AN10315

LED dimmer demoboard

Rev. 01 — 7 January 2005 **Application note**

Semiconductors

PHILIPS

Revision history

Rev	Date	Description
01	20050107	Application note (9397 750 14062); initial version.

Contact information

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

For sales office addresses, please send an email to: **sales.addresses@www.semiconductors.philips.com**

9397 750 14062

Application note **Rev. 01 — 7 January 2005** **2 of 45**

1. Introduction

The LED Dimmer Demoboard demonstrates the capability of the I^{2}C-bus to control a keypad and perform Red/Green/Blue LED lighting color mixing operations.

The demoboard is a stand alone solution using the Philips P89LV51RD2 microcontroller interfacing with a Philips PCA9564 I^{2}C-bus controller to generate the different I^{2}C commands. The microcontroller comes programmed with a default firmware that is explained in this Application Note. If additional programming is required, the user must remove the microcontroller from its socket and use an external programmer.

The demoboard is composed by 2 sub-boards that are directly connected to each other without the need for external cables:

- The Keypad Control Card contains the microcontroller, the keypad with the keypad controller and the power supply/regulator module (external 9 V power supply or 9 V battery).
- The LED Control Card contains all the LEDs and the devices controlling them.

Each card can be used separately and can be connected to other companion cards.

I^{2}C slave devices used on the demoboard are the following:

- Philips PCA9555PW, 16-bit GPIO functioning as a 16-key keypad controller.
- Philips PCA9531PW and PCA9533DP/01, 8-bit and 4-bit LED dimmer for LED control and color mixing (Red, Green and White LEDs, RGB LEDs).

2. Ordering information

The complete demoboard kit consists of:

- The keypad control card
- The LED display card

Kit can be obtained through your local Philips Semiconductors Sales organization or via e-mail to i2c.support@philips.com

3. Technical information—hardware
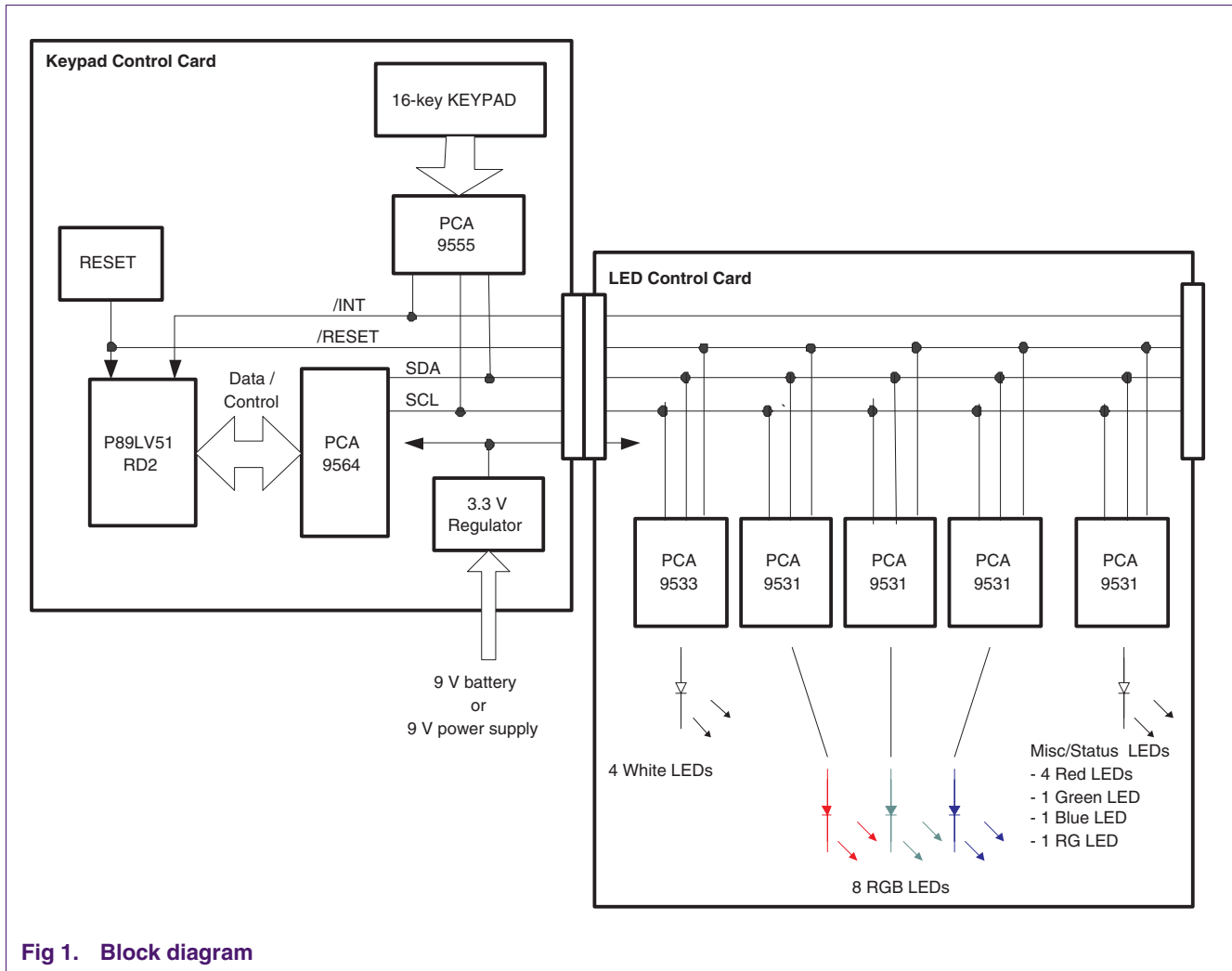
3.1 Block diagram

Fig 1. Block diagram

3.2 I^{2}C-bus device addresses

Table 1: I^{2}C-bus device addresses

Device type	Description	I^{2}C address (hexadecimal)
P89LV51RD2/PCA9564PW	microcontroller / I^{2}C-bus controller	User definable when slave
PCA9531PW (Red - RGB LEDs)	8-bit I^{2}C LED dimmer	0xCA
PCA9531PW (Green - RGB LEDs)	8-bit I^{2}C LED dimmer	0xCC
PCA9531PW (Blue - RGB LEDs)	8-bit I^{2}C LED dimmer	0xCE
PCA9531PW (Misc/Status LEDs)	8-bit I^{2}C LED dimmer	0xC6
PCA9533DP/01 (White LEDs)	4-bit I^{2}C LED dimmer	0xC4
PCA9555	16-bit I^{2}C GPIO	0x46

3.3 Schematic
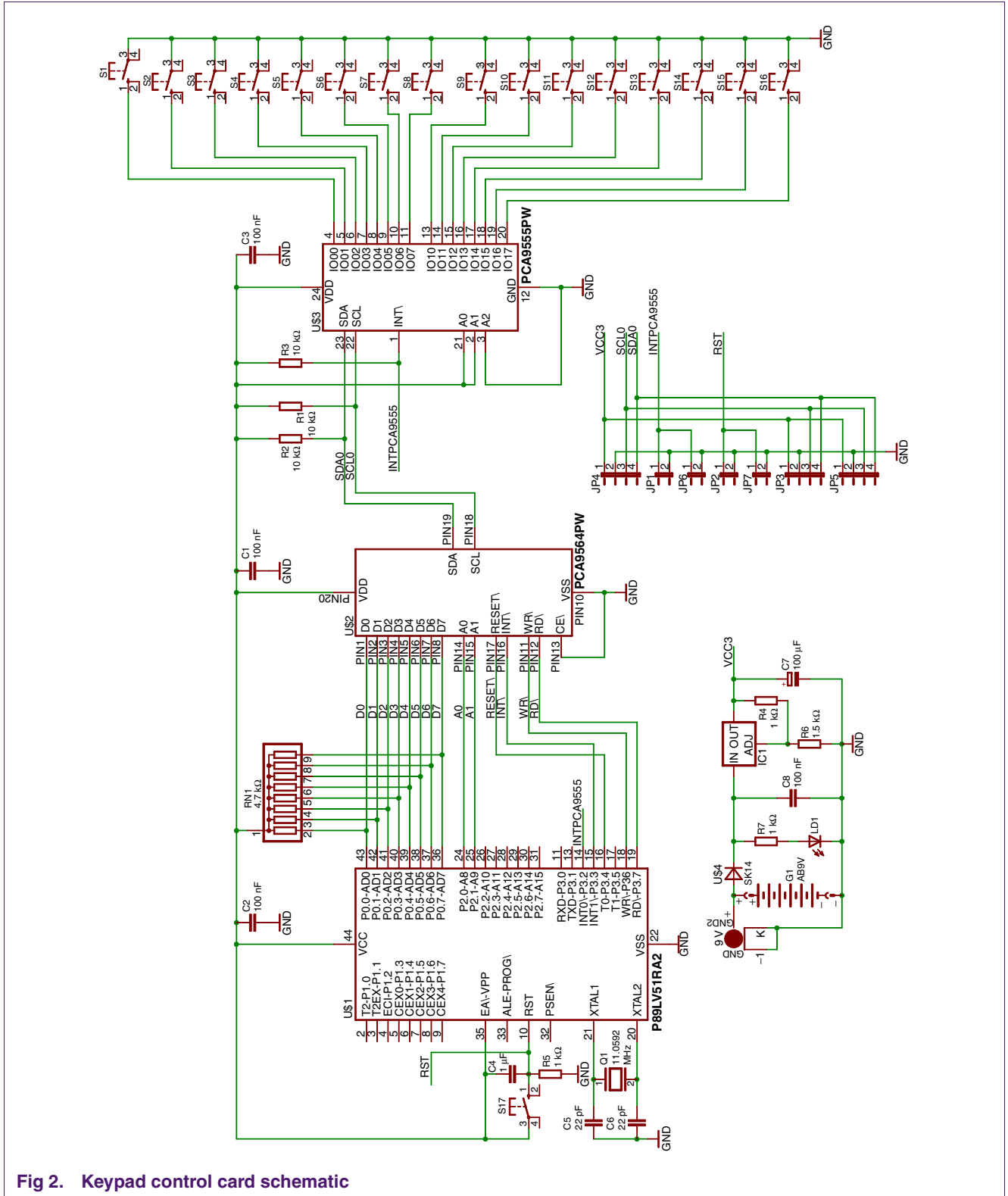
3.3.1 Keypad control card

Fig 2. Keypad control card schematic
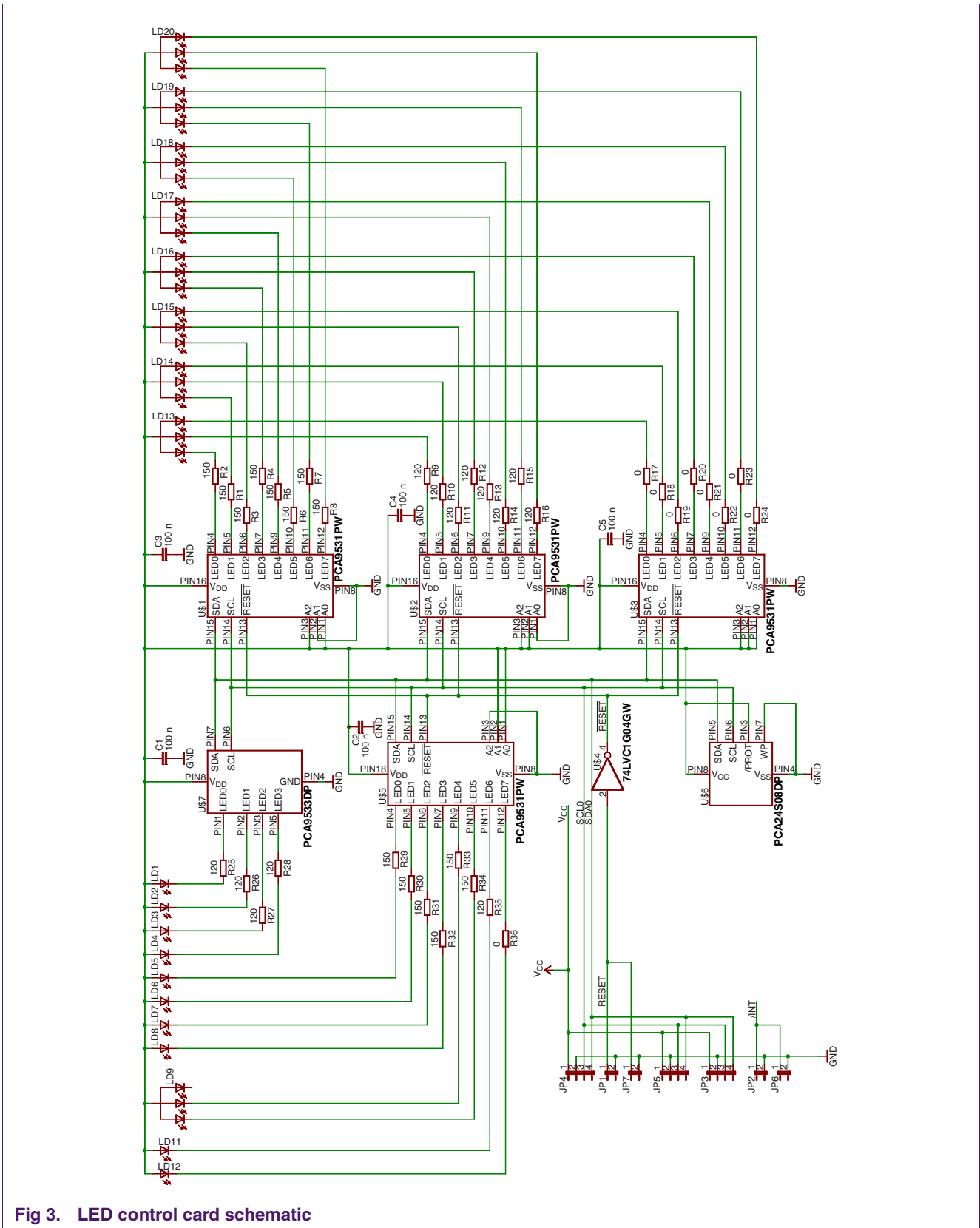
3.3.2 LED control card

Fig 3. LED control card schematic

3.4 Demoboard (top view)
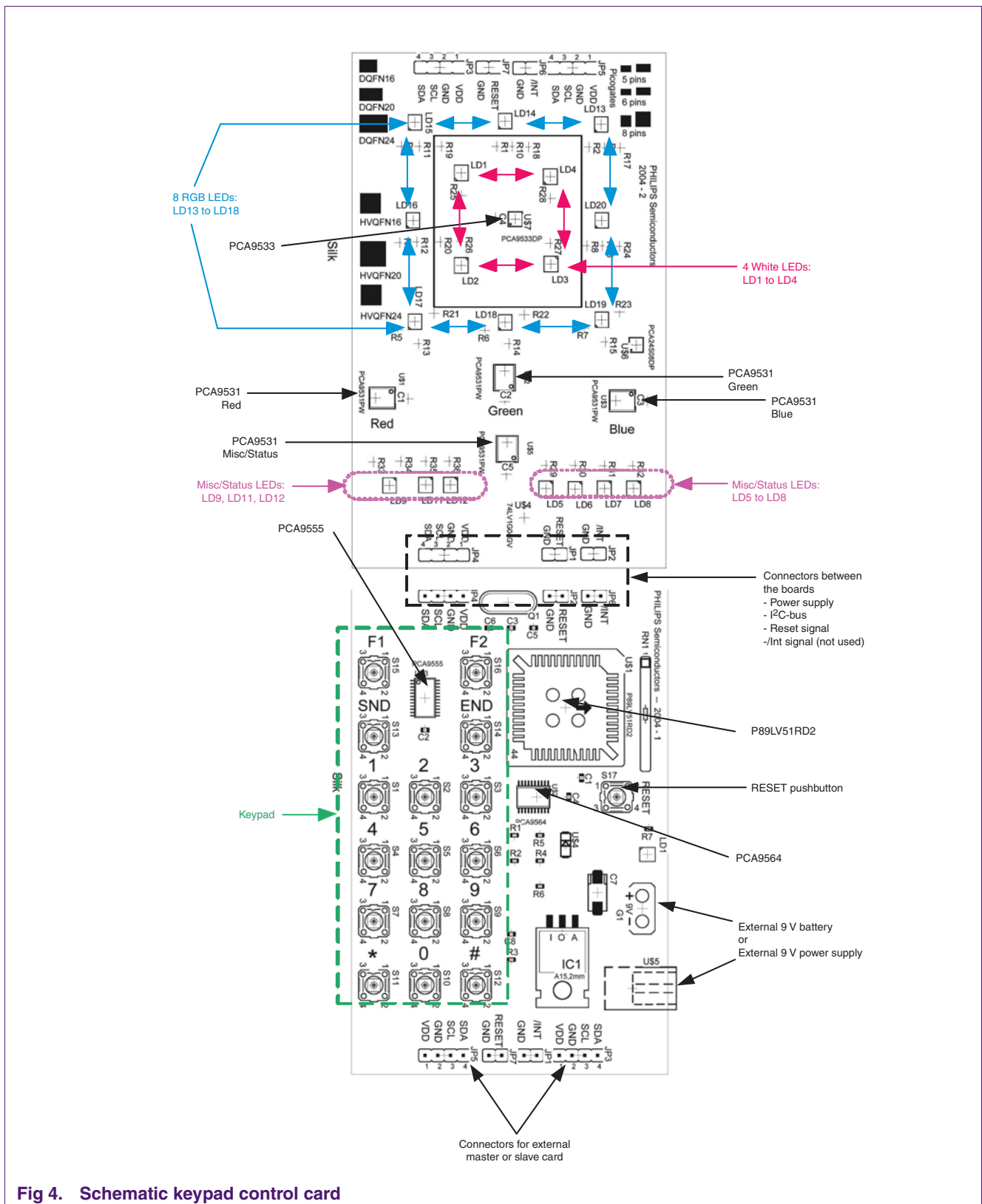
Fig 4. Schematic keypad control card

3.5 RGB color mixing

Red, Green and Blue are the 3 primary colors allowing creating all the other colors by mixing them together.

The desired color is created by applying the right amount of Red, plus the right amount of Green plus the right amount of Blue light from the RGB LED.

A 152 Hz frequency voltage is applied to the Red, Green and Blue LED drivers so that the human eye does not see the ON/OFF cycle (typically frequencies higher than100 Hz are required).

Varying the duty cycle controls the average current flowing through the LED, thus controlling the brightness for each color. The human eye sees an average brightness value since it cannot see the ON/OFF cycle.

The sum of the 3 primary colors at various brightness values will then define the resulting color: <IRed> + <IGreen> + <IBlue> gives the desired visible color.

Fig 5. Color mixing theory

4. Technical information—how to use the demobord

4.1 Introduction

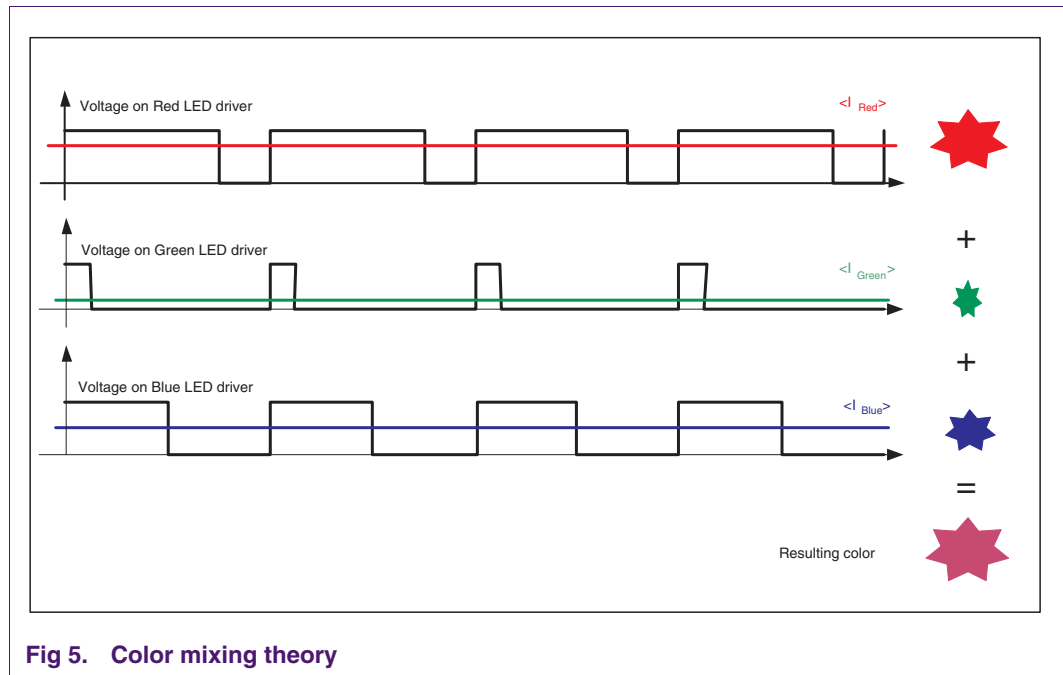Firmware in the P89LV51RD2 microcontroller emulates a cell phone application, programs fun patterns, and controls the brightness of a virtual display. A battery discharge emulation with a visual charge status can also be performed. The firmware is intended to show a specific application (i.e., cell phone), but this demoboard can be used for any application that requires a keypad control and LED lighting/color mixing.

Most of the code, written in C language, has been re-used from the PCA9564 Evaluation Board (drivers, main interfacing files between the P89LV51RD2 and the PCA9564). Only the *mainloop.c*, *i2c_routines.c* and *i2cexprt.h* files have been modified to implement the code specific to the demoboard. For more information about the files different from the 3 mentioned above, refer to the Application Notes *AN10149: PCA9564 evaluation board*, and to the PCA9564 Evaluation Board main page at: http://www.standardproducts.philips.com/support/boards/pca9564/.

The default firmware in the P89LV51RD2 allows the user to:

- Control the 16-key keypad (numbers from 0 to 9 are displayed in binary code)
- Control the RGB LEDs and program 3 different colors / display speed
- Control the brightness of the white LEDs
- Emulate a 'battery discharge' application
- Enable an 'Auto Demo Mode' showing some 'fun light' application / RGB mixing

The P89LV51RD2 can be programmed with any user defined firmware. Since the demoboard does not have any built-in programming feature, the user must then program the microcontroller with an external programmer.

Default firmware (source files and '.hex' file) can be downloaded from the following link: http://www.standardproducts.philips.com/support/boards/leddemo/.

9397 750 14062

Application note Rev. 01 — 7 January 2005 9 of 45
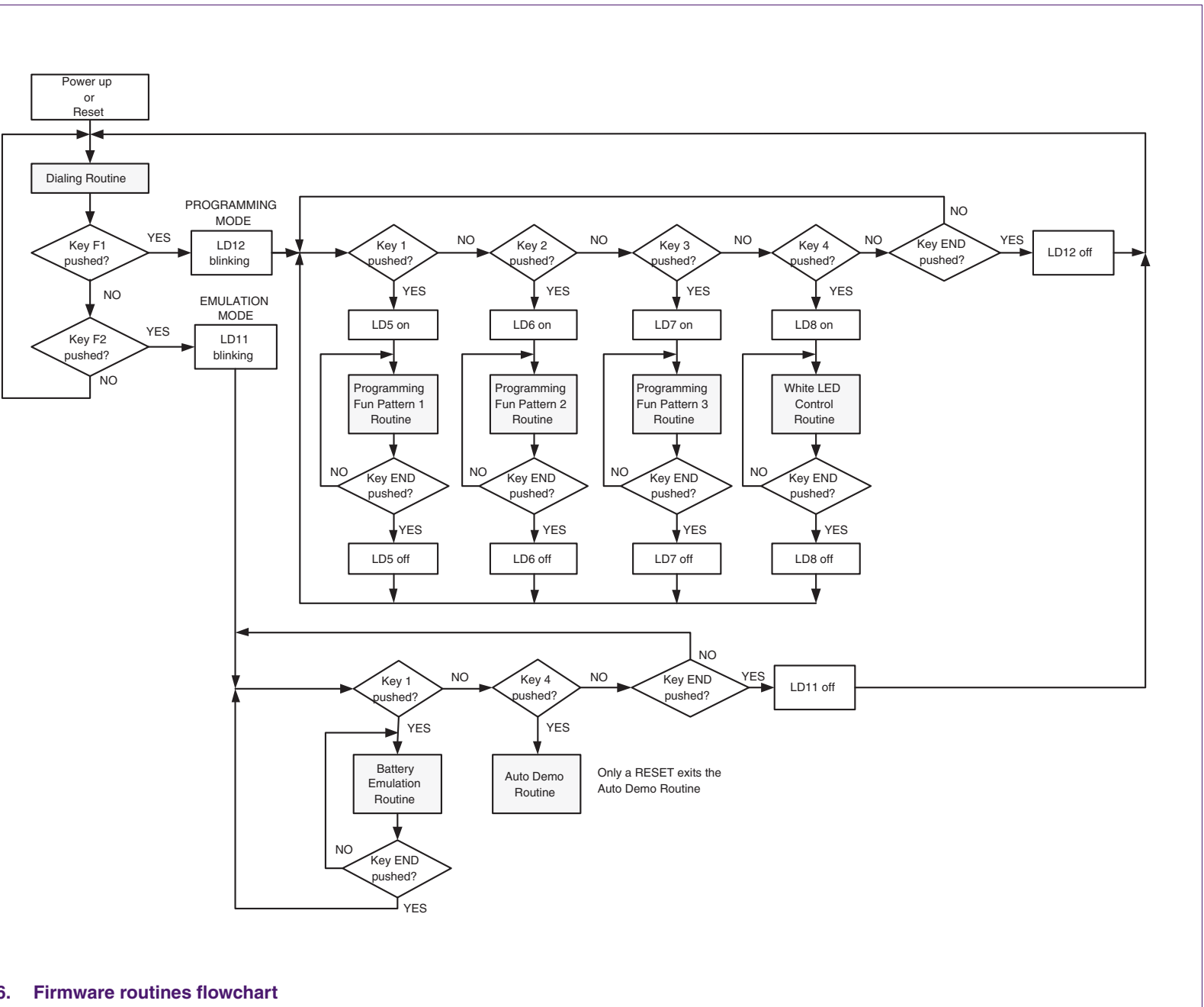
Fig 6.   Firmware routines flowchart

4.3 Dialing routine

4.3.1 How the keypad control works

Keypad control is done using a PCA9555 with its $\overline{INT}$ signal connected to a GPIO of the P89LV51RD2 microcontroller. The 16 keys of the keypad are simply connected to the 16 GPIOs of the PCA9555 and to the ground, without any other external components since internal 100 kΩ pull-ups are provided by the PCA9555.

Each time one (or more) key is pressed, the PCA9555 generates an Interrupt that is detected by the P89LV51RD2. A read of the input port registers of the PCA9555 is then initiated in order to determine which key(s) has (have) been pushed. Two modes are possible:

1. One time only pushed detection: only one read is performed.
2. Continuous push detection: the P89LV51RD2 keeps polling the input port registers as long as the register content is different from 0xFF (meaning that at least one key is pushed).

Code is described in the function called 'GPIO_Interrupt_Handler()' described in Section 5.3 "I2C_Routines.c".

When an Interrupt is detected, the P89LV51RD2 initiates 2 input port register readings. A delay between the 2 readings allows:

- potential switch bounces to be filtered.
- Detection of a one time or continuous push detection.

The delay is long enough so that if 2 readings are identical, it assumes that a continuous push is applied to the pushbutton. If the second reading is different, it assumes that the user released the pushbutton. The delay is also short enough so that a push in another key is not masked.

A variable called 'GPIO_Polling_On' enable/disables the polling option thus making the scan suitable for a 'one shot' detection.

Continuous push detection is enabled when set to '1'.

4.3.2 Keypad mapping with the PCA9555 I/Os

Table 2: PCA9555 Input port Register 0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8	7	6	5	4	3	2	1

Table 3: PCA9555 Input port Register 1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
F2	F1	END	SND	#	*	0	9

4.3.3 Application: cell phone type keyboard—Dialing routine

The firmware in the P89LV51RD2 targets a cell phone type application where the keypad is used to dial phone numbers and access to a programming mode in order to access the RGB and White LEDs.

When dialing a phone number (keys 0 to 9 are used), the pushed key value is displayed using its binary code with LD5 (MSB) to LD8 (LSB).

Table 4:   Binary code for numbers from 0 to 9

KEY	LD5	LD6	LD7	LD8
0				
1				♦
2			♦	
3			♦	♦
4		♦		
5		♦		♦
6		♦	♦	
7		♦	♦	♦
8	♦			
9	♦			♦

Note that a maximum of 10 keys can be pushed when dialing a phone number. If more than 10 keys are entered, LD5 to LD8 will start flashing at 1 Hz / 50 % duty cycle to let the user know that the maximum number has been reached.

After dialing the number, the user has to push the SND key to send the number or the END key to end the call. LD5 to LD8 become off when those 2 keys are pushed.

When SND is pushed, all eight RGB LEDs will blink at 1 Hz / 50 % duty cycle with either a red color (visual emulating a 'busy' line) or a green color (visual emulating a 'non busy' line).

Pushing END ends the call and the RGB LEDs (Red or Green) are off.

Remark: To dial a number, the user needs to be sure that the programming mode is not enabled (LD12 is not blinking) or the emulation mode is not enabled (LD11 is not blinking). If the demoboard is either in the programming mode or in the emulation mode, the END key must be pushed one or two times (depending which branch of the firmware is active) until LD11 and LD12 are off. A reset can also be performed to reach the dialing routine.

4.4 Programming mode

The programming mode shows capabilities of the PCA9533 and the PCA9531 LED dimmers to control LEDs for:

- RGB (Red / Green / Blue) color mixing
- White LED brightness control

Power supply used for both I^2C devices and LEDs is equal to 3.3 V. Applications requiring LEDs connected to 5 V or even higher power supply are also possible. Power supply values higher than 5 V require the use of external FET drivers. More information can be found in the Application Notes *AN264 - I^2C devices for LED display control*.

- Programming mode is entered by pushing key F1. LD12 (Blue LED) starts then blinking to indicate that the programming mode is active.
- Keys 1, 2 and 3 select the 3 programmable 'fun patterns'. See Section 4.4.1 "Programming fun patterns routine—RGB color mixing" for more detail.
- Key 4 selects the white LED's control. See Section 4.4.3 "White LED control routine" for more detail.

LD5 to LD8 let the user know which programming is active.

Table 5:   Active programming

Programming	Key to be pushed	LD5	LD6	LD7	LD8
Fun Pattern 1	1	ON	OFF	OFF	OFF
Fun Pattern 2	2	OFF	ON	OFF	OFF
Fun Pattern 3	3	OFF	OFF	ON	OFF
White LED control	4	OFF	OFF	OFF	ON

- Pushing Key END once leaves the current programming that was performed (fun pattern 1, 2 or 3, white LED control). The corresponding LED (LD5 to LD8) is then off.
- Pushing Key END one more time leaves the programming mode. LD12 stops blinking.

4.4.1 Programming fun patterns routine—RGB color mixing

Three 'fun patterns' can be programmed when using the default firmware. For each 'fun pattern', the user can select:

- The resulting color: selection of amount or Red, Green and Blue.
- The rotating speed:

  LD13 →LD14 →LD15 →LD16 →LD17 →LD18 →LD19 →LD20 →LD13

For each pattern, primary colors (Red, Green, Blue) can be added (+) or removed (-) and the rotating speed can be increased (+) or decreased (-) by pushing the corresponding key in the keypad.

- Key 2: Red (-)
- Key 3: Red (+)
- Key 5: Green (-)
- Key 6: Green (+)
- Key 8: Blue (-)

9397 750 14062

Application note   Rev. 01 — 7 January 2005   13 of 45

- Key 9: Blue (+)
- Key 0: Speed (-)
- Key #: Speed (+)

Remark: Those keys have been programmed to detect a continuous push applied to them. When pushed continuously, (+) or (-) action is performed continuously until max (0xFF) or min (0x00) in the PWM registers.

Remark: Keys in this routine have been programmed to detect a continuous push applied to them. When pushed continuously, (+) or (-) action is performed continuously until max (0xFF) or min (0x00) in the PWM registers.

4.4.2 PCA953x mapping with the LEDs

Table 6:   PCA933 (White LEDs) - LS0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LD4		LD3		LD2		LD1	

Table 7:   PCA931 (Misc/Status LEDs) - LS0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LD8		LD7		LD6		LD5	

Table 8:   PCA931 (Misc/Status LEDs) - LS1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LD12		LD11		LD9 Green		LD9 Red	

Table 9:   PCA931 (Red/Green/Blue LEDs) - LS0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LD16		LD15		LD14		LD13	

Table 10:   PCA931 (Red/Green/Blue LEDs) - LS1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LD20		LD19		LD18		LD17	

4.4.3 White LED control routine

Brightness control for the 4 white LEDs is performed with the following keys:

- Key 2: Brightness (-)
- Key 3: Brightness (+)

Remark: Keys in this routine have been programmed to detect a continuous push applied to them. When pushed continuously, (+) or (-) action is performed continuously until max (0xFF) or min (0x00) in the PWM registers.

4.5 Emulation mode: battery discharge application

Emulation mode can be described as a mode that would require external 'stimulus' to show an application in a real environment. The example in the firmware shows a simple application where a battery discharge (for example in a cell phone application) is emulated by pushing a key. A LED controlled by a PCA9531 is used to provide visual status of the battery level:

- Emulation mode is entered by pushing key F2. LD11 starts blinking.
- Pushing key 1 starts battery discharge emulation.
  - LD5 to LD8 are on, indicating a fully charged battery (100 %).
  - LD9 blinks slowly (1 Hz) and high duty cycle, with a Green color.
- Pushing continuously Key 3 emulates a battery discharge (from 100 % with LD5 to LD8 on, down to 0 % with LD5 to LD8 off). The different steps are explained in Table 11. Principle is to change the LED color (Green, Orange, Red) and the LED duty cycle (shorter duty cycle) to catch the user's attention when using their cell phone.

Table 11:   Battery discharge steps

Step	Battery charge	LD5	LD6	LD7	LD8	LD9
1	100 %	ON	ON	ON	ON	Green - 1 Hz - 93 % duty cycle
2	75 %	OFF	ON	ON	ON	Green - 1 Hz - 93 % duty cycle
3	50 %	OFF	OFF	ON	ON	Orange - 1 Hz - 50 % duty cycle
4	25 %	OFF	OFF	OFF	ON	Red - 1 Hz - 6 % duty cycle
5	0 %	OFF	OFF	OFF	OFF	Red - 1 Hz - 0.4 % duty cycle

- A reset of the emulation is performed by pushing key 6. Battery is then fully charged.
- Pushing key END exits the battery discharge emulation mode. LD5 to LD9 are off.
- Pushing again key END exits the Emulation mode. LD11 stops blinking.

Remark: Keys in this routine have been programmed to detect a continuous push applied to them. When pushed continuously, (+) or (-) action is performed continuously.

4.6 Auto Demo routine

The Auto Demo routine shows some light effects real time RGB mixing without having to push any buttons.

- Emulation mode is entered by pushing key F2. LD11 starts blinking
- Pushing key 4 starts the auto demo mode

Remark: Once the Auto Demo Mode starts (after pushing key 4), the user must use the RESET button to exit the mode.

4.7 RESET

The RESET button located in the Keypad Control Card allows the user to reinitialize the P89LV51RD2 and the I^2C devices and go to a known state. It causes the firmware to start again at the beginning point and initiates the dialing routine.

4.8 How to code I²C commands using the P89LV51RD2/PCA9564

I²C messages are described using a 'structure' type definition where the I²C address, the number of bytes to be sent/received and a pointer to a buffer with the data are used:

```
typedef struct
{
  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;
```

The user must then use a variable with an 'I2C_MESSAGE' type and a variable acting as a buffer that will be filled with the message to send (Write operation) or filled with the message received (Read operation).

Example 1: Program the PCA9531_Red with BR0 at (max frequency, 50 % duty cycle) and BR1 at (max frequency, 10 % duty cycle), with LD13 to LD16 blinking at BR0 and LD17 to LD20 blinking at BR1.

```
idata I2C_MESSAGE   Message1;
idata BYTE   Buffer1[16];

Message1.nrBytes  = 7;
Message1.buf       = Buffer1;
Message1.address  = 0xCA;      // I2C address PCA9531 Red (RGB LEDs)
Buffer1[0]         = 0x11;     // auto increment + register 1
Buffer1[1]         = 0x00;     // max frequency
Buffer1[2]         = 0x80;     // 50 % duty cycle
Buffer1[3]         = 0x00;     // max frequency
Buffer1[4]         = 0x19;     // 10 % duty cycle
Buffer1[5]         = 0x00;     // Red RGB LED's = off
Buffer1[6]         = 0x00;     // Red RGB LED's = off
I2C_Write(&Message1);          // Function sending the I2C sequence
```

Example 2: Read the PCA9555 Input port Registers. To perform this operation a Write to the device must been initiated first in order to provide the command code (or pointer information) to decide which register(s) needs to be read. Then a read is performed.

```
idata I2C_MESSAGE   Message2;
data I2C_MESSAGE   Message3;
idata BYTE   Buffer2[16];
idata BYTE   Buffer3[16];

Message2.nrBytes = 1;                 // The 1st message is 1 byte long
Message3.nrBytes = 2                  // The 2nd message is 2 bytes long;
Message2.buf     = Buffer2;
Message3.buf     = Buffer3;
Message2.address = 0x46;              // I2C address PCA9555 Write
Message3.address = 0x47;              // I2C address PCA9555 Read
Buffer2[0] = 0x00;                    // Set the command byte / pointer
                                         1st part of the message)
```

```
                I2C_WriteRepRead(&Message2,&Message3); // Function sending the I2C sequence
```

After the read is performed Buffer3[0] and Buffer3[1] contain the input port register values.

5. Source code

5.1 i2cexpert.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 Increment  0
#define Decrement  1

#define PCA9531_WR    0xC8        // i2c address LED Dimmer              - Write operation
#define PCA9531_RD    0xC9        // i2c address LED Dimmer              - Read operation
#define PCA9555_WR    0x46        // i2c address i/o expander            - Write operation
#define PCA9555_RD    0x47        // i2c address i/o expander            - Read operation
#define PCA9531_R_WR  0xCA        // i2c address LED Dimmer Red          - Write operation
#define PCA9531_R_RD  0xCB        // i2c address LED Dimmer Red          - Read operation
#define PCA9531_G_WR  0xCC        // i2c address LED Dimmer Green        - Write operation
```

```
#define PCA9531_G_RD    0xCD        // i2c address LED Dimmer Green   - Read operation
#define PCA9531_B_WR    0xCE        // i2c address LED Dimmer Blue    - Write operation
#define PCA9531_B_RD    0xCF        // i2c address LED Dimmer Blue    - Read operation
#define PCA9531_M_WR    0xC6        // i2c address LED Dimmer Misc    - Write operation
#define PCA9531_M_RD    0xC7        // i2c address LED Dimmer Misc    - Read operation
#define PCA9533_W_WR    0xC4        // i2c address LED Dimmer White   - Write operation
#define PCA9533_W_RD    0xC5        // i2c address LED Dimmer White   - Read operation
#define PCA24S08_WR     0xA0
#define PCA24S08_RD     0xA1


/**** 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 Init_White(void);
extern void Init_RGB(void);
extern void Init_Misc(void);
extern void Init_GPIO(void);
extern void GPIO_Interrupt_Handler(void);
extern void InsertDelay(unsigned char delayTime);

extern void Backlight_Programming(void);
extern void Fun_Pattern_Programming(void);
extern void Battery_Status(void);
extern void Dial_Number(void);
extern void Intro_Patterns(void);
extern void Fun_Pattern_Display (short int Red_Value, short int Green_Value, short int Blue_Value, short int
Speed_Value);
extern void Auto_Demo(void);
```

```
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 PCA9555_Int = P3^2;     // Interrupt PCA9555 mapped with LV51's P3[2]
sbit PCA9564_Reset      = P3^4;     // Reset PCA9564 mapped with LV51's P3[4]
```

5.2 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_MESSAGE  Message1;
idata I2C_MESSAGE  Message2;
idata I2C_MESSAGE  Message3;
idata I2C_MESSAGE  Message4;

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(2);                // PCA9564 reset time = 2 ms
  PCA9564_Reset = 1;

  AUXR = 2;                      // External memory space
  I2C_InitializeMaster(0x00);   // 330 kHz
}

void Init_White(void)
{
  Message1.buf     = Buffer1;
  Message1.nrBytes = 6;
```

```
    Message1.address  = PCA9533_W_WR;
    Buffer1[0]   = 0x11;            // autoincrement + register 1
    Buffer1[1]   = 0x00;            // default prescaler pwm0
    Buffer1[2]   = 0x10;            // default duty cycle for pwm0
    Buffer1[3]   = 0x00;            // default prescaler pwm1
    Buffer1[4]   = 0x10;            // default duty cycle for pwm1
    Buffer1[5]   = 0xAA;            // LD1-LD4 on at BR0 (White LEDs)
    I2C_Write(&Message1);
}


void Init_RGB(void)
{
    Message1.buf      = Buffer1;
    Message1.nrBytes  = 7;
    Message1.address  = PCA9531_R_WR;
    Buffer1[0]   = 0x11;            // autoincrement + register 1
    Buffer1[1]   = 0x00;            // default prescaler pwm0
    Buffer1[2]   = 0x6B;            // default duty cycle for pwm0
    Buffer1[3]   = 0x00;            // default prescaler pwm1
    Buffer1[4]   = 0x6B;            // default duty cycle for pwm1
    Buffer1[5]   = 0x00;            // Red RGB LED's = off
    Buffer1[6]   = 0x00;            // Red RGB LED's = off
    I2C_Write(&Message1);
    Message1.address = PCA9531_G_WR;
    Buffer1[0]   = 0x11;            // autoincrement + register 1
    Buffer1[1]   = 0x00;            // default prescaler pwm0
    Buffer1[2]   = 0x00;            // default duty cycle for pwm0
    Buffer1[3]   = 0x00;            // default prescaler pwm1
    Buffer1[4]   = 0x00;            // default duty cycle for pwm1
    Buffer1[5]   = 0x00;            // Green RGB LED's = off
    Buffer1[6]   = 0x00;            // Green RGB LED's = off
    I2C_Write(&Message1);
    Message1.address = PCA9531_B_WR;
    Buffer1[0]   = 0x11;            // autoincrement + register 1
    Buffer1[1]   = 0x00;            // default prescaler pwm0
    Buffer1[2]   = 0x25;            // default duty cycle for pwm0
    Buffer1[3]   = 0x00;            // default prescaler pwm1
    Buffer1[4]   = 0x25;            // default duty cycle for pwm1
    Buffer1[5]   = 0x00;            // Blue RGB LED's = off
    Buffer1[6]   = 0x00;            // Blue RGB LED's = off
    I2C_Write(&Message1);
}


void Init_Misc(void)
{
    Message1.buf      = Buffer1;
    Message1.nrBytes  = 7;
    Message1.address  = PCA9531_M_WR;
    Buffer1[0]   = 0x11;            // autoincrement + register 1
    Buffer1[1]   = 0x97;            // default prescaler pwm0
    Buffer1[2]   = 0x80;            // default duty cycle for pwm0
    Buffer1[3]   = 0x97;            // default prescaler pwm1 = 1 Hz
    Buffer1[4]   = 0x08;            // default duty cycle for pwm1 = 50%
    Buffer1[5]   = 0x00;            // Misc LED's = off
    Buffer1[6]   = 0x00;            // Misc LED's = off
    I2C_Write(&Message1);
}
```

```
void Init_GPIO(void)
{
  Message2.address  = PCA9555_WR;
  Message2.buf      = Buffer2;
  Message2.nrBytes  = 1;
  Buffer2[0]        = 0;           // subaddress = 0

  Message3.address  = PCA9555_RD;
  Message3.buf      = Buffer3;
  Message3.nrBytes  = 2;           // read 2 bytes
  Buffer3[0]        = 0xFF;
  Buffer3[1]        = 0xFF;
}


//******************************************************************************
// 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
  }
}


//******************************************************************************
// Determine which program the user wants to run
//******************************************************************************

static void Function_Select(void)
{
  if (Buffer3[1] == 0xBF)              // Push on F1 detected - enter LED programming mode
  {
    Message1.address  = PCA9531_M_WR;  // PCA9531 Misc to indicate LED programming mode
    Message1.nrBytes  = 2;
    Buffer1[0]        = 0x16;          // subaddress = 0x06
    Buffer1[1]        = 0xC0;          // LD12 blinking at BR1 --> Indicate LED programming mode active
    I2C_Write(&Message1);             // Program PCA9531 (2 bytes)
    while (Buffer3[1] != 0xDF)         // Loop as long as END button not pushed (programming mode active)
    {
      Buffer3[1] = 0xFF;                 // Clear Key F1 pushed
      GPIO_Interrupt_Handler();          // Check if a new key has been pushed (sub mode - function to be programmed)
      if (Buffer3[0] == 0xFE | Buffer3[0] == 0xFD | Buffer3[0] == 0xFB)
```

```
                                        // Key pushed = 1, 2, or 3 --> Fun pattern programming
      {
        Fun_Pattern_Programming();
      }
      if (Buffer3[0] == 0xF7)           // Key pushed = 4 --> Backlight programming
      {
        Backlight_Programming();
      }
    }
    Buffer3[1] = 0xFF;                 // Clear Key END pushed - leave programming mode
    Message1.address  = PCA9531_M_WR;  // PCA9531 Misc
    Message1.nrBytes  = 2;
    Buffer1[0]        = 0x16;          // subaddress = 0x16
    Buffer1[1]        = 0x00;          // Misc Green LED = off --> Indicate LED programming mode left
    I2C_Write(&Message1);             // Program PCA9531 (2 bytes)
  }
  if (Buffer3[1] == 0x7F)              // Push on F2 detected - enter simulation mode
  {
    Message1.address  = PCA9531_M_WR;  // PCA9531 Misc to indicate simulation mode
    Message1.nrBytes  = 2;
    Buffer1[0]        = 0x16;          // subaddress = 0x06
    Buffer1[1]        = 0x30;          // LD11 blinking at BR1 --> Indicate simulation mode active
    I2C_Write(&Message1);             // Program PCA9531 (2 bytes)
    while (Buffer3[1] != 0xDF)         // Loop as long as END button not pushed (programming mode active)
    {
      Buffer3[1] = 0xFF;               // Clear Key F2 pushed
      GPIO_Interrupt_Handler();        // Check if a new key has been pushed (sub mode - function to be programmed)
      if (Buffer3[0] == 0xFE)          // Key pushed = 1 --> Battery discharge emulation
      {
        Battery_Status();
      }
      if (Buffer3[0] == 0xF7)          // Key 4 pushed
      {
        Auto_Demo();
      }
    }
    Buffer3[1] = 0xFF;                 // Clear Key END pushed - leave simulation mode
    Message1.address  = PCA9531_M_WR;  // PCA9531 Misc to indicate LED programming mode
    Message1.nrBytes  = 2;
    Buffer1[0]        = 0x16;          // subaddress = 0x06
    Buffer1[1]        = 0x00;          // LD11 blinking at BR1 --> Indicate LED programming mode active
    I2C_Write(&Message1);             // Program PCA9531 (2 bytes)
  }
  if (Buffer3[0] != 0xFF | Buffer3[1] == 0xFE | Buffer3[1] == 0xFD | Buffer3[1] == 0xFB | Buffer3[1] == 0xF7)
  {
    Dial_Number();
    Buffer3[1] = 0xFF;          // Clear Key END pushed
  }
  Buffer3[1] = 0xFF;            // Clear Key END pushed
}
```

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

void main(void)
{
  Init_PCA9564();           // Initialization PCA9564
  Init_White();             // Initialization White LED's
  Init_RGB();               // Initialization RGB LED's
  Init_Misc();              // Initialization Misc LED's
  Init_GPIO();              // Initialization GPIO
  Intro_Patterns();         // Patterns displayed at power up

  while (1)
  {
    GPIO_Interrupt_Handler();
    Function_Select();       // Enter a specific mode (programming, dial a number ...)
  }
}
```

5.3   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"

unsigned char Data_Received;
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;

idata BYTE Snapshot_1_1st_Byte  = 0x0F;
idata BYTE Snapshot_1_2nd_Byte  = 0x0F;
idata BYTE Snapshot_2_1st_Byte  = 0x00;
idata BYTE Snapshot_2_2nd_Byte  = 0x00;
int Trigger_GPIO_Polling;
int GPIO_Polling_On = 0;     // Enable (1) or disable (0) the PCA9555 polling option - default = off
```

9397 750 14062

Application note **Rev. 01 — 7 January 2005** 23 of 45

```
void InsertBigDelay(void)
{
  InsertDelay(255);
  InsertDelay(255);
  InsertDelay(255);
  InsertDelay(255);
  InsertDelay(255);
}


//*****************************************************************************
// Program the 3 PCA9531 (R/G/B) with the same parameter(s)
//*****************************************************************************

void Write_RGB_Controller(void)
{
  Message1.address = PCA9531_R_WR;
  I2C_Write(&Message1);
  Message1.address = PCA9531_G_WR;
  I2C_Write(&Message1);
  Message1.address = PCA9531_B_WR;
  I2C_Write(&Message1);
}



//*****************************************************************************
// 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  = PCA9555_WR;
  Message2.buf      = Buffer2;
  Message2.nrBytes  = 1;
  Buffer2[0]        = 0;                 //subaddress = 0
  Message3.address  = PCA9555_RD;
  Message3.buf      = Buffer3;
  Message3.nrBytes = 2;                  // read 2 bytes

  if (PCA9555_Int==0)                         // Action on pushbutton detected
  {
    I2C_WriteRepRead(&Message2,&Message3);        // 1st read the PCA9555
    if (Buffer3[0] ==0xFF & Buffer3[1] ==0xFF);
    else
    {
      Snapshot_1_1st_Byte  = Buffer3[0];        // load the 1st read data (Byte 1) in a temp memory
      Snapshot_1_2nd_Byte  = Buffer3[1];        // load the 1st read data (Byte 2) in a temp memory
    }

    InsertDelay(255);
    InsertDelay(255);
    InsertDelay(255);
    I2C_WriteRepRead(&Message2,&Message3);        // 2nd read the PCA9555
    Snapshot_2_1st_Byte    = Buffer3[0];        // load the 2nd read data (Byte 1) in a temp memory
    Snapshot_2_2nd_Byte    = Buffer3[1];        // load the 2nd read data (Byte 2) in a temp memory
                                                // Compare the 2 read data in the temp memories
    if (Snapshot_1_1st_Byte == Snapshot_2_1st_Byte & Snapshot_1_2nd_Byte == Snapshot_2_2nd_Byte & GPIO_Polling_On == 1)
```

```
    {
      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_1st_Byte;         // Buffer loaded again with the initial push value
      Buffer3[1] = Snapshot_1_2nd_Byte;         // Buffer loaded again with the initial push value
    }
  }
  if (Trigger_GPIO_Polling == 1)                // Start Polling PCA9554 when permanent push detected
  {
    I2C_WriteRepRead(&Message2,&Message3);
  }
}


//******************************************************************************
// Pattern displayed at power up or after a reset
//******************************************************************************

void Intro_Patterns(void)
{
  Message1.nrBytes  = 7;
  Buffer1[0]    = 0x11;     // autoincrement + register 1
  Buffer1[1]    = 0x00;     // default prescaler pwm0
  Buffer1[2]    = 0x3B;     // default duty cycle for pwm0
  Buffer1[3]    = 0x00;     // default prescaler pwm1
  Buffer1[4]    = 0x01;     // default duty cycle for pwm1
  Buffer1[5]    = 0x00;     // Green RGB LED's = off
  Buffer1[6]    = 0x00;     // Green RGB LED's = off
  Message1.address = PCA9531_R_WR;
  I2C_Write(&Message1);
  Message1.address = PCA9531_G_WR;
  I2C_Write(&Message1);
  Message1.address = PCA9531_B_WR;
  I2C_Write(&Message1);
  Message1.nrBytes = 3;
  Buffer1[0]   = 0x15;
  Buffer1[1]   = 0x02;      // LD13 @ BR0
  Buffer1[2]   = 0x00;
  Write_RGB_Controller();
  InsertDelay(250);
  Buffer1[1]   = 0x0A;      // LD14 @ BR0
  Buffer1[2]   = 0x00;
  Write_RGB_Controller();
  InsertDelay(250);
  Buffer1[1]   = 0x20;      // LD15 @ BR0
  Buffer1[2]   = 0x00;
  Write_RGB_Controller();
  InsertDelay(250);
  Buffer1[1]   = 0xA0;      // LD16 @ BR0
  Buffer1[2]   = 0x00;
  Write_RGB_Controller();
  InsertDelay(250);
  Buffer1[1]   = 0x00;
  Buffer1[2]   = 0x02;      // LD17 @ BR0
  Write_RGB_Controller();
  InsertDelay(250);
```

```
  Buffer1[1]    = 0x00;
  Buffer1[2]    = 0x0A;      // LD18 @ BR0
  Write_RGB_Controller();
  InsertDelay(250);
  Buffer1[1]    = 0x00;
  Buffer1[2]    = 0x20;      // LD19 @ BR0
  Write_RGB_Controller();
  InsertDelay(250);
  Buffer1[1]    = 0x00;
  Buffer1[2]    = 0xA0;      // LD20 @ BR0
  Write_RGB_Controller();
  InsertDelay(250);
  Buffer1[1]    = 0x02;      // LD13 @ BR0
  Buffer1[2]    = 0x00;
  Write_RGB_Controller();
  InsertDelay(250);
  Buffer1[1]    = 0x00;      // off
  Buffer1[2]    = 0x00;      // off
  Write_RGB_Controller();
}


//****************************************************************************
// Function controlling number dial
// Number = 10 digits : xxx-xxx-xxxx
// Once dialed, SND button is pushed
//****************************************************************************

idata BYTE Key_Pushed;
short int Call      = 0;

void Dial_Number(void)
{
  int Nb_Key_Pressed   = 0;
  int One_To_Eight     = 0;
  int Nine_Zero        = 0;

  Call++;                         // When Call = even number, line is busy - When Call = odd number, line is not busy
  Message1.address = PCA9531_M_WR; // PCA9531 Misc
  Message1.nrBytes = 6;            // Reset the PCA9531 to its default programmed values
  Message1.buf     = Buffer1;
  Buffer1[0]       = 0x11;         // subaddress = 0x01
  Buffer1[1]       = 0x97;         // BR0 = 1 Hz
  Buffer1[2]       = 0x80;         // BR0 duty cycle = 50%
  Buffer1[3]       = 0x97;         // BR1 = 1 Hz
  Buffer1[4]       = 0x08;         // duty cycle BR1 = 50%
  Buffer1[5]       = 0x00;         // All 4 LEDs off
  I2C_Write(&Message1);            // Program PCA9531 (6 bytes)

  Message1.nrBytes = 2;
  while (Buffer3[1] != 0xDF)       // Loop as long as END button not pushed (call not ended)
  {
    if (Buffer3[0] != 0xFF & Nb_Key_Pressed < 11 & Buffer3[1] != 0xEF)  // Key pushed = 1, 2, 3, 4, 5, 6, 7, 8 and != SND
    {
      Key_Pushed = Buffer3[0];
      Nb_Key_Pressed++;
      One_To_Eight = 1;
      Nine_Zero = 0;
    }
```

```
if (Buffer3[1] != 0xFF & Nb_Key_Pressed < 11 & Buffer3[1] != 0xEF)  // Key pushed = 9, 0 and != SND
{
  Key_Pushed = Buffer3[1];
  Nb_Key_Pressed++;
  One_To_Eight = 0;
  Nine_Zero = 1;
}
if (Nb_Key_Pressed < 11 & Buffer3[1] != 0xEF & (Buffer3[0] != 0xFF | Buffer3[1] != 0xFF))
{
  Buffer3[0] = 0xFF;     // Clear key pushed
  Buffer3[1] = 0xFF;     // Clear key pushed
  Buffer1[0]= 0x15;      // subaddress PCA9531_M= 0x05
  switch (Key_Pushed)
  {
    case 0xFE:  if (One_To_Eight == 1)     // 1 pushed
                {
                  Buffer1[1] = 0x40;       // LD8 = on
                  I2C_Write(&Message1);    // Program PCA9531 (2 bytes)
                }
                if (Nine_Zero == 1)        // 9 pushed
                {
                  Buffer1[1] = 0x41;       // LD5 and LD8 = on
                  I2C_Write(&Message1);    // Program PCA9531 (2 bytes)
                }
                break;
    case 0xFD:  if (One_To_Eight == 1)     // 2 pushed
                {
                  Buffer1[1] = 0x10;       // LD7 = on
                  I2C_Write(&Message1);    // Program PCA9531 (2 bytes)
                }
                if (Nine_Zero ==1)         // 0 pushed
                {
                  Buffer1[1] = 0x00;       // LD5 to LD8 = off
                  I2C_Write(&Message1);    // Program PCA9531 (2 bytes)
                }
                break;
    case 0xFB:                             // 3 pushed
                Buffer1[1] = 0x50;         // LD7 and LD8 = on
                I2C_Write(&Message1);      // Program PCA9531 (2 bytes)
                break;
    case 0xF7:                             // 4 pushed
                Buffer1[1] = 0x04;         // LD6 = on
                I2C_Write(&Message1);      // Program PCA9531 (2 bytes)
                break;
    case 0xEF:                             // 5 pushed
                Buffer1[1] = 0x44;         // LD6 and LD8 = on
                I2C_Write(&Message1);      // Program PCA9531 (2 bytes)
                break;
    case 0xDF:                             // 6 pushed
                Buffer1[1] = 0x14;         // LD6 and LD7 = on
                I2C_Write(&Message1);      // Program PCA9531 (2 bytes)
                break;
    case 0xBF:                             // 7 pushed
                Buffer1[1] = 0x54;         // LD6, LD7 and LD8 = on
                I2C_Write(&Message1);      // Program PCA9531 (2 bytes)
                break;
    case 0x7F:                             // 8 pushed
                Buffer1[1] = 0x01;         // LD5 = on
```

```
                    I2C_Write(&Message1);       // Program PCA9531 (2 bytes)
                    break;
          }
     }
   if (Nb_Key_Pressed == 11 & Buffer3[1] != 0xEF)  // more than 10 keys pushed and SND not pushed yet
   {
     Buffer3[0]  = 0xFF;                        // Clear key pushed
     Buffer3[1]  = 0xFF;                        // Clear key pushed
     Buffer1[1]  = 0xAA;                        // LD5 to LD8 = BR0 to indicate that the 10 numbers have been dialed
     I2C_Write(&Message1);                      // Program PCA9531 (2 bytes)
     Nb_Key_Pressed++;
   }
   if (Buffer3[1] == 0xEF)                      // SND pushed: Send a call
   {
     Buffer1[1]      = 0x00;                    // LD5 to LD8 = off (dial number = done)
     I2C_Write(&Message1);                      // Program PCA9531  (2 bytes)
     Message1.nrBytes = 7;
     Buffer1[0]      = 0x11;                    // subaddress = 0x01
     Buffer1[1]      = 0x97;                    // BR0 = 1 Hz
     Buffer1[2]      = 0x80;                    // duty cycle BR0 = 50%
     Buffer1[3]      = 0x00;                    // max freq BR1
     Buffer1[4]      = 0xFF;                    // max duty cycle BR1
     Buffer1[5]      = 0xAA;                    // All 4 RGB LEDs blinking red
     Buffer1[6]      = 0xAA;                    // All 4 RGB LEDs blinking red
     if (Call & 0x01)                           // Busy signal
     {
       Message1.address = PCA9531_R_WR;         // PCA9531 Red
       I2C_Write(&Message1);                    // Program PCA9531 (7 bytes)
     }
     else                                       // Non Busy signal
     {
       Message1.address  = PCA9531_G_WR;        // PCA9531 Green
       I2C_Write(&Message1);                    // Program PCA9531 (7 bytes)
     }
   }
   Buffer3[0] = 0xFF;
   Buffer3[1] = 0xFF;
   GPIO_Interrupt_Handler();                    // Check if a key has been pushed
 }
 Message1.address  = PCA9531_M_WR;              // PCA9531 Misc
 Message1.nrBytes  = 2;
 Buffer1[1]      = 0x00;                        // LD5 to LD8 = off (dial number = done)
 I2C_Write(&Message1);                          // Program PCA9531  (2 bytes)
 Message1.nrBytes  = 3;
 Buffer1[2]      = 0x00;                        // All RGB LEDs = off
 Message1.address  = PCA9531_R_WR;              // PCA9531 Red to switch off Red LEDs
 I2C_Write(&Message1);                          // Program PCA9531 (3 bytes)
 Message1.address  = PCA9531_G_WR;              // PCA9531 Red to switch off Green LEDs
 I2C_Write(&Message1);                          // Program PCA9531 (3 bytes)
}
```

9397 750 14062

Application note **Rev. 01 — 7 January 2005** **28 of 45**

```
//****************************************************************************
// Function controlling the Duty Cycle for a specific device
// inputs  = Key "-", Key "+", I2C address
// output  = new Duty Cycle value
//****************************************************************************

short int  Duty_Cycle_Control(short int  Operation, short int I2C_Address, short int  Duty_Cycle_Value)
{
  Message1.address  = I2C_Address;
  Message1.nrBytes = 2;
  if (Operation == Decrement & Duty_Cycle_Value > 0x00)   // Key pushed = 2  - decrease brightness
  {
    Buffer1[0] = 0x02;                       // subaddress = 0x02
    Duty_Cycle_Value--;                      // decrement pwm 0
    Buffer1[1] = Duty_Cycle_Value;
    I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
  }
  if (Operation == Increment & Duty_Cycle_Value < 0xFF)   // Key pushed = 3 pushed - increase brightness
  {
    Buffer1[0]  = 0x02;                      // subaddress = 0x02
    Duty_Cycle_Value++;                      // increment pwm 0
    Buffer1[1]  = Duty_Cycle_Value;
    I2C_Write(&Message1);                    // send new data to PCA9531 (2 bytes)
  }
  Buffer3[0]  = 0xFF;                        // Clear Key Plus pushed
  Buffer3[1]  = 0xFF;                        // Clear Key Plus pushed (Key 9 only)
  return Duty_Cycle_Value;
}


//****************************************************************************
// Function controlling the Backlight programming
// Entered by pushing 1
// Key 2 = decrease brightness
// Key 3 = Increase brightness
// Leave the mode by pushing END
//****************************************************************************

idata BYTE Duty_Cycle_White;

void Backlight_Programming (void)
{
  Buffer3[0]        = 0xFF;                  // Clear Key 1 pushed
  Message1.address  = PCA9531_M_WR;          // PCA9531 Misc to indicate backlight programming mode
  Message1.nrBytes  = 2;
  Buffer1[0]        = 0x15;                  // subaddress = 0x15
  Buffer1[1]        = 0x40;                  // LD8 on --> backlight programming mode active
  I2C_Write(&Message1);
  GPIO_Polling_On   = 1;                     // Enable PCA9555 polling option (see GPIO_Interrupt_Handler function)
  Message2.address  = PCA9533_W_WR;          // Read the current brightness value from the PCA9533
  Message2.buf      = Buffer2;
  Message2.nrBytes  = 1;
  Buffer2[0]        = 0x02;                  // subaddress = 12
  Message3.address  = PCA9533_W_RD;
  Message3.buf      = Buffer3;
  Message3.nrBytes  = 1;                     // read 1 byte
  I2C_WriteRepRead(&Message2,&Message3);     // read PWM0 of the PCA9531
  Duty_Cycle_White  = Buffer3[0];
  while (Buffer3[1] != 0xDF)
```

```
  {
    GPIO_Interrupt_Handler();
    InsertDelay(100);
    if (Buffer3[0] == 0xFD) Duty_Cycle_White = Duty_Cycle_Control(Decrement, PCA9533_W_WR, Duty_Cycle_White);
                                          // "-" Red (Key 2)
    if (Buffer3[0] == 0xFB) Duty_Cycle_White = Duty_Cycle_Control(Increment, PCA9533_W_WR, Duty_Cycle_White);
                                          // "+" Red (Key 3)
  }
  Buffer3[1]      = 0xFF;                // Clear Key END pushed
  GPIO_Polling_On = 0;                  // Disable PCA9555 GPIO option
  Message1.address = PCA9531_M_WR;      // PCA9531 Misc to indicate LED programming mode
  Message1.nrBytes = 2;
  Buffer1[0]      = 0x15;               // subaddress = 0x15
  Buffer1[1]      = 0x00;               // LD8 off --> backlight programming mode left
  I2C_Write(&Message1);
}


//****************************************************************************
// Function displaying a selected Fun Pattern
// Inputs = Amount of Red, Green and Blue, Rotating Speed
//****************************************************************************

int Fun_Loop_Counter = 1;
int Speed_Prog_On    = 0;

void Fun_Pattern_Display( short int Red_Value, short int Green_Value, short int Blue_Value, short int Speed_Value)
{
  Message1.buf     = Buffer1;
  Message1.nrBytes = 2;
  Buffer1[0]      = 0x12;               // subaddress = 12
  Message1.address = PCA9531_R_WR;      // PCA9531 Red
  Buffer1[1]      = Red_Value;          // Programming Red
  I2C_Write(&Message1);                 // Program PCA9531 Red (2 bytes)
  Message1.address = PCA9531_G_WR;      // PCA9531 Green
  Buffer1[1]      = Green_Value;        // Programming Green
  I2C_Write(&Message1);                 // Program PCA9531 Green (2 bytes)
  Message1.address = PCA9531_B_WR;      // PCA9531 Blue
  Buffer1[1]      = Blue_Value;         // Programming Blue
  I2C_Write(&Message1);                 // Program PCA9531 Blue (2 bytes)
  Message1.nrBytes = 3;
  Buffer1[0]      = 0x15;               // subaddress = 15

  // Loop as long as a pushbutton not pressed
  while ((((Buffer3[0]==0xFF & Buffer3[1]==0xFF) | Buffer3[1] == 0xFD | Buffer3[1] == 0xF7) & Speed_Prog_On ==0)
  {
    if (Fun_Loop_Counter < 8)
    {
      Fun_Loop_Counter++;
    }
    else
    {
      Fun_Loop_Counter = 1;
    }
    switch (Fun_Loop_Counter)
    {
      case 1: Buffer1[1] = 0x02;        // Programming LD13  blinking at BR0
              Buffer1[2] = 0x00;        // LED's off
              break;
```

```
     case 2:  Buffer1[1]  = 0x08;              // Programming LD14  blinking at BR0
              Buffer1[2]  = 0x00;              // LED's off
              break;
     case 3:  Buffer1[1]  = 0x20;              // Programming LD15  blinking at BR0
              Buffer1[2]  = 0x00;              // LED's off
              break;
     case 4:  Buffer1[1]  = 0x80;              // Programming LD16  blinking at BR0
              Buffer1[2]  = 0x00;              // LED's off
              break;
     case 5:  Buffer1[1]  = 0x00;              // LED's off
              Buffer1[2]  = 0x02;              // Programming LD17  blinking at BR0
              break;
     case 6:  Buffer1[1]  = 0x00;              // Programming LD18  blinking at BR0
              Buffer1[2]  = 0x08;              // LED's off
              break;
     case 7:  Buffer1[1]  = 0x00;              // LED's off
              Buffer1[2]  = 0x20;              // Programming LD19  blinking at BR0
              break;
     case 8:  Buffer1[1]  = 0x00;              // LED's off
              Buffer1[2]  = 0x80;              // Programming LD20  blinking at BR0
              break;
   }
   Message1.address  = PCA9531_R_WR;           // PCA9531 Green
   I2C_Write(&Message1);                       // Program PCA9531 Red (3 bytes)
   Message1.address  = PCA9531_G_WR;           // PCA9531 Green
   I2C_Write(&Message1);                       // Program PCA9531 Green (3 bytes)
   Message1.address  = PCA9531_B_WR;           // PCA9531 Blue
   I2C_Write(&Message1);                       // Program PCA9531 Blue (3 bytes)
   InsertDelay(Speed_Value);                   // Programmable delay
   InsertDelay(Speed_Value);
   InsertDelay(Speed_Value);
   GPIO_Interrupt_Handler();
   if (Buffer3[1] == 0xFD | Buffer3[1] == 0xF7)  Speed_Prog_On = 1;
 }
 if ((Buffer3[0]!=0xFF | Buffer3[1]!=0xFF) & Buffer3[1] != 0xFD & Buffer3[1] != 0xF7)  // All the LEDs blinking at BR0
 {
   Message1.nrBytes  = 3;
   Buffer1[0]        = 0x15;                   // subaddress = 15
   Buffer1[1]        = 0xAA;                   // Programming All  LED's blinking at BR0
   Buffer1[2]        = 0xAA;                   // Programming All  LED's blinking at BR0
   Message1.address  = PCA9531_R_WR;           // PCA9531 Green
   I2C_Write(&Message1);                       // Program PCA9531 Red (3 bytes)
   Message1.address  = PCA9531_G_WR;           // PCA9531 Green
   I2C_Write(&Message1);                       // Program PCA9531 Green (3 bytes)
   Message1.address  = PCA9531_B_WR;           // PCA9531 Blue
   I2C_Write(&Message1);                       // Program PCA9531 Blue (3 bytes)
 }
}
```

```
//*****************************************************************************
// Function controlling the Fun Pattern programming
// 3 programmable patterns: color and speed
// Entered by pushing 4
// Key 1 = select pattern 1
// Key 4 = select pattern 2
// Key 7 = select pattern 3
// Key 2 = decrease red of selected pattern
// Key 3 = increase red of selected pattern
// Key 5 = decrease green of selected pattern
// Key 6 = increase green of selected pattern
// Key 8 = decrease blue of selected pattern
// Key 9 = increase blue of selected pattern
// Key 0 = decrease speed of selected pattern
// Key # = increase speed of selected pattern
//*****************************************************************************

idata BYTE Duty_Cycle_R_One   = 0x6B;
idata BYTE Duty_Cycle_G_One   = 0x01;
idata BYTE Duty_Cycle_B_One   = 0x25;

idata BYTE Duty_Cycle_R_Two   = 0x01;
idata BYTE Duty_Cycle_G_Two   = 0x6B;
idata BYTE Duty_Cycle_B_Two   = 0x25;

idata BYTE Duty_Cycle_R_Three = 0x26;
idata BYTE Duty_Cycle_G_Three = 0x6B;
idata BYTE Duty_Cycle_B_Three = 0x01;

int Speed_One   = 255;
int Speed_Two   = 25;
int Speed_Three = 100;

void Fun_Pattern_Programming (void)
{
  Init_RGB();
  GPIO_Interrupt_Handler();             // Check if an action on pushbutton happened
  if (Buffer3[0] == 0xFE)               // Pattern 1 selected - Key 1 pushed
  {
    Buffer3[0]        = 0xFF;           // Clear Key 1 pushed
    Message1.address  = PCA9531_M_WR;   // PCA9531 Misc to indicate backlight programming mode
    Message1.nrBytes  = 2;
    Buffer1[0]        = 0x15;           // subaddress = 0x15
    Buffer1[1]        = 0x01;           // LD5 on --> Pattern 1 programming active
    I2C_Write(&Message1);
    while (Buffer3[1]!=0xDF)            // Loop as long as END button not pushed (Fun pattern 1 programming active)
    {
      Buffer3[1] = 0xFF;
      GPIO_Interrupt_Handler();         // Check if an action on pushbutton happened
      GPIO_Polling_On = 1;              // Enable PCA9555 polling option (see GPIO_Interrupt_Handler function)
      Fun_Pattern_Display(Duty_Cycle_R_One, Duty_Cycle_G_One, Duty_Cycle_B_One, Speed_One);
      if (Buffer3[0] == 0xFD) Duty_Cycle_R_One = Duty_Cycle_Control(Decrement, PCA9531_R_WR, Duty_Cycle_R_One);
                                        // "-" Red (Key 2)
      if (Buffer3[0] == 0xFB) Duty_Cycle_R_One = Duty_Cycle_Control(Increment, PCA9531_R_WR, Duty_Cycle_R_One);
                                        // "+" Red (Key 3)
      if (Buffer3[0] == 0xEF) Duty_Cycle_G_One = Duty_Cycle_Control(Decrement, PCA9531_G_WR, Duty_Cycle_G_One);
                                        // "-" Green (Key 5)
```

```
      if (Buffer3[0] == 0xDF) Duty_Cycle_G_One = Duty_Cycle_Control(Increment, PCA9531_G_WR, Duty_Cycle_G_One);
                                          // "+" Green (Key 6)
      if (Buffer3[0] == 0x7F) Duty_Cycle_B_One = Duty_Cycle_Control(Decrement, PCA9531_B_WR, Duty_Cycle_B_One);
                                          // "-" Blue (Key 8)
      if (Buffer3[1] == 0xFE) Duty_Cycle_B_One = Duty_Cycle_Control(Increment, PCA9531_B_WR, Duty_Cycle_B_One);
                                          // "+" Blue (Key 9)
      if (Buffer3[1] == 0xFD & Speed_One > 0)
      {
        Speed_One--;
      }
      if (Buffer3[1] == 0xF7 & Speed_One < 255)
      {
        Speed_One++;
      }
      Speed_Prog_On  = 0;
      GPIO_Polling_On = 0;               // Disable PCA9555 GPIO Polling option

    }
    Buffer3[0]       = 0xFF;
    Buffer3[1]       = 0xFF;
    Message1.address = PCA9531_M_WR;     // PCA9531 Misc to indicate backlight programming mode
    Message1.nrBytes = 2;
    Buffer1[0]       = 0x15;             // subaddress = 0x15
    Buffer1[1]       = 0x00;             // LD5 off --> Pattern 1 programming left
    I2C_Write(&Message1);

  }

  if (Buffer3[0] == 0xFD)               // Pattern 2 selected - Key 2 pushed
  {
    Buffer3[0]       = 0xFF;            // Clear Key 2 pushed
    Message1.address = PCA9531_M_WR;     // PCA9531 Misc to indicate backlight programming mode
    Message1.nrBytes = 2;
    Buffer1[0]       = 0x15;             // subaddress = 0x15
    Buffer1[1]       = 0x04;             // LD6 on --> Pattern 2 programming active
    I2C_Write(&Message1);
    while (Buffer3[1]!=0xDF)             // Loop as long as END button not pushed (Fun pattern 2 programming active)
    {
      Buffer3[1] = 0xFF;
      GPIO_Interrupt_Handler();         // Check if an action on pushbutton happened
      GPIO_Polling_On = 1;              // Enable PCA9555 polling option (see GPIO_Interrupt_Handler function)
      Fun_Pattern_Display(Duty_Cycle_R_Two, Duty_Cycle_G_Two, Duty_Cycle_B_Two, Speed_Two);
      if (Buffer3[0] == 0xFD) Duty_Cycle_R_Two = Duty_Cycle_Control(Decrement, PCA9531_R_WR, Duty_Cycle_R_Two);
                                          // "-" Red (Key 2)
      if (Buffer3[0] == 0xFB) Duty_Cycle_R_Two = Duty_Cycle_Control(Increment, PCA9531_R_WR, Duty_Cycle_R_Two);
                                          // "+" Red (Key 3)
      if (Buffer3[0] == 0xEF) Duty_Cycle_G_Two = Duty_Cycle_Control(Decrement, PCA9531_G_WR, Duty_Cycle_G_Two);
                                          // "-" Green (Key 5)
      if (Buffer3[0] == 0xDF) Duty_Cycle_G_Two = Duty_Cycle_Control(Increment, PCA9531_G_WR, Duty_Cycle_G_Two);
                                          // "+" Green (Key 6)
      if (Buffer3[0] == 0x7F) Duty_Cycle_B_Two = Duty_Cycle_Control(Decrement, PCA9531_B_WR, Duty_Cycle_B_Two);
                                          // "-" Blue (Key 8)
      if (Buffer3[1] == 0xFE) Duty_Cycle_B_Two = Duty_Cycle_Control(Increment, PCA9531_B_WR, Duty_Cycle_B_Two);
                                          // "+" Blue (Key 9)
      if (Buffer3[1] == 0xFD & Speed_Two > 0)
      {
        Speed_Two--;
      }
```

```
    if (Buffer3[1] == 0xF7 & Speed_Two < 255)
    {
      Speed_Two++;
    }
    Speed_Prog_On = 0;
    GPIO_Polling_On = 0;                // Disable PCA9555 GPIO Polling option
  }
  Buffer3[1]      = 0xFF;
  Message1.address = PCA9531_M_WR;     // PCA9531 Misc to indicate backlight programming mode
  Message1.nrBytes = 2;
  Buffer1[0]      = 0x15;              // subaddress = 0x15
  Buffer1[1]      = 0x00;              // LD6 off --> Pattern 2 programming left
  I2C_Write(&Message1);
}

if (Buffer3[0] == 0xFB)               // Pattern 3 selected - Key 3 pushed
{
  Buffer3[0]      = 0xFF;             // Clear Key 3 pushed
  Message1.address = PCA9531_M_WR;    // PCA9531 Misc to indicate backlight programming mode
  Message1.nrBytes = 2;
  Buffer1[0]      = 0x15;             // subaddress = 0x15
  Buffer1[1]      = 0x10;             // LD7 on --> Pattern 3 programming active
  I2C_Write(&Message1);

  while (Buffer3[1]!=0xDF)            // Loop as long as END button not pushed (Fun pattern 1 programming active)
  {
    Buffer3[1] = 0xFF;
    GPIO_Interrupt_Handler();         // Check if an action on pushbutton happened
    GPIO_Polling_On = 1;              // Enable PCA9555 polling option (see GPIO_Interrupt_Handler function)
    Fun_Pattern_Display(Duty_Cycle_R_Three, Duty_Cycle_G_Three, Duty_Cycle_B_Three, Speed_Three);
    if (Buffer3[0] == 0xFD) Duty_Cycle_R_Three = Duty_Cycle_Control(Decrement, PCA9531_R_WR, Duty_Cycle_R_Three);
                                      // "-" Red (Key 2)
    if (Buffer3[0] == 0xFB) Duty_Cycle_R_Three = Duty_Cycle_Control(Increment, PCA9531_R_WR, Duty_Cycle_R_Three);
                                      // "+" Red (Key 3)
    if (Buffer3[0] == 0xEF) Duty_Cycle_G_Three = Duty_Cycle_Control(Decrement, PCA9531_G_WR, Duty_Cycle_G_Three);
                                      // "-" Green (Key 5)
    if (Buffer3[0] == 0xDF) Duty_Cycle_G_Three = Duty_Cycle_Control(Increment, PCA9531_G_WR, Duty_Cycle_G_Three);
                                      // "+" Green (Key 6)
    if (Buffer3[0] == 0x7F) Duty_Cycle_B_Three = Duty_Cycle_Control(Decrement, PCA9531_B_WR, Duty_Cycle_B_Three);
                                      // "-" Blue (Key 8)
    if (Buffer3[1] == 0xFE) Duty_Cycle_B_Three = Duty_Cycle_Control(Increment, PCA9531_B_WR, Duty_Cycle_B_Three);
                                      // "+" Blue (Key 9)
    if (Buffer3[1] == 0xFD & Speed_Three > 0)
    {
      Speed_Three--;
    }
    if (Buffer3[1] == 0xF7 & Speed_Three < 255)
    {
      Speed_Three++;
    }
    Speed_Prog_On = 0;
    GPIO_Polling_On = 0;              // Disable PCA9555 GPIO Polling option
  }                                   // end programming pattern 3 (END pushed and detected)
  Buffer3[1]      = 0xFF;
  Message1.address = PCA9531_M_WR;    // PCA9531 Misc to indicate backlight programming mode
  Message1.nrBytes = 2;
  Buffer1[0]      = 0x15;             // subaddress = 0x15
  Buffer1[1]      = 0x00;             // LD7 off --> Pattern 1 programming left
```

```
    I2C_Write(&Message1);

  }                                      // end if
  Message1.address  = PCA9531_R_WR;      // PCA9531 Red
  Message1.nrBytes  = 3;
  Buffer1[0]        = 0x15;              // subaddress = 15
  Buffer1[1]        = 0x00;              // all Red LED's off
  Buffer1[2]        = 0x00;              // all Red LED's off
  I2C_Write(&Message1);
  Message1.address  = PCA9531_G_WR;      // PCA9531 all Green LED's off
  I2C_Write(&Message1);
  Message1.address  = PCA9531_B_WR;      // PCA9531 all Blue LED's off
  I2C_Write(&Message1);
  Buffer3[1]        = 0xFF;              // Clear Key END pushed
}


//*****************************************************************************
// Function emulating a Battery Discharge
// Pushing Key 3 discharges the battery (level can be seen with LD5 to LD8
// Pushing 6 resets the emulation (battery fully charged again
//*****************************************************************************

void Battery_Status (void)
{
  int Battery_Level = 0xFF;

  Buffer3[0]        = 0xFF;              // Clear Key 1 pushed
  Message1.address  = PCA9531_M_WR;      // PCA9531 Misc
  Message1.nrBytes  = 7;
  Buffer1[0]        = 0x11;              // subaddress = 0x01
  Buffer1[1]        = 0x97;              // Blinking rate
  Buffer1[2]        = 0xF0;              // High Duty Cycle when Battery charge > 50%
  Buffer1[3]        = 0x97;              // default prescaler pwm1 = 1 Hz
  Buffer1[4]        = 0x08;              // default duty cycle for pwm1 = 50%
  Buffer1[5]        = 0x55;              // LD5 to LD8 on --> Indicate battery fully charged
  Buffer1[6]        = 0x32;              // RG LED Green blinking at BR0
  I2C_Write(&Message1);
  while (Buffer3[1]!=0xDF)               // Loop as long as END button not pushed (Fun pattern 2 programming active)
  {
    InsertDelay(150);
    Buffer3[1] = 0xFF;
    GPIO_Interrupt_Handler();           // Check if an action on pushbutton happened
    GPIO_Polling_On = 1;                // Enable PCA9555 polling option (see GPIO_Interrupt_Handler function)
    if (Buffer3[0] == 0xFB)             // Key 3 pushed - Battery is discharging when Key 3 pushed (continuous)
    {
      if (Battery_Level != 0x00) Battery_Level--;
      if (Battery_Level == 0xC0)
      {
        Message1.address  = PCA9531_M_WR; // PCA9531 Misc
        Message1.nrBytes  = 2;
        Buffer1[0]        = 0x15;         // subaddress = 0x05
        Buffer1[1]        = 0x54;         // LD5 now off
        I2C_Write(&Message1);
      }
      if (Battery_Level == 0x80)
      {
        Message1.address  = PCA9531_M_WR; // PCA9531 Misc
        Message1.nrBytes  = 6;
```

```
      Buffer1[0]       = 0x12;          // subaddress = 0x02
      Buffer1[1]       = 0x80;          // RG (Orange) LED shorter duty cycle
      Buffer1[2]       = 0x97;          // default prescaler pwm1 = 1 Hz
      Buffer1[3]       = 0x08;          // default duty cycle for pwm1 = 50%
      Buffer1[4]       = 0x50;          // LD5 and LD6 now off
      Buffer1[5]       = 0x3A;          // RG LED Green and Red blinking at BR0 (Orange)
      I2C_Write(&Message1);
    }
    if (Battery_Level == 0x40)
    {
      Message1.address  = PCA9531_M_WR; // PCA9531 Misc
      Message1.nrBytes  = 6;
      Buffer1[0]        = 0x12;          // subaddress = 0x02
      Buffer1[1]        = 0x10;          // RG (Red only) LED even shorter duty cycle
      Buffer1[2]        = 0x97;          // default prescaler pwm1 = 1 Hz
      Buffer1[3]        = 0x08;          // default duty cycle for pwm1 = 50%
      Buffer1[4]        = 0x40;          // LD5, LD6 and LD7 now off
      Buffer1[5]        = 0x38;          // RG LED Green and Red blinking at BR0
      I2C_Write(&Message1);
    }
    if (Battery_Level == 0x00)
    {
      Message1.address  = PCA9531_M_WR; // PCA9531 Misc
      Message1.nrBytes  = 6;
      Buffer1[0]        = 0x12;          // subaddress = 0x02
      Buffer1[1]        = 0x01;          // Duty Cycle = 0x01 --> Red LED actually almost off
      Buffer1[2]        = 0x97;          // default prescaler pwm1 = 1 Hz
      Buffer1[3]        = 0x08;          // default duty cycle for pwm1 = 50%
      Buffer1[4]        = 0x00;          // LD5, LD6, LD7 and LD8 now off
      Buffer1[5]        = 0x38;          // RG LED Green and Red blinking at BR0
      I2C_Write(&Message1);
    }
  }
  if (Buffer3[0] == 0xDF)               // Reset the simulation and recharge completly the battery (Key 6 pushed)
  {

    Battery_Level     = 0xFF;
    Message1.address  = PCA9531_M_WR;    // PCA9531 Misc
    Message1.nrBytes  = 6;
    Buffer1[0]        = 0x12;          // subaddress = 0x02
    Buffer1[1]        = Battery_Level - 0x10;  // High Duty Cycle when Battery charge > 50%
    Buffer1[2]        = 0x97;          // default prescaler pwm1 = 1 Hz
    Buffer1[3]        = 0x08;          // default duty cycle for pwm1 = 50%
    Buffer1[4]        = 0x55;          // LD5 to LD8 on --> Indicate battery fully charged
    Buffer1[5]        = 0x32;          // RG LED Green blinking at BR0
    I2C_Write(&Message1);
  }
  Buffer3[0] = 0xFF;                    // Clear Key 3 pushed
}
GPIO_Polling_On  = 0;                   // Disable PCA9555 polling option (see GPIO_Interrupt_Handler function)
Buffer3[1]      = 0xFF;
Message1.address = PCA9531_M_WR;        // PCA9531 Misc
Message1.nrBytes = 3;
Buffer1[0]      = 0x15;
Buffer1[1]      = 0x00;                 // LD5 to LD8 off
Buffer1[2]      = 0x30;
I2C_Write(&Message1);
}
```

```
//**************************************************************************
// Auto demo mode
// Reset only allows leaving this mode
//**************************************************************************

void Auto_Demo(void)
{
  int i;
  int j;
  int k;

  Message1.buf     = Buffer1;
  Message1.nrBytes = 7;

  Message1.address = PCA9531_R_WR;
  Buffer1[0]   = 0x11;     // autoincrement + register 1
  Buffer1[1]   = 0x00;     // default prescaler pwm0
  Buffer1[2]   = 0x6B;     // default duty cycle for pwm0
  Buffer1[3]   = 0x00;     // default prescaler pwm1
  Buffer1[4]   = 0x01;     // default duty cycle for pwm1
  Buffer1[5]   = 0x00;     // Green RGB LED's = off
  Buffer1[6]   = 0x00;     // Green RGB LED's = off
  I2C_Write(&Message1);
  Message1.address = PCA9531_G_WR;
  Buffer1[0]   = 0x11;     // autoincrement + register 1
  Buffer1[1]   = 0x00;     // default prescaler pwm0
  Buffer1[2]   = 0x01;     // default duty cycle for pwm0
  Buffer1[3]   = 0x00;     // default prescaler pwm1
  Buffer1[4]   = 0x6B;     // default duty cycle for pwm1
  Buffer1[5]   = 0x00;     // Green RGB LED's = off
  Buffer1[6]   = 0x00;     // Green RGB LED's = off
  I2C_Write(&Message1);
  Message1.address = PCA9531_B_WR;
  Buffer1[0]   = 0x11;     // autoincrement + register 1
  Buffer1[1]   = 0x00;     // default prescaler pwm0
  Buffer1[2]   = 0x25;     // default duty cycle for pwm0
  Buffer1[3]   = 0x00;     // default prescaler pwm1
  Buffer1[4]   = 0x25;     // default duty cycle for pwm1
  Buffer1[5]   = 0x00;     // Green RGB LED's = off
  Buffer1[6]   = 0x00;     // Green RGB LED's = off
  I2C_Write(&Message1);

  Message1.nrBytes = 3;
  Buffer1[0]   = 0x15;

  // Animation 1
  for (i = 0; i < 3; i++)
  {
    Buffer1[1]   = 0xEE;  // LD13 and 15 @ BR0 - LD14 and 16 @ BR1
    Buffer1[2]   = 0xEE;  // LD17 and 19 @ BR0 - LD18 and 20 @ BR1
    Write_RGB_Controller();
    InsertBigDelay();
    Buffer1[1]   = 0xBB;  // LD13 and 15 @ BR1 - LD14 and 16 @ BR0
    Buffer1[2]   = 0xBB;  // LD17 and 19 @ BR1 - LD18 and 20 @ BR0
    Write_RGB_Controller();
    InsertBigDelay();
  }
```

```
 // Animation 2
 Buffer1[1]  = 0xAA;  // LD13-16 @ BR0
 Buffer1[2]  = 0xAA;  // LD17-20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xAB;  // LD13 @ BR1 - LD14-16 @ BR0
 Buffer1[2]  = 0xAA;  // LD17-20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xAE;  // LD14 @ BR1 - LD13,14,16 @ BR0
 Buffer1[2]  = 0xAA;  // LD17-20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xBA;  // LD15 @ BR1 - LD13,14,16 @ BR0
 Buffer1[2]  = 0xAA;  // LD17-20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xEA;  // LD16 @ BR1 - LD13-15 @ BR0
 Buffer1[2]  = 0xAA;  // LD17-20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xAA;  // LD13-16 @ BR0
 Buffer1[2]  = 0xAB;  // LD17 @ BR1 - LD18-20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xAA;  // LD13-16 @ BR0
 Buffer1[2]  = 0xAE;  // LD18 @ BR1 - LD17,19,20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xAA;  // LD13-16 @ BR0
 Buffer1[2]  = 0xBA;  // LD19 @ BR1 - LD17,18,20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xAA;  // LD13-16 @ BR0
 Buffer1[2]  = 0xEA;  // LD18 @ BR1 - LD17,19,20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xAA;  // LD13-16 @ BR0
 Buffer1[2]  = 0xAA;  // LD17-20 @ BR0
 Write_RGB_Controller();
 InsertBigDelay();

// Animation 3
for (i = 0; i < 3; i++)
{
 Buffer1[1]  = 0xAB;  // LD13@BR1, LD14-15-16@BR0
 Buffer1[2]  = 0xAB;  // LD17@BR1, LD18-19-20@BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xAE;  // LD14@BR1, LD13-15-16@BR0
 Buffer1[2]  = 0xAE;  // LD18@BR1, LD17-19-20@BR0
 Write_RGB_Controller();
 InsertBigDelay();
 Buffer1[1]  = 0xBA;  // LD15@BR1, LD13-14-16@BR0
 Buffer1[2]  = 0xBA;  // LD19@BR1, LD17-18-20@BR0
 Write_RGB_Controller();
 InsertBigDelay();
```

```
    Buffer1[1]   = 0xEA;  // LD16@BR1, LD13-14-15@BR0
    Buffer1[2]   = 0xEA;  // LD20@BR1, LD17-18-19@BR0
    Write_RGB_Controller();
    InsertBigDelay();
  }


// Animation 4
Buffer1[1]     = 0x00;  // LD13-16     = off
Buffer1[2]     = 0x00;  // LD17-20     = off
Write_RGB_Controller();
for (i = 0; i < 3; i++)
{
  Message1.address = PCA9531_R_WR;
  Buffer1[1]   = 0x41;  // LD13,LD16   = red
  Buffer1[2]   = 0x10;  // LD19        = red
  I2C_Write(&Message1);
  Message1.address = PCA9531_G_WR;
  Buffer1[1]   = 0x04;  // LD14        = green
  Buffer1[2]   = 0x41;  // LD17,20     = green
  I2C_Write(&Message1);
  Message1.address = PCA9531_B_WR;
  Buffer1[1]   = 0x10;  // LD15        = blue
  Buffer1[2]   = 0x04;  // LD18        = blue
  I2C_Write(&Message1);
  InsertBigDelay();
  Message1.address = PCA9531_G_WR;
  Buffer1[1]   = 0x41;  // LD13,LD16   = green
  Buffer1[2]   = 0x10;  // LD19        = green
  I2C_Write(&Message1);
  Message1.address = PCA9531_B_WR;
  Buffer1[1]   = 0x04;  // LD14        = blue
  Buffer1[2]   = 0x41;  // LD17,20     = blue
  I2C_Write(&Message1);
  Message1.address = PCA9531_R_WR;
  Buffer1[1]   = 0x10;  // LD15        = red
  Buffer1[2]   = 0x04;  // LD18        = red
  I2C_Write(&Message1);
  InsertBigDelay();
  Message1.address = PCA9531_B_WR;
  Buffer1[1]   = 0x41;  // LD13,LD16   = blue
  Buffer1[2]   = 0x10;  // LD19        = blue
  I2C_Write(&Message1);
  Message1.address = PCA9531_R_WR;
  Buffer1[1]   = 0x04;  // LD14        = red
  Buffer1[2]   = 0x41;  // LD17,20     = red
  I2C_Write(&Message1);
  Message1.address = PCA9531_G_WR;
  Buffer1[1]   = 0x10;  // LD15        = green
  Buffer1[2]   = 0x04;  // LD18        = green
  I2C_Write(&Message1);
  InsertBigDelay();
}

Message1.nrBytes = 7;

Message1.address = PCA9531_R_WR;
Buffer1[0]   = 0x11;    // autoincrement + register 1
Buffer1[1]   = 0x00;    // default prescaler pwm0
```

```
Buffer1[2]   = 0x00;     // default duty cycle for pwm0
Buffer1[3]   = 0x00;     // default prescaler pwm1
Buffer1[4]   = 0x00;     // default duty cycle for pwm1
Buffer1[5]   = 0xAA;     // Green RGB LED's = BR0
Buffer1[6]   = 0xAA;     // Green RGB LED's = BR0
I2C_Write(&Message1);
Message1.address = PCA9531_G_WR;
Buffer1[0]   = 0x11;     // autoincrement + register 1
Buffer1[1]   = 0x00;     // default prescaler pwm0
Buffer1[2]   = 0x00;     // default duty cycle for pwm0
Buffer1[3]   = 0x00;     // default prescaler pwm1
Buffer1[4]   = 0x00;     // default duty cycle for pwm1
Buffer1[5]   = 0xAA;     // Green RGB LED's = BR0
Buffer1[6]   = 0xAA;     // Green RGB LED's = BR0
I2C_Write(&Message1);
Message1.address = PCA9531_B_WR;
Buffer1[0]   = 0x11;     // autoincrement + register 1
Buffer1[1]   = 0x00;     // default prescaler pwm0
Buffer1[2]   = 0x00;     // default duty cycle for pwm0
Buffer1[3]   = 0x00;     // default prescaler pwm1
Buffer1[4]   = 0x00;     // default duty cycle for pwm1
Buffer1[5]   = 0xAA;     // Green RGB LED's = BR0
Buffer1[6]   = 0xAA;     // Green RGB LED's = BR0
I2C_Write(&Message1);

Message1.nrBytes = 2;
Buffer1[0]   = 0x12;

// Red only from min to max brightness
Message1.address = PCA9531_R_WR;
for (i = 0x00; i < 0xFF; i++)
{
  Buffer1[1] = i;
  I2C_Write(&Message1);
  InsertDelay(40);
};
Buffer1[1]   = 0x00;     // No red
I2C_Write(&Message1);

// Green only from min to max brightness
Message1.address = PCA9531_G_WR;
for (j = 0x00; j < 0xFF; j++)
{
  Buffer1[1] = j;
  I2C_Write(&Message1);
  InsertDelay(40);
};
Buffer1[1]   = 0x00;     // No green
I2C_Write(&Message1);

// Blue only from min to max brightness
Message1.address = PCA9531_B_WR;
for (k = 0x00; k < 0xFF; k++)
{
  Buffer1[1] = k;
  I2C_Write(&Message1);
  InsertDelay(40);
};
```

9397 750 14062

Application note **Rev. 01 — 7 January 2005** **40 of 45**

```
Buffer1[1]  = 0x00;    // No blue
I2C_Write(&Message1);

// Some color mixing 1
Message1.address = PCA9531_R_WR;     // Program some red
Buffer1[1] = 0x36;
I2C_Write(&Message1);
Message1.address = PCA9531_G_WR;     // Program some green
Buffer1[1] = 0x2B;
I2C_Write(&Message1);
Message1.address = PCA9531_B_WR;
for (k = 0x00; k < 0xFF; k++)        // Increase amount of blue
{
  Buffer1[1] = k;
  InsertDelay(40);
  I2C_Write(&Message1);
};

// Some color mixing 2
Buffer1[1] = 0x0A;                   // Program some blue
I2C_Write(&Message1);
Message1.address = PCA9531_G_WR;     // Program some green
Buffer1[1] = 0x0A;
I2C_Write(&Message1);
Message1.address = PCA9531_R_WR;
for (i = 0x00; i <0xFF; i++)         // Increase amount of red
{
  Buffer1[1] = i;
  I2C_Write(&Message1);
  InsertDelay(40);
};

// Some color mixing 3
Buffer1[1] = 0x10;                   // Program some red
I2C_Write(&Message1);
Message1.address = PCA9531_B_WR;
Buffer1[1] = 0x0A;                   // Program some blue
I2C_Write(&Message1);
Message1.address = PCA9531_G_WR;
for (j = 0x00; j < 0xFF; j++)        // Increase amount of green
{
  Buffer1[1] = j;
  I2C_Write(&Message1);
  InsertDelay(40);
};

// Some color mixing 4
Buffer1[1] = 0x00;                   // No color
Message1.address = PCA9531_R_WR;
I2C_Write(&Message1);
Message1.address = PCA9531_G_WR;
I2C_Write(&Message1);
Message1.address = PCA9531_B_WR;
I2C_Write(&Message1);
Message1.address = PCA9531_R_WR;
for (i = 0x00; i < 0xFF; i++)        // Increase amount of red
{
  Buffer1[1] = i;
```

```
    I2C_Write(&Message1);
    InsertDelay(40);
};
Message1.address = PCA9531_G_WR;
for (j = 0x00; j < 0xFF; j++)        // Increase amount of green
{
    Buffer1[1] = j;
    I2C_Write(&Message1);
    InsertDelay(40);
};
Buffer1[1] = 0x00;                   // Remove Green
I2C_Write(&Message1);
Message1.address = PCA9531_B_WR;
for (k = 0x00; k < 0xFF; k++)        // Increase amount of blue
{
    Buffer1[1] = k;
    I2C_Write(&Message1);
    InsertDelay(40);
};
Message1.address = PCA9531_R_WR;
Buffer1[1] = 0x00;                   // Remove Red
I2C_Write(&Message1);
Message1.address = PCA9531_G_WR;
for (j = 0x00; j < 0xFF; j++)        // Increase amount of green
{
    Buffer1[1] = j;
    I2C_Write(&Message1);
    InsertDelay(40);
};
Buffer1[1] = 0x00;                   // Remove Green
I2C_Write(&Message1);
Message1.address = PCA9531_R_WR;
for (i = 0x00; i < 0xFF; i++)        // Increase amount of red
{
    Buffer1[1] = i;
    I2C_Write(&Message1);
    InsertDelay(40);
};
Buffer1[1] = 0x00;                   // Remove Red
I2C_Write(&Message1);
Message1.address = PCA9531_G_WR;
Buffer1[1] = 0xFF;                   // Max Green
I2C_Write(&Message1);
Message1.address = PCA9531_B_WR;
for (k = 0x00; k < 0xFF; k++)        // Increase amount of blue
{
    Buffer1[1] = k;
    I2C_Write(&Message1);
    InsertDelay(40);
};
Buffer1[1] = 0x00;                   // Remove Blue
I2C_Write(&Message1);
Message1.address = PCA9531_R_WR;
for (i = 0x00; i < 0xFF; i++)        // Increase amount of red
{
    Buffer1[1] = i;
    I2C_Write(&Message1);
    InsertDelay(40);
```

```
  };
  Message1.address = PCA9531_G_WR;
  for (j = 0x00; j < 0xFF; j++)          // Increase amount of green
  {
    Buffer1[1] = j;
    I2C_Write(&Message1);
    InsertDelay(40);
  };
}


// END OF THE I2C ROUTINES //
```

6. Disclaimers

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 in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status 'Production'), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence 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.

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.

7. Licenses

Purchase of Philips I^{2}C components

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

9397 750 14062

Application note **Rev. 01 — 7 January 2005** **44 of 45**

8. Contents

PHILIPS