



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

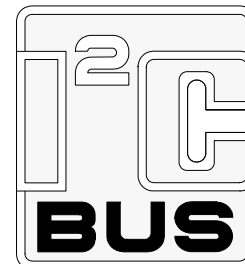
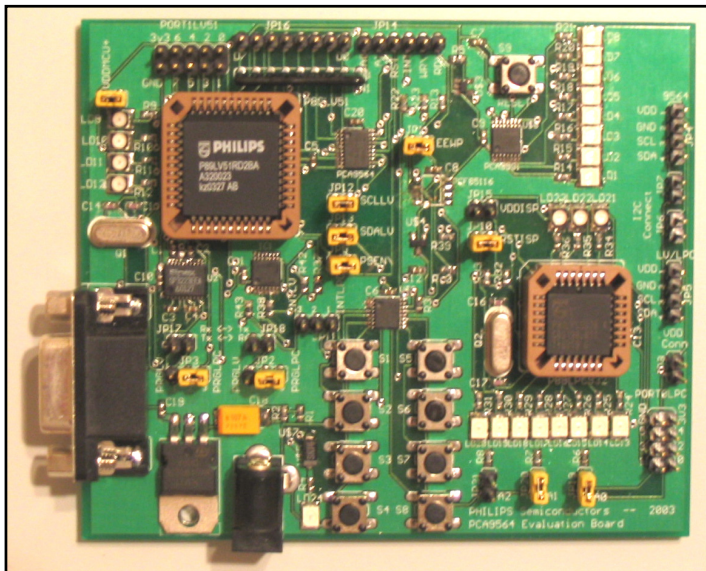
Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China



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
Paul Boogaards
Bauke Siderius

PCA Technical Marketing Manager
Sr. Field Application Engineer
Application Engineer

2004 Aug 19

TABLE OF CONTENTS

OVERVIEW	3
DESCRIPTION	3
ORDERING INFORMATION	4
TECHNICAL INFORMATION – HARDWARE	4
BLOCK DIAGRAM	4
I ² C DEVICE ADDRESSES	4
SCHEMATIC	5
PCA9564 EVALUATION BOARD TOP VIEW	6
JUMPERS AND HEADERS	6
PUSHBUTTONS – USER INTERFACE AND RESET	8
IN-SYSTEM PROGRAMMING MODE	8
<i>P89LV51RD2 ISP programming</i>	9
<i>P89LPC932 ISP programming</i>	9
OTHER FEATURES	9
<i>Write Protect PCF85116</i>	9
<i>Use of other 80C51 type Philips microcontrollers</i>	10
<i>Use of any other non 80C51 type master devices</i>	10
<i>Communication between the 2 microcontrollers</i>	10
<i>Miscellaneous</i>	11
TECHNICAL INFORMATION – EMBEDDED FIRMWARE	12
OVERVIEW.....	12
EMBEDDED PROGRAMS FLOWCHARTS	14
<i>Program Selection</i>	14
<i>Program 1: P89LV51RD2–PCA9564–PCA9531; PCA9531 dynamic programming</i>	15
<i>Program 2: P89LV51RD2–PCA9564–PCA9531–PCF85116–P89LPC932; Predefined blinking patterns</i>	16
<i>Program 3: P89LV51RD2–PCA9564–PCA9531–P89LPC932; P89LPC932 LED programming</i>	16
<i>Program 4: P89LV51RD2–PCA9564–PCA9531–P89LPC932; I²C address search</i>	17
SOURCE CODE P89LV51RD2 – REV 1.0	18
SOURCE CODE P89LPC932 – REV 1.0	18
DOWNLOAD SOFTWARE, PROGRAMS AND DOCUMENTATION	19
PCA9564 EVALUATION BOARD WEB PAGE	19
APPENDIX 1: P89LV51RD2 MICROCONTROLLER SOURCE CODE – REV 1.0	20
I2CEXPRT.H	20
MAINLOOP.C.....	21
I2C_ROUTINES.H.....	23
I2C_ROUTINES.C.....	23
I2CDRIVR.H.....	36
I2CDRIVR.C.....	36
I2CMASTR.H.....	37
I2CMASTR.C	38
I2CINTFC.C.....	41
PCA9564SYS.H	43
PCA9564SYS.C	43
INTERRUPTS.C.....	44
APPENDIX 2: P89LPC932 MICROCONTROLLER SOURCE CODE – REV 1.0	45
MAIN.C.....	45
I2CSLAVE.C.....	46
UA_EXPRT.H	48
APPENDIX 3: PCA9564 EVALUATION BOARD BILL OF MATERIAL	49
REVISION HISTORY	51
DISCLAIMERS.....	52

OVERVIEW

Description

The PCA9564 Evaluation Board demonstrates the Philips PCA9564 I²C-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 I²C-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** I²C-bus controller interfacing between the P89LV51RD2 and the I²C-bus.
- **Philips PCA9531** I²C 8-bit LED dimmer used as an I²C target slave device for the P89LV51RD2/PCA9564.
- **Philips P89LPC932** microcontroller connected to the I²C-bus. It can act as either a target slave device with the default P89LV51RD2 firmware programs or as a master connected to the I²C-bus through some stored user definable routines.
- **Philips PCF85116** 16 kbits (2KB) I²C EEPROM used to store information that can be used by the evaluation board firmware.
- **Philips PCA9554A** I²C 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 I²C 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 I²C 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

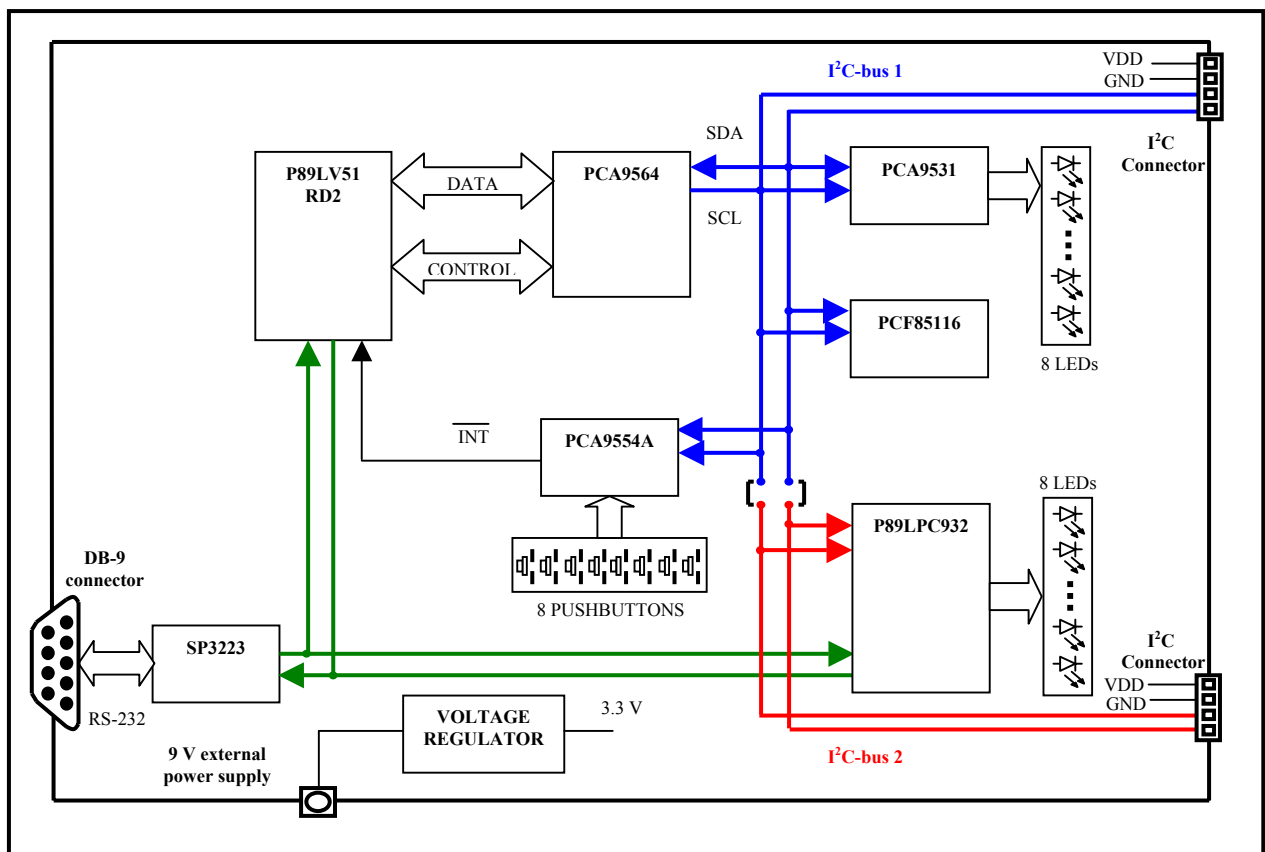


Figure 1. Evaluation board block diagram

I²C device addresses

Device type	Description	I ² C Address (Hexadecimal)
P89LV51RD2 / PCA9564	Microcontroller / I ² C-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 I ² C LED Dimmer	0xC8
PCF85116	16kbits I ² C EEPROM	0xA0 to 0xA8 (function of the addressed memory)
PCA9554A	8-bit I ² C GPIO	0x7E

Table 1. I²C device addresses

PCA9564 Evaluation Board Top view

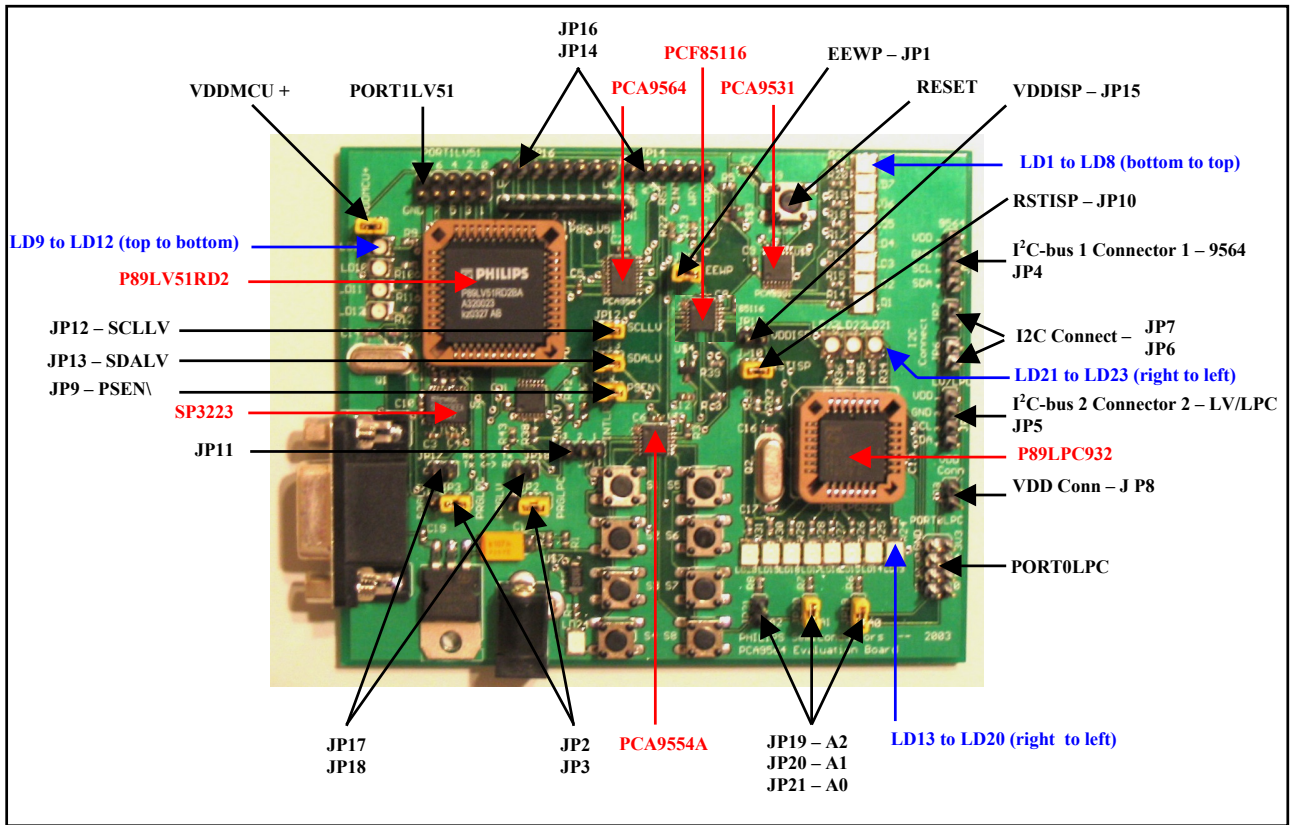


Figure 3. PCA9564 Evaluation Board Top View

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 ₁ IN of SP3223
		Closed between 2 and 3 (PRGLV)	ISP programming of P89LV51RD2 can be performed TxD pin of P89LV51RD2 connected to T ₁ IN of SP3223
JP3	Selection of the microcontroller to be programmed through ISP (Rx/D)	Open	No ISP programming can be performed
		Closed between 1 and 2 (PRGLPC)	ISP programming of P89LPC932 can be performed Rx/D pin of P89LPC932 connected to R ₁ OUT of SP3223
		Closed between 2 and 3 (PRGLV)	ISP programming of P89LV51RD2 can be performed Tx/D pin of P89LV51RD2 connected to R ₁ OUT of SP3223
JP4 (9564)	I ² C-bus connector 1		I ² C-bus 1 – Bus connected to the PCA9564, PCA9531, PCF85116 and PCA9554A Note: I ² C-bus 1 and I ² C-bus 2 can be connected together through jumpers JP6 and JP7
JP5 (LV/LPC)	I ² C-bus connector 2		I ² C-bus 2 – Bus connected to the P89LPC932. It is also connected to the I ² C-bus of a P89C51Rx+/Rx2/66x with I ² C-bus (SCL = P1.6, SDA = P1.7) when JP12 and JP13 closed Note: I ² C-bus 1 and I ² C-bus 2 can be connected together through jumpers JP6 and JP7

JP6 (I2C Connect)	Connect I ² C-bus 1 and I ² C-bus 2	Open	SCL I ² C-bus 1 and SCL I ² C-bus 2 are not connected together
		Closed	SCL I ² C-bus 1 and SCL I ² C-bus 2 are connected together
JP7 (I2C Connect)	Connect I ² C-bus 1 and I ² C-bus 2	Open	SDA I ² C-bus 1 and SDA I ² C-bus 2 are not connected together
		Closed	SDA I ² C-bus 1 and SDA I ² C-bus 2 are connected together
JP8 (VDD Conn)	Power supply for the I ² C-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 $\overline{\text{INT}}$ pin not monitored
		Closed between 1 and 2 (INTLPC)	PCA9554A $\overline{\text{INT}}$ pin can be monitored by P89LPC932
		Closed between 2 and 3 (INTLV)	PCA9554A $\overline{\text{INT}}$ pin can be monitored by P89LV51RD2
JP12 (SCLLV)	P89x51 with I ² C-bus connection to I ² C-bus 2	Open	P89C51Rx+/Rx2/66x with I ² C-bus (SCL = P1.6) not connected to SCL I ² C-bus 2
		Closed	P89C51Rx+/Rx2/66x with I ² C-bus (SCL = P1.6) connected to SCL I ² C-bus 2
JP13 (SDALV)	P89x51 with I ² C-bus connection to I ² C-bus 2	Open	P89C51Rx+/Rx2/66x with I ² C-bus (SDA = P1.7) not connected to SDA I ² C-bus 2
		Closed	P89C51Rx+/Rx2/66x with I ² C-bus (SDA = P1.7) connected to SDA I ² C-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 ² C 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 ² C 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 ² C 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, I²C 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 I²C address with a Write command followed by 0x00 (Input register pointer).
2. A Re-Start Command followed by the PCA9554A I²C 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. I²C-bus 1 and I²C-bus 2 need to be connected together since the PCA9554A is located on I²C-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 I²C interface:
Port P1.6 (SCL) and P1.7 (SDA) can be connected to the internal I²C-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 I²C-bus:
Jumpers JP6 and JP7 allow to connect or split the I²C-bus in one same bus or 2 different buses.
I²C-bus 1 contains the following devices: P89LV51RD2/ PCA9564, PCA9531, PCF85116 and PCA9554A
I²C-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 togetherSince 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 I²C-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-I²C 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 I²C-bus connectors:
 Jumper JP8 (VDD Conn), when closed, connect the V_{DD} pins in the two I²C-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 I²C-bus connectors is floating
 - JP8 closed: V_{DD} pin in the two I²C-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 I²C 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 I²C 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 I²C master, the PCA9531 (with LD1 to LD8) as an I²C 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 I²C master, the PCF85116, the PCA9531 (with LD1 to LD8) and the P89LPC932 (with LD13 to LD20) as I²C slaves to display preprogrammed blinking patterns stored in the EEPROM.

For a specific selected pattern:

- a) Data used to program the PCA9531 is 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.

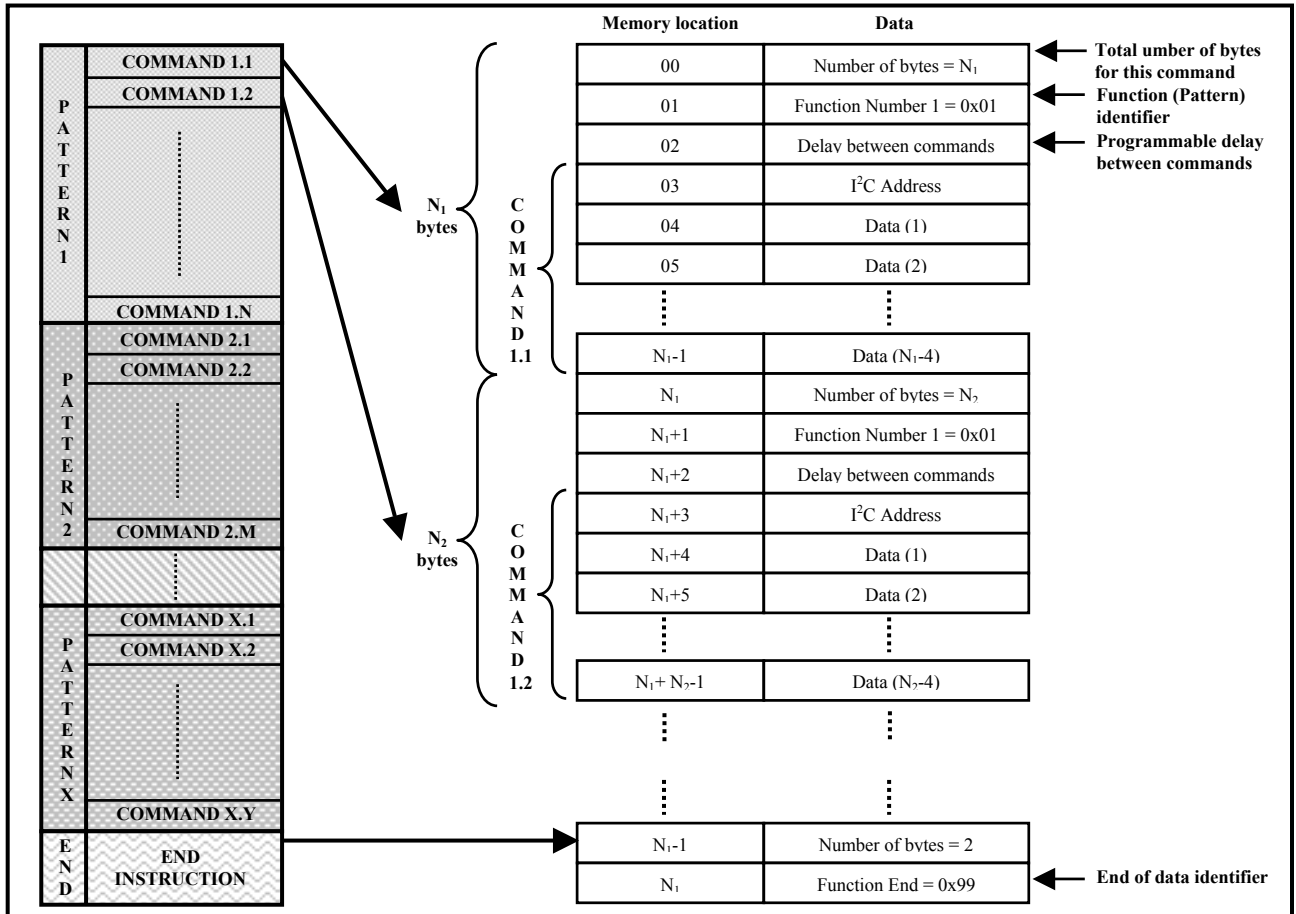


Figure 4. PCF85116 memory organization

- **Program 3 (LD9 = OFF, LD10 = ON): P89LPC932 LED programming**
 Program 3 uses P89LV51RD2/PCA9564 as an I²C master, the PCA9531 (with LD1 to LD8) and the P89LPC932 (with LD13 to LD20) as I²C 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): I²C address search**

Program 4 uses the P89LV51RD2/PCA9564 as an I²C master and the P89LPC932 (with jumpers JP19 to JP21) as an I²C slave. In this mode, the PCA9564 searches for the P89LPC932's I²C slave address (JP19 to JP21 programs the 3 LSB's of the P89LPC932 I²C 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 I²C address search routine
- S2: Resets the P89LV51RD2 search routine algorithm and initiates a P89LPC932 I²C 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 → I²C address not found or search routine not performed yet
- LD11 = ON → I²C address search routine successful
- LD12 = OFF → search routine not performed yet
- LD12 = ON → search routine performed and I²C address not found

Embedded programs flowcharts

Program Selection

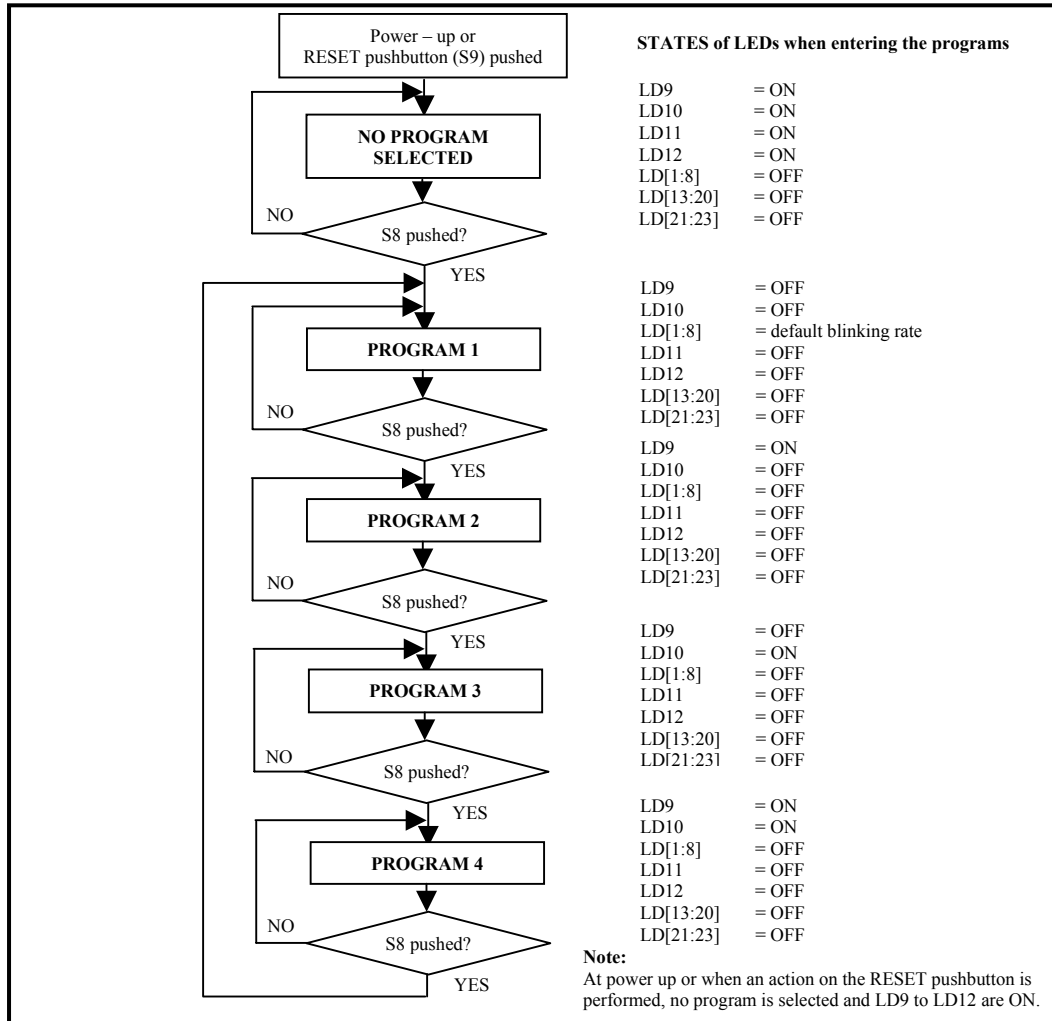


Figure 5. Program selection

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

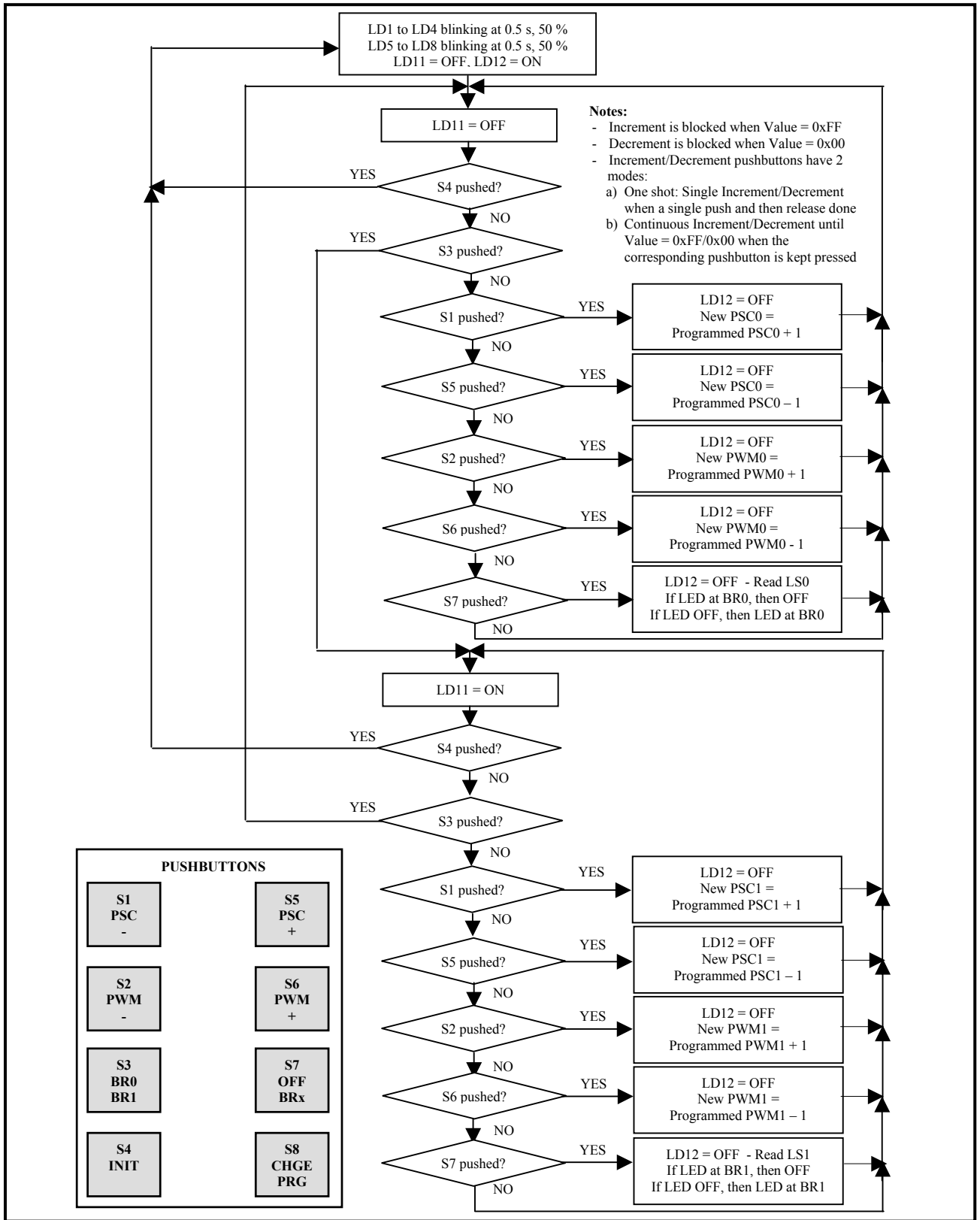


Figure 6. Program 1 – PCA9531 dynamic programming

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

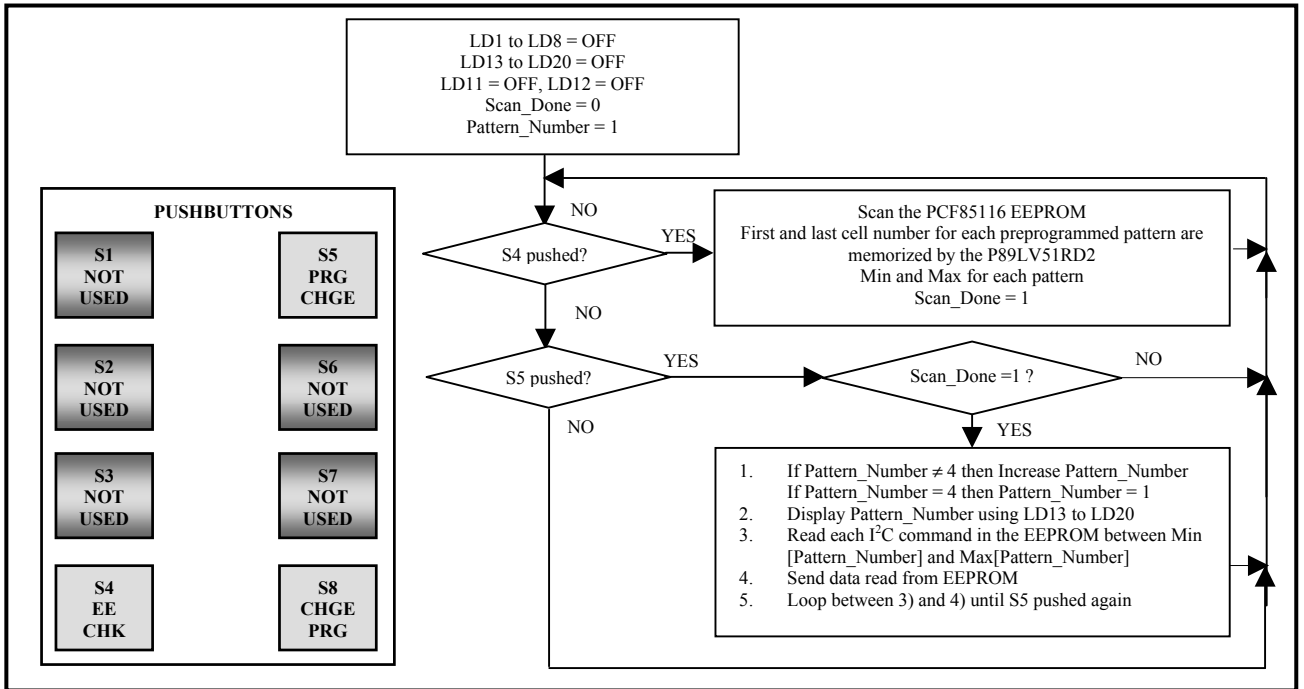


Figure 7. Program 2 – Preprogrammed blinking patterns

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

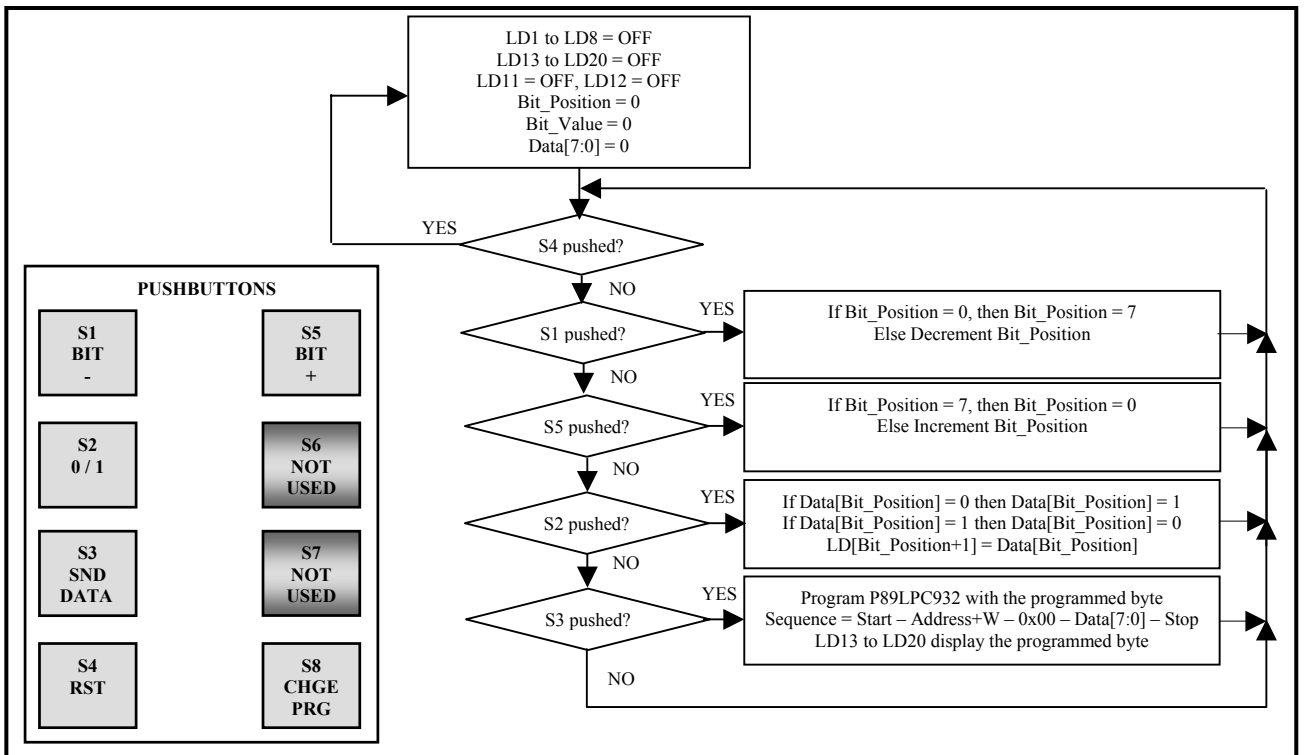


Figure 8. Program 3 – P89LPC932 LED programming

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

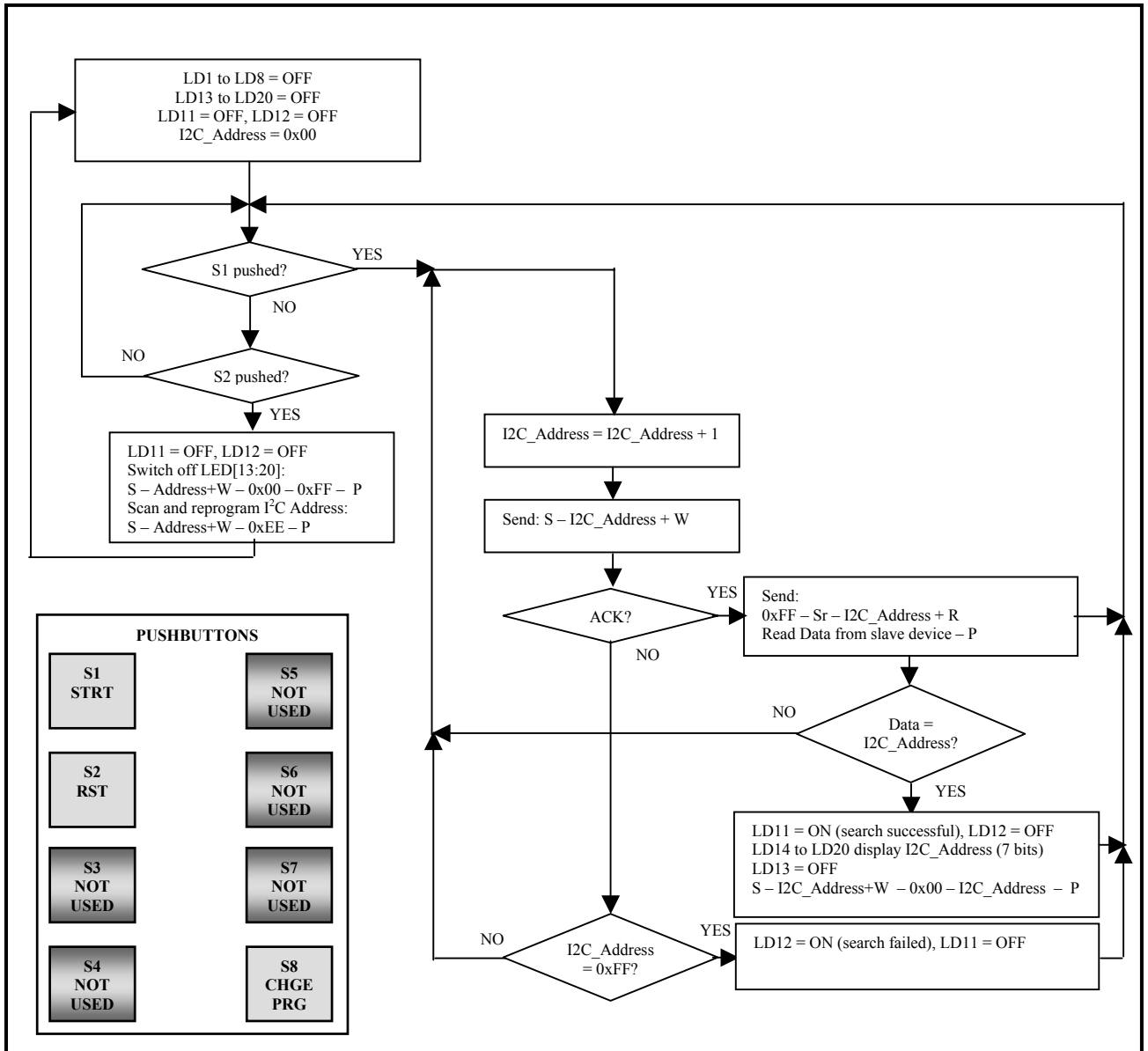


Figure 9. Program 4 – I²C address search

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 I²C 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 I²C 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 “I²C address Scan and Memorize” procedure: S – Address+W – 0xEE – P
 - c) Instruction allowing reading back the I²C slave address: S – Address+W – 0xFF – Sr – Address+R – Data – P with Data = I²C slave address

2. **i2slave.c:**
Contains the source code of the I²C slave core
3. **ua_exprt.h:**
Contains the definition of variables used in the I²C 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 I²C slave devices and μ controllers 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
{
    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
}

```