Output(s):    status field of I2C_TRANSFER contains the driver status:
*               I2C_OK        Transfer was successful.
*               I2C_TIME_OUT  Timeout occurred
*               Otherwise     Some error occurred.
* Returns:      None.
* Description:  Start I2C transfer and wait (with timeout) until the
*               driver has completed the transfer(s).
************************************************************************** */

 static void StartTransfer(void)
{
  LONG timeOut;
  BYTE retries = 0;

  do
  {
   drvStatus = I2C_BUSY;
   I2C_Transfer(&iicTfr, I2cReady);
   timeOut = 0;
   while (drvStatus == I2C_BUSY)
   {
     if (++timeOut > 60000)
       drvStatus = I2C_TIME_OUT;
     if (PCA9564_Read(I2CCON) & 0x08)    // wait until SI bit is set
       MainStateHandler();
   }
   if (retries == 6)
   {
     drvStatus = I2C_RETRIES;            // too many retries
   }
   else
      retries++;
  } while ((drvStatus != I2C_OK) & (drvStatus!=I2C_RETRIES));
}
```

```
/*****************************************************************************
* Input(s):       msg     I2C message
* Returns:        None.
* Description:    Read a message from a slave device.
* PROTOCOL:       <S><SlvA><R><A><D1><A> ... <Dnum><N><P>
*****************************************************************************/

void I2C_Read(I2C_MESSAGE *msg)
{
    iicTfr.nrMessages = 1;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0]     = msg;

    StartTransfer();
}

/*****************************************************************************
* Input(s):       msg     I2C message
* Returns:        None.
* Description:    Write a message to a slave device.
* PROTOCOL:       <S><SlvA><W><A><D1><A> ... <Dnum><N><P>
*****************************************************************************/

void I2C_Write(I2C_MESSAGE *msg)
{
    iicTfr.nrMessages = 1;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg;

    StartTransfer();
}

/*****************************************************************************
* Input(s):       msg1     first  I2C message
*                 msg2     second I2C message
* Returns:        None.
* Description:    A message is sent and received to/from two different
*                 slave devices, separated by a repeat start condition.
* PROTOCOL:       <S><Slv1A><W><A><D1><A>...<Dnum1><A>
*                 <S><Slv2A><R><A><D1><A>...<Dnum2><N><P>
*****************************************************************************/

void I2C_WriteRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
{
    iicTfr.nrMessages = 2;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg1;
    p_iicMsg[1] = msg2;

    StartTransfer();
}

/*****************************************************************************
 * Input(s):       msg1     first I2C message
 *                 msg2     second I2C message
 * Returns:        None.
 * Description:    Writes two messages to different slave devices separated
 *                 by a repeated start condition.
 * PROTOCOL:       <S><Slv1A><W><A><D1><A>...<Dnum1><A>
 *                 <S><Slv2A><W><A><D1><A>...<Dnum2><A><P>
 *****************************************************************************/

void I2C_WriteRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
{
    iicTfr.nrMessages = 2;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg1;
    p_iicMsg[1] = msg2;

    StartTransfer();
}
```

PCA9564sys.h

```
//*****************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   PCA9564sys.h
// Created:     June 2, 2003
// Modified:    June 4, 2003
// Revision:    1.00
//
//*****************************************************************************

#ifndef __PCA9564SYS_H__
#define __PCA9564SYS_H__

#define MCU_COMMAND 0xFF         // dummy address
#define I2CSTA    0x00           // PCA9564 Status Register
#define I2CTO     0x00           // PCA9564 Timeout Register
#define I2CDAT    0x01           // PCA9564 Data Register
#define I2CADR    0x02           // PCA9564 Address Register
#define I2CCON    0x03           // PCA9564 Control Register

// I2CCON = AA + ENSIO + STA + STO + SI + CR2 + CR1 + CR0
#define AA_ENSIO_STA_NOTSTO_NOTSI     0xE0          // Generate Start
#define AA_ENSIO_NOTSTA_STO_NOTSI     0xD0          // Generate Stop
#define AA_ENSIO_NOTSTA_NOTSTO_NOTSI  0xC0          // Release bus and Ack + set to slave receiver
#define NOTAA_ENSIO_NOTSTA_NOTSTO_NOTSI 0x40        // Release bus and NOT Ack

void PCA9564_Write(unsigned char Reg, unsigned char val);
unsigned char PCA9564_Read(unsigned char Reg);

#endif
```

PCA9564sys.c

```
//*****************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name:   PCA9564sys.C
// Created:     June 2, 2003
// Modified:    June 10, 2003
// Revision:    1.00
//
//*****************************************************************************

#include <REG51RX.H>           // special function register declarations
//#include "BasicTyp.h"
#include "PCA9564sys.h"

sbit A0     = P2^0;
sbit A1     = P2^1;

sbit Reset9564 = P3^4;

// +---------------------------------------------+
// | PCA9564 register commands
// +---------------------------------------------+

void PCA9564_Write(unsigned char Reg, unsigned char val)
{
  A0 = Reg & 0x01;
  A1 = Reg & 0x02;
  AUXR = 0x02;                      // Access external memory, emit/don't emit ALE --> emit when using emulator
  *((unsigned char pdata *)MCU_COMMAND) = val;
  AUXR = 0x00;                      // Access internal memory, emit ALE when using emulator
}
```

```
unsigned char PCA9564_Read(unsigned char Reg)
{
  unsigned char RegData;
  int i;

  A0 = Reg & 0x01;
  A1 = Reg & 0x02;
  AUXR = 0x02;                    // Access external memory, emit ALE when using emulator
  for (i=0;i<10;i++);             // add small delay due to rise time issues on demoboard
  RegData = *((unsigned char pdata *)MCU_COMMAND);
  AUXR = 0x00;                    // Access internal memory, emit ALE when using emulator
  return RegData;
}
```

Interrupts.c

```
//***************************************************************************
//
//                  P H I L I P S   P R O P R I E T A R Y
//
//           COPYRIGHT (c)   2003 BY PHILIPS SEMICONDUCTORS
//                     --  ALL RIGHTS RESERVED  --
//
// File Name: Interrupts.c             --  Only for reference (not used)
// Created:   June 2, 2003
// Modified:  November 07, 2003
// Revision:  1.00
//
//***************************************************************************


void ExternalInt0(void) interrupt 0
{

}

void Timer0(void) interrupt 1
{

}

void ExternalInt1(void) interrupt 2
{

}

void Timer1(void) interrupt 3
{

}
```

Appendix 2: P89LPC932 Microcontroller Source Code – Rev 1.0

Main.c

```
//***************************************************************************
//*  main.c
//*  Date :          July 2003
//*  Discription :   Using I2Cslave code to interact with the
//*                  PCA9564 on the I2C-bus
//***************************************************************************
#include <Reg932.h>
#include "ua_exprt.h"
//***************************************************************************
//* Definitions
//***************************************************************************
typedef unsigned char   BYTE;
typedef unsigned short  WORD;
//***************************************************************************
//* Functions
//***************************************************************************
void init(void);
//***************************************************************************
//* init()
//* Input(s) :     none.
//* Returns :      none.
//* Description :  initialization of P89LPC903
//***************************************************************************
void init(void)
{
  P0M1 = 0x00;                  // P0 in Quasi bi mode
  P1M1 = 0x0C;                  // P1 in Quasi bi mode
  P1M2 = 0x0C;                  // P1.2 P13 open drain
  P2M1 = 0x00;                  // P2 in Quasi bi mode
}
//***************************************************************************
//* main()
//* Input(s) : none.
//* Returns : none.
//* Description : main loop
//***************************************************************************
void main(void)
{
   har temp;
  init();                           // initialize P89LPC932
    I2C_Init();                     // initialize I2C block
    EA  = 1;                        // enable interrupts
    while(1)                        // main loop
    {
        switch(slaveBuf[0])         // switch on first byte received
        {
          case(0x00):               // Command 00, write byte to P2
          {
              P2 = slaveBuf[1];
          }
          break;
          case(0xEE):               // command EE, change address
          {                         // according jumper settings
              temp = P0;
              temp &= 0x07;         // mask out non address bits
              temp <<= 1;           // shift left one
              I2ADR = (0xE0 | temp);  // generate I2C address depending on P0
          }
          break;
          case(0xFF):               // command FF, send back I2C slave address
          {
              slaveBuf[0] = I2ADR;
          }
          break;
          default:
          {
          }
        break;
    }
  }
}
```

i2cslave.c

```
/*****************************************************************
/*  Name of module:     I2CSLAVE.C
/*  Creation date:        12 March 2003
/*  Program language  :     C
/*
/*        (C) Copyright 2003 Philips Semiconductors B.V.
/*
/*****************************************************************
#include <Reg932.H>
#include "ua_exprt.h"
typedef unsigned char    BYTE;
typedef unsigned short   WORD;

#define SLAVE_IDLE             0
#define SLAVE_BUSY             1
#define SLAVE_READY            2
#define SLAVE_TRX_ERROR        3
#define SLAVE_RCV_ERROR        4

#define GENERATE_STOP          0x54                // set STO, clear STA and SI
#define RELEASE_BUS_ACK        0x44                // clear STO,STA,SI and set AA (ack)
#define RELEASE_BUS_NOACK      0x40                // clear STO, STA, SI and AA (noack)
#define RELEASE_BUS_STA        0x64                // generate (rep)START, set STA

static BYTE count;                                 // bytes send/received of current message
char slaveBuf[3];
static BYTE size;                                  // size of slave mode buffer
static BYTE slaveStatus;                           // status of the slave

void I2C_Interrupt(void) interrupt 6 using 1
{
  switch(I2STAT)
    {
        case 0x60:                                 // own SLA+W received, Ack returned
        case 0x68:                                 // Addressed as slave
        case 0x70:                                 // General call received, Ack returned
        case 0x78:
            slaveStatus = SLAVE_BUSY;              // Data will be received
            count = 0;
            if (size > 1)
              I2CON = RELEASE_BUS_ACK;             // return ACK on first byte
            else
                I2CON = RELEASE_BUS_NOACK;         // return NACK on first byte
          break;
        case 0x80:
        case 0x90:                                 // Data received, ACK returned
            slaveBuf[count++] = I2DAT;             // read data
            if (count == size)
              I2CON = RELEASE_BUS_NOACK;           // return NACK on next byte
            else
              I2CON = RELEASE_BUS_ACK;             // return ACK on next byte
          break;
        case 0x88:
        case 0x98:                                 // received, NACK returned
            slaveStatus = SLAVE_RCV_ERROR;
            I2CON = RELEASE_BUS_ACK;               // clr SI, set AA
          break;
        case 0xA0:                                 // STOP or REP.START received, addressed as slave
            slaveStatus = SLAVE_READY;
            I2CON = RELEASE_BUS_ACK;               // clr SI, set AA
          break;
        case 0xB0:                                 // Arb. lost as MST, addressed as slave transmitter
            slaveStatus = SLAVE_BUSY;
            count = 0;
            I2DAT = slaveBuf[count++];             // Transmit next data, restart
            I2CON = RELEASE_BUS_STA;               // MST mode if bus is free again
          break;
        case 0xA8:                                 // Addressed as slave transmitter
            slaveStatus = SLAVE_BUSY;
            count = 0;
        case 0xB8:                                 // Data transmitted, ACK received
              I2DAT = slaveBuf[count++];           // Transmit next data
              I2CON = RELEASE_BUS_ACK;             // clr SI, set AA
         break;
        case 0xC0:                                 // Data transmitted, NOT ACK received
            slaveStatus = SLAVE_TRX_ERROR;
        case 0xC8:
            I2CON = RELEASE_BUS_ACK;               // clr SI, set AA
```

```
        break;
       default:
    break;
  }
}

//void I2C_Init(BYTE *buf, BYTE size)
void I2C_Init(void)
{
  unsigned char temp;
  slaveStatus = SLAVE_IDLE;
    slaveBuf[0] = 0xA5;
    slaveBuf[1] = 0x55;
    slaveBuf[2] = 0x99;
  size = 3;

    P1M1  |= 0x0C;                          // Configure P1.2 and P1.3 to open drain
    P1M2  |= 0x0C;

//*****************************************************************************
//*  Modified from Paul's code to select an address depending on the jumper
//*  settings of A2, A1 and A0
//*****************************************************************************
    temp = P0;
    temp &= 0x07;                           // mask out non address bits
    temp <<= 1;                             // shift left one
    I2ADR = (0xE0 | temp);                  // generate I2C address depending on P0
//*****************************************************************************
//* End modification
//*****************************************************************************

    I2SCLH = 0x05;
    I2SCLL = 0x04;
    I2CON = RELEASE_BUS_ACK;                // enable I2C hardware
    EI2C  = 1;                              // enable I2C interrupt
}

void I2C_ProcessSlave(void)
/*****************************************************************************
 * Input(s) :    None.
 * Output(s) :   None.
 * Returns :     None.
 * Description:  Process the slave.
 *               This function must be called by the application to check
 *               the slave status. The USER should adapt this function to
 *               his personal needs (take the right action at a certain
 *               status).
 *****************************************************************************/
{
  switch(slaveStatus)
    {
       case SLAVE_IDLE :
         /* do nothing or fill transmit buffer for transfer        */
        break;
       case SLAVE_BUSY :
         /* do nothing if interrupt driven                          */
         break;
       case SLAVE_READY :
         /* read or fill buffer for next transfer, signal application */
         slaveStatus = SLAVE_IDLE;
         break;
       case SLAVE_TRX_ERROR :
         /* generate error message                                   */
         slaveStatus = SLAVE_IDLE;
         break;
       case SLAVE_RCV_ERROR :
         /* generate error message                                   */
         slaveStatus = SLAVE_IDLE;
    break;
     }
}
```

ua_exprt.h

```
/*****************************************************************************

/*
/*        (C) Copyright 1993 Philips Semiconductors B.V.
/*
/*****************************************************************************
/*
/*  Description:
/*
/*  This module consists a number of exported declarations of the I2C
/*  driver package. Include this module in your source file if you want
/*  to make use of one of the interface functions of the package.
/*
/*****************************************************************************

/**** Status Errors ****/

#define I2C_OK                  0        /* transfer ended No Errors  */
#define I2C_BUSY                1        /* transfer busy             */
#define I2C_ERR                 2        /* err: general error        */
#define I2C_NO_DATA             3        /* err: No data in block     */
#define I2C_NACK_ON_DATA        4        /* err: No ack on data       */
#define I2C_NACK_ON_ADDRESS     5        /* err: No ack on address    */
#define I2C_DEVICE_NOT_PRESENT  6        /* err: Device not present   */
#define I2C_ARBITRATION_LOST    7        /* err: Arbitration lost     */
#define I2C_TIME_OUT            8        /* err: Time out occurred    */
#define I2C_SLAVE_ERROR         9        /* err: slave mode error     */
#define I2C_INIT_ERROR          10      /* err: Initialization (not done)*/

extern char slaveBuf[];                 // ptr to rec/trm data into/from if slave

/*****************************************************************************/
/*       I N T E R F A C E   F U N C T I O N   P R O T O T Y P E S        */
/*****************************************************************************/

extern void I2C_Init(void);
```

Appendix 3: PCA9564 evaluation Board Bill Of Material

Part	Value	Package	Description
C1	100 nF	C0603	CAPACITOR
C2	100 nF	C0603	CAPACITOR
C3	100 nF	C0603	CAPACITOR
C4	100 nF	C0603	CAPACITOR
C5	100 nF	C0603	CAPACITOR
C6	100 nF	C0603	CAPACITOR
C7	1 µF	C0603	CAPACITOR
C8	100 nF	C0603	CAPACITOR
C9	100 nF	C0603	CAPACITOR
C10	100 nF	C0603	CAPACITOR
C11	100 nF	C0603	CAPACITOR
C12	100 nF	C0603	CAPACITOR
C13	100 nF	C0603	CAPACITOR
C14	22 pF	C0603	CAPACITOR
C15	22 pF	C0603	CAPACITOR
C16	22 pF	C0603	CAPACITOR
C17	22 pF	C0603	CAPACITOR
C18	100 µF	085CS_1AR	POLARIZED CAPACITOR
C19	100 nF	C0603	CAPACITOR
C20	100 nF	C0603	CAPACITOR
DB9	F09HP		SUB-D Connector
IC1	7402PW	TSSOP14	Quad 2-input NOR gate
IC3	LM317TL	317TL	VOLTAGE REGULATOR
JP1			JUMPER
JP2			JUMPER
JP3			JUMPER
JP4			FEMALE 90 DEGREES CONNECTOR
JP5			FEMALE 90 DEGREES CONNECTOR
JP6			JUMPER
JP7			JUMPER
JP8			JUMPER
JP9			JUMPER
JP10			JUMPER
JP11			JUMPER
JP12			JUMPER
JP13			JUMPER
JP14		1X06	PIN HEADER
JP15			JUMPER
JP16		1X08	PIN HEADER
JP17			JUMPER
JP18			JUMPER
JP19			JUMPER
JP20			JUMPER
JP21			JUMPER
LD1 to LD8		PLCC2	Red LED
LD9 to LD12		PLCC2	Green LED
LD13 to LD20		PLCC2	Red LED
LD21 to LD23		PLCC2	Green LED
LD24		PLCC2	Red LED
PCA9564	PCA9564PW	TSSOP20	I2C-bus Controller
PCF85116	PCF85116	SO8	256 x 8 bits I2C EEPROM
PORT0LPC		2X04	PIN HEADER
PORT1LV51		2X05	PIN HEADER
Q1	11.0592 MHz	QS	CRYSTAL
Q2	12 MHz	QS	CRYSTAL
R1	1 kΩ	R0603	RESISTOR
R2	1.5 kΩ	R0603	RESISTOR
R3	10 kΩ	R0603	RESISTOR
R4	1 kΩ	R0603	RESISTOR
R5	1 kΩ	R0603	RESISTOR
R6	10 kΩ	R0603	RESISTOR
R7	10 kΩ	R0603	RESISTOR
R8	10 kΩ	R0603	RESISTOR
R9	270 Ω	R0603	RESISTOR

Part	Value	Package	Description
R10	270 Ω	R0603	RESISTOR
R11	270 Ω	R0603	RESISTOR
R12	270 Ω	R0603	RESISTOR
R13	10 kΩ	R0603	RESISTOR
R14	270 Ω	R0603	RESISTOR
R15	270 Ω	R0603	RESISTOR
R16	270 Ω	R0603	RESISTOR
R17	270 Ω	R0603	RESISTOR
R18	270 Ω	R0603	RESISTOR
R19	270 Ω	R0603	RESISTOR
R20	270 Ω	R0603	RESISTOR
R21	270 Ω	R0603	RESISTOR
R22	10 kΩ	R0603	RESISTOR
R23	10 kΩ	R0603	RESISTOR
R24	270 Ω	R0603	RESISTOR
R25	270 Ω	R0603	RESISTOR
R26	270 Ω	R0603	RESISTOR
R27	270 Ω	R0603	RESISTOR
R28	270 Ω	R0603	RESISTOR
R29	270 Ω	R0603	RESISTOR
R30	270 Ω	R0603	RESISTOR
R31	270 Ω	R0603	RESISTOR
R32	10 kΩ	R0603	RESISTOR
R33	10 kΩ	R0603	RESISTOR
R34	270 Ω	R0603	RESISTOR
R35	270 Ω	R0603	RESISTOR
R36	270 Ω	R0603	RESISTOR
R37	47 kΩ	R0603	RESISTOR
R38	47 kΩ	R0603	RESISTOR
R39	4.7 kΩ	R0603	RESISTOR
R40	4.7 kΩ	R0603	RESISTOR
R42	47 kΩ	R0603	RESISTOR
R43	47 kΩ	R0603	RESISTOR
RN1	4.7 kΩ	SIL9	SIL RESISTOR
S1		B3F-10XX	PUSHBUTTON
S2		B3F-10XX	PUSHBUTTON
S3		B3F-10XX	PUSHBUTTON
S4		B3F-10XX	PUSHBUTTON
S5		B3F-10XX	PUSHBUTTON
S6		B3F-10XX	PUSHBUTTON
S7		B3F-10XX	PUSHBUTTON
S8		B3F-10XX	PUSHBUTTON
S9		B3F-10XX	PUSHBUTTON
U$1	P89LV51RD2	PLCC44SOCKET	8-bit 8kB Flash 512 B RAM Low Voltage Microcontroller
U$3	74LVC1G04GW	SOT753	
U$4	PMBTA92	SOT23	
U$5	SK14	DO214AC	
U$6	P89LPC932	PLCC28SOCKET	80C51 8-bit microcontroller with two-clock core
U$7	9VDCJACK		
U$8	PCA9531PW	TSSOP16	8-bit LED Dimmer
U$9	PCA9554PW	TSSOP16	8-bit I2C GPIO with Interrupt
U2	SP3223EY	TSSOP20	
VDDMCU+			JUMPER

REVISION HISTORY

Revision	Date	Description
_2	20040819	Application note (9397 750 13956). Modifications: - Added “PCA9564 evaluation board web page” paragraph - Replaced www.philipslogic.com by www.standardproducts.philips.com
_1	20031211	Application note, initial version (9397 750 12508).

Purchase of Philips I^2C components conveys a license under the Philips I^2C patent to use the components in the I^2C system provided the system conforms to the I^2C specifications defined by Philips. This specification can be ordered using the code 9398 393 40011.

Disclaimers

Application information – Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Life support – These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes – Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Contact information
For additional information please visit
http://www.semiconductors.philips.com

Fax: +31 40 27 24825

For sales offices addresses send e-mail to:
sales.addresses@www.semiconductors.philips.com

Date of release: 08-04
Document order number: 9397 750 13956

Let's make things better.

Philips Semiconductors

PHILIPS