Application Note AN2304

Using Cypress I^2C Port Expander with Flash Storage

Author: Andrew Smetana
Associated Project: No
Associated Part Family: CY8C95xx
PSoC Designer Version: Not Used
Associated Application Notes: CY8C95xx Data Sheet

Abstract

This Application Note demonstrates the basic features of the Cypress I/O Port Expander family of devices, CY8C95xx. The application examples are described with step-by-step explanation of the configuration process. Attention focuses on both the expander I^2C-logic interface characteristics and IC details. The information in this note, along with the CY8C95xx Data Sheet, will simplify system design of I^2C slave devices based on the Cypress I/O Expander.

Introduction

The I^2C interface was developed by Philips Electronics and expanded upon in various microelectronic systems. I^2C logic is popular in computing, consumer electronics, and telecommunications because it allows seamless communication between two systems through two signal lines, SDA (Serial DAta line) and SCL (Serial CLock line). The schematic of an example I^2C system is shown in Figure 1.

Every device connected to the I^2C bus has its own address and acts as a receiver or transmitter, depending on its designated function. The I^2C bus is a multi-master bus, meaning that there can be one or more bus masters.

Some of the advantages of using the I^2C bus Expander include:

- o In traditional parallel bus systems, the address, data and control lines are routed to each device, which increases the number of copper traces on the PCB and requires address decoders and glue-logic circuitry that connects to many devices. Using an I^2C bus interface instead of the parallel structure reduces copper traces, renders the system less susceptible to disturbance by electromagnetic interference and electrostatic discharge, as well as saves board area, increases reliability and reduces debugging time.
- o I^2C bus implementation is useful in cases where the microcontroller has a limited number of I/O pins. The multi-parallel interface can be organized for microprocessors using I^2C Expander devices.
- o I/O Expanders with I^2C logic reduces the burden of interrupts by aggregating all conditions into one interrupt.

The I/O Expander devices are designed to provide general-purpose, remote I/O expansion for most microcontroller families through a two-wire bidirectional Inter-Integrated Circuit (I^2C) bus. The CY8C95xx I/O Expander device, in addition to parallel I/Os, has PWM User Modules and EEPROM capabilities. These modules can emulate an independent bus slave device with its own address.

The capability of PWMs permits control of gradual power increases in heating appliances and electrical fans as well as regulation of incandescent lamp brightness, alarm volume, and so on. The EEPROM is convenient for storing current system configuration, user settings, calibration parameters etc.

The technique used to determine the Expander device addresses (Extendable Soft Addressing^{TM}) allows address ports to be used as GPIO ports in cases where there is no address port assignment.

All the above-mentioned characteristics of I/O Expander devices are discussed further in the Cypress CY8C95xx Data Sheet available on http://www.cypress.com. It is useful to have this document on hand as you explore the application examples.

All examples were tested using an open source program − LPT-to-I^2C adapter accessible at http://www.boerde.de/~matthias/delphi/. The adapter's schematic is shown in the Appendix of this document.

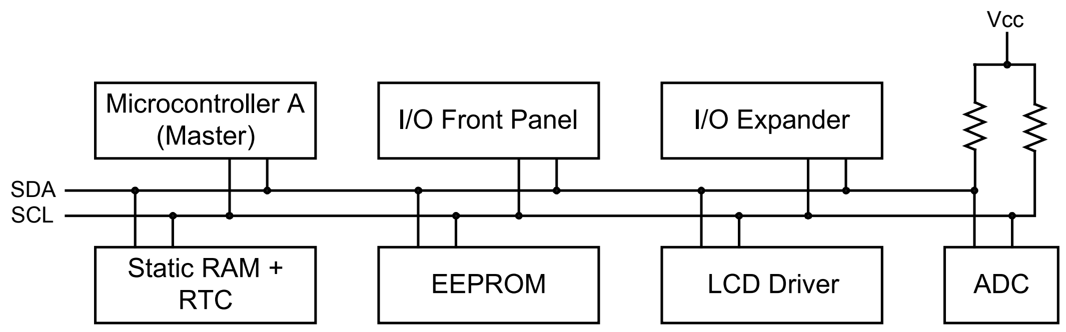

Figure 1. I^2C-Bus-Based System Example

Device Address Assignment

Each device on the I^2C bus must be configured to a unique address. The Expander has A_0−A_6 pins for addresses. By default there are two possible address formats in binary numbering: $010000A_0$ and $101000A_0$. The first is used to access the multi-port device and the second is used to access the EEPROM. If additional address lines (A_1-A_6) are used, the device addresses are defined as in the Device Addressing Table 1. This addressing method is called Extendable Soft Addressing^{TM}.

Table 1. Device Addressing

Multi-Port Device							
0	1	0	0	0	0	A_0	$R/\overline{W}$
0	1	0	0	0	A_1	A_0	$R/\overline{W}$
0	1	0	0	A_2	A_1	A_0	$R/\overline{W}$
0	1	0	A_3	A_2	A_1	A_0	$R/\overline{W}$
0	1	A_4	A_3	A_2	A_1	A_0	$R/\overline{W}$
0	A_5	A_4	A_3	A_2	A_1	A_0	$R/\overline{W}$
A_6	A_5	A_4	A_3	A_2	A_1	A_0	$R/\overline{W}$
EEPROM							
1	0	1	0	0	0	A_0	$R/\overline{W}$
1	0	1	0	0	A_1	A_0	$R/\overline{W}$
1	0	1	0	A_2	A_1	A_0	$R/\overline{W}$
1	0	1	A_3	A_2	A_1	A_0	$R/\overline{W}$
1	0	A_4	A_3	A_2	A_1	A_0	$R/\overline{W}$
1	A_5	A_4	A_3	A_2	A_1	A_0	$R/\overline{W}$
A_6	A_5	A_4	A_3	A_2	A_1	A_0	$R/\overline{W}$ [1]

[1] When all 7 address lines are used to define the I^2C device address, EEPROM is accessed through registers of the multi-port device.

Pin A_0 must be set as the LSb of the address port in each application. The other address pins are used if necessary. The last used address pin must be strong pull-up or strong pull-down (wired through a 300-Ω, or less, resistor to Vdd or Vss). All pins used before the last should be weak pull-up or weak pull-down (connected to Vdd or Vss through a 75 - 200 kΩ resistor).

For example, let's define an Expander that uses only 5 address pins (A_4-A_0). Figure 2 illustrates this Expander's schematic with hardware address assignments.

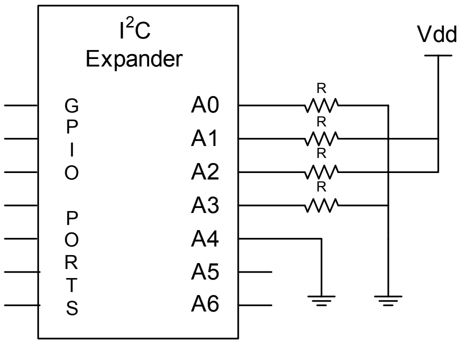

Figure 2. Expander Address Assignment Example

- R = 75 - 200 kΩ.
- Pin A_4 is last in the address pin chain and is strong pull-down meaning that $A_4 = 0$.
- A_0−A_3 are weak pull-up or pull-down, accordingly. A_5 and A_6 are not used so they can be utilized as GPIO ports.

What is the expander's full address? The 5 low bits (A_4−A_0) are known, and the 2 high bits (A_6,A_5) must be set according to the Device Addressing Table 1. A_5=1 and A_6=0. The full device address is $A_6A_5A_4A_3A_2A_1A_0$ = 0100110.

This technique works well, but when there are situations where **all** addressing pins (A_6−A_0) are used to assign a device address, EEPROM is accessed through registers of the multi-port device.

Access to the devices is performed according to the data packet. Consider the data addressing packets shown in Figure 3. The top packet in Figure 3a shows access to EEPROM. The bottom packet in Figure 3b shows access to the multi-port device. The second byte of the EEPROM packet is what distinguishes these two data packets. When all address lines A_6−A_0 are used, the device being accessed is defined by this second byte. If the most significant bit (MSb) of this byte is '0', this byte is treated as a command (register address) of the multi-port device. If the MSb is '1', this byte is the first of a 2-byte EEPROM address.

Let's briefly review some transactions between the master and I^2C Expander's devices. Assume the Expander's address is 1111111_2=$7F_{16}$. Consider the following transactions:

- sFE 00 sFF x
- sFE 2D 43 4D 53 02
- sFE 80 00 10 20 30
- sFE 80 00 sFF x x x

's' is a start condition. 'x' is a byte to be read from the Expander.

These transactions may be slightly difficult to understand now, but everything is clarified ahead in this Application Note.

The first line is addressed to the multi-port device. The command selects 00 register (input port 0), after that, sFF x reads port state.

The second line enables operation of EEPROM. There is a detailed description of EEPROM ahead in this document.

In the third line, data 10 20 30 is written to EEPROM starting at address 0000. The next transaction reads previously stored data. The MSb='1' in the address field of the second byte (80) indicates the EEPROM data packet. In the multi-port device this bit is '0'.

S	A6	A5	A4	A3	A2	A1	A0	R/W	A	Address High	A	Address Low	A	...

Figure 3a. EEPROM

S	A6	A5	A4	A3	A2	A1	A0	R/W	A	Register Addr	A	...

Figure 3b. Multi-Port Device

Figure 3. Data Addressing Packets

I^{2}C Expander Examples

1-kHz (50% Duty Cycle) Signal Generator

This example is for test purposes and demonstrates how to configure the device to generate beeps (or other indicating sounds). To configure a selected pin (for example, GPort0_Bit3_PWM1) as a 1-kHz generator, execute the following:

1. Select PWM1 using the PWM Select register (28h).
2. Select 93.75 kHz as clock source (Config PWM register (29h)).
3. Set $Period = \frac{t_{OUT}}{t_{CLK}} = \frac{1/1\,kHz}{1/93.75\,kHz} \approx 94 = 5\mathrm{Eh}$ (Period PWM register (2Ah)).
4. Set $PulseWidth = DutyCycle \cdot Period = 0.5 \cdot 94 = 47 = 2Fh$ (Pulse Width register (2Bh)).
5. Select GPort 0 using Port Select register (18h).
6. Configure GPort0_Bit3_PWM1 as output (Pin Direction register (1Ch)).
7. Set GPort0_Bit3_PWM1 as PWM output (Select PWM register (1Ah)).

Each of these steps can be performed by one I^{2}C transaction as shown in Table 2.

Table 2. 1-kHz Signal Generator Device Configuration Steps

Data	Description
s	I^{2}C start condition
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
28h	PWM Select register address
01h	Select PWM1
03h	Config PWM register (select 93.75 kHz)
5Eh	Period
2Fh	Pulse width
s	Restart
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
18h	Port Select register address
00h	Select GPort 0
s	Restart
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
1Ch	Pin Direction register address
00h	Set all pins of GPort 0 as output
s	Restart
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
1Ah	Select PWM register address
08h	Set GPort0_Bit3_PWM1 as PWM output
p	I^{2}C stop condition

1-MHz (10% Duty Cycle) Square Wave Generator

To configure a selected pin (for example, GPort1_Bit6_PWM2) as a 1-MHz 10% duty cycle square wave generator, execute the following:

1. Select PWM2 using PWM Select register (28h).
2. Select 24 MHz as clock source (Config PWM register (29h)).
3. Set $Period = \frac{t_{OUT}}{t_{CLK}} = \frac{1/1\,MHz}{1/24\,MHz} = 24 = 18\mathrm{h}$ (Period PWM register (2Ah)).
4. Set $PulseWidth = DutyCycle \cdot Period = 0.1 \cdot 24 \approx 2 = 02\mathrm{h}$ (Pulse Width register (2Bh)).
5. Select GPort 1 using Port Select register (18h).
6. Configure GPort1_Bit6_PWM2 as output (Pin Direction register (1Ch)).
7. Set GPort1_Bit6_PWM2 as PWM output (Select PWM register (1Ah)).
8. Set GPort1_Bit6_PWM2 pin to Strong drive mode (21h register).

Each of these steps can be performed by one I^{2}C transaction as shown in Table 3.

Table 3. 1-MHz Square Wave Generator Device Configuration Steps

Data	Description
S	I^{2}C start condition
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
28h	PWM Select register address
02h	Select PWM2
01h	Config PWM register (select 24 MHz)
18h	Period
02h	Pulse width
S	Restart
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
18h	Port Select register address
01h	Select GPort 1
S	Restart
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
1Ch	Pin Direction register address
00h	Set all pins of GPort 1 as output
S	Restart
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
1Ah	Select PWM register address
40h	Set GPort1_Bit6_PWM2 as PWM output
S	Restart
40h	Device address $\mathrm{R}/\overline{\mathrm{W}} = 0$
21h	Strong drive mode register address
40h	Set GPort1_Bit6_PWM2 pin Strong drive
P	I^{2}C stop condition

Blinking LED (Hardware Implementation)

This example demonstrates how to configure the device to get a 1-Hz blink rate on an LED. In contrast to the upcoming example (Blinking LEDs using ISRs and Master), this example implements blinking LEDs using only the hardware capabilities of the Expander.

Assume the LED is connected to the GPort0_Bit3_PWM1 pin and to GND via a resistor (the pin is set to Strong drive mode). We chain two PWMs in series so that PWM0 drives PWM1 and the output of PWM1 is connected to the LED. To configure a 1-Hz blink rate on an LED, execute the following:

1. Set Programmable Divider to 127 using register (2Ch).
2. Select PWM0 using PWM Select register (28h).
3. Select programmable frequency (configured to 738.2 Hz) as clock source (Config PWM register (29h)).
4. Set Period = 37 = 25h, $t_{OUT\_PWM0} = \frac{1}{738.2} \cdot 37 \approx 0.05\ s$ (Period PWM register (2Ah)).
5. Set arbitrary allowed Pulse Width, for example, 18 = 12h (Pulse Width register (2Bh)).
6. Select PWM1 using PWM Select register (28h).
7. Select previous PWM as clock source (Config PWM register (29h)).
8. Set Period = 20 = 14h, $t_{OUT\_PWM1} = 0.05 \cdot 20 = 1.0\ s$ (Period PWM register (2Ah)).
9. Set 50% Pulse Width equal to 0Ah (Pulse Width register (2B h)).
10. Select GPort 0 using Port Select register (18h).
11. Configure GPort0_Bit3_PWM1 as output (Pin Direction register (1Ch)).
12. Set Strong drive mode for GPort0_Bit3_PWM1 (register 20h).
13. Set GPort0_Bit3_PWM1 as PWM output (Select PWM register (1Ah)).

Table 4. Blinking LED (Hardware Implementation) Device Configuration Steps

Data	Description
S	I^{2}C start condition
40h	Device address $R/\overline{W} = 0$
28h	PWM Select register address
00h	Select PWM0
04h	Config PWM register (738.2 Hz)
25h	Period
12h	Pulse width
7Fh	Divider register
S	I^{2}C start condition
40h	Device address $R/\overline{W} = 0$
28h	PWM Select register address
01h	Select PWM1
05h	Config PWM register (previous PWM)
14h	Period
0Ah	Pulse width
S	Restart
40h	Device address $R/\overline{W} = 0$
18h	Port Select register address
00h	Select GPort 0
S	Restart
40h	Device address $R/\overline{W} = 0$
1Ch	Pin Direction register
00h	Pin Direction – all output
S	Restart
40h	Device address $R/\overline{W} = 0$
1Ah	Select PWM for output register
08h	Set PWM output to GPort0_Bit3_PWM1
S	Restart
40h	Device address $R/\overline{W} = 0$
21h	Drive Mode Strong register
08h	Set GPort0_Bit3_PWM1 Strong mode
P	I^{2}C stop condition

Blinking LED using ISRs and Master

This example demonstrates how to configure the device to set up the ISRs in order to get a 0.5 Hz LED winking rate for 10 blinks.

Assume the LED is connected to the GPort0_Bit3_PWM1 pin and to Vdd via a resistor. This pin should be set to Open Drain Low drive mode. GPort0_Bit2_PWM3 is used as PWM output in order to generate the INT signal. To configure this device, execute the following:

1. Set Programmable Divider to 255 (mandatory to enable PWM interrupt) using register (2Ch).
2. Select PWM3 using the PWM Select register (28h).
3. Select 367.6 Hz as clock source (mandatory to enable PWM interrupt) (Config PWM register (29h)).
4. Set Period = 37 = 25h, $t_{OUT} = \frac{1}{367.6} \cdot 37 \approx 0.1\,s$ (Period PWM register (2Ah)).
5. Set arbitrary Pulse Width, for example, 18 = 12h (Pulse Width register (2Bh)).
6. Select GPort 0 using Port Select register (18h).
7. Configure GPort0_Bit3_PWM1 and GPort0_Bit2_PWM3 as output (Pin Direction register (1Ch)).
8. Set GPort0_Bit3_PWM1 to Open Drain Low drive mode (register 20h).
9. Enable the interrupt from GPort0_Bit2_PWM3 in the Interrupt Mask register (19h).
10. Set GPort0_Bit2_PWM3 as PWM output (Select PWM register (1Ah)).

Table 5. Blinking LED (using ISRs and Master) Device Configuration Steps

Data	Description
S	I^2C start condition
40h	Device address $R/\overline{W} = 0$
28h	PWM Select register address
03h	Select PWM3
04h	Config PWM register (select 367.6 Hz)
25h	Period
12h	Pulse width
FFh	Divider register
S	Restart
40h	Device address $R/\overline{W} = 0$
18h	Port Select register address
00h	Select GPort 0
FBh	Interrupt Mask, unmask bit 2
04h	Set PWM output to GPort0_Bit2_PWM3
00h	Inversion
00h	Pin Direction – all output
S	Restart
40h	Device address $R/\overline{W} = 0$
20h	Open Drain Low register address
08h	GPort0_Bit3_PWM1 is Open Drain Low
P	I^2C stop condition

Configured this way, the device generates an INT signal 10 times per second. The master device handles these interrupt requests, determines if each interrupt was caused by the PWM, and then counts the interrupts. As soon as 10 interrupts have been reached, the master issues a command to toggle the GPort0_Bit3_PWM1 pin output.

The following simplified algorithm for the master device is recommended:

1. Configure the slave device (see Table 5).
2. Set LedOutput = 08h, Periods = 0.
3. Send LedOutput to the slave device (s 40h 08h LedOutput).
4. Set IntCount = 0.
5. Wait for INT request.
6. Read all Interrupt Status registers (s 40h 10h s 41h x x x x x x x x).
7. Check interrupt source: if (Interrupt status Port 0) and 04h == 0 then GoTo 5.
8. Increment IntCount = IntCount + 1.
9. If IntCount < 10 then GoTo 5.
10. Toggle LedOutput = LedOutput XOR 08h.
11. Periods + = 1.
12. if (Periods == 20) then STOP.
13. GoTo 3.

This example also demonstrates “burst“ writing and reading procedures. If sequential registers have to be read or written to, it is sufficient to address the first register and perform sequential read/writes. Similarly, reading of the Interrupt Mask registers (item 6 in the previous algorithm) is performed in one cycle. If the “burst“ of read/writes is performed, each data byte is acted upon independently before the next byte is received (written to) or sent (read).

EEPROM

The Expander’s EEPROM serves two purposes: non-volatile storage of multi-port device configurations and ordinary EEPROM storage. The ordinary EEPROM storage is considered an individual I^{2}C bus slave device and has its own address. This non-volatile memory can be used to store data. After the Expander is powered on, the non-volatile memory is not seen on the I^{2}C bus, but can be enabled by sending a sequence of commands to the multi-port device address. The non-volatile storage of multi-port device configurations is for storing the Expander’s user-determined configuration, which is automatically loaded after power on.

Execute the following sequence of steps to enable EEPROM storage and perform read and write operations:

Table 6. Enable EEPROM Operation

Data	Description
S	I^{2}C start condition
40h	Device address $R/\overline{W} = 0$
2Dh	Select Enable register
43h	Send byte ‘C’
4Dh	Send byte ‘M’
53h	Send byte ‘S’
02h	Enable EEPROM, disable WD pin
P	I^{2}C stop condition

Table 7. EEPROM Write Operation

Data	Description
S	I^{2}C start condition
A0h	EEPROM device address $R/\overline{W} = 0$
00h	A_{HI} – Address high byte
03h	A_{LO} – Address low byte
07h	Byte writing to EEPROM
08h	Byte writing to EEPROM
09h	Byte writing to EEPROM
0Ah	Byte writing to EEPROM
0Bh	Byte writing to EEPROM
P	I^{2}C stop condition

Table 8. EEPROM Read Operation

Data	Description
S	I^{2}C start condition
A0h	EEPROM device address $R/\overline{W} = 0$
00h	A_{HI} – Address high byte
03h	A_{LO} – Address low byte
S	I^{2}C start condition
A1	EEPROM device address $R/\overline{W} = 1$
X	Byte to be read from EEPROM
X	Byte to be read from EEPROM
X	Byte to be read from EEPROM
X	Byte to be read from EEPROM
X	Byte to be read from EEPROM
P	I^{2}C stop condition

According to these steps, first, you enable EEPROM operation, second, write data to EEPROM starting at address 0003h, and last, read previously written data. Read and write operations are based byte-by-byte. Data ‘X’ in Table 8 means that in this position a byte will be returned from EEPROM.

EEPROM multi-port device configurations are accessed through the Command register (30h). This register sends commands to the device, which include configuration as new POR defaults, restore factory defaults, define POR defaults, read POR defaults, write device configuration, read device configuration, and reconfigure device with stored POR defaults. Let’s consider a typical example for use of this register.

Imagine that you have configured your Expander device with the Blinking LED (Hardware Implementation) settings. Typically, after each power on, the master reloads this configuration. To make this automatic, load the necessary device settings and execute the steps in Table 9. After that, your I^{2}C device will be self-configurable after power on.

Table 9. Storing Device Configuration to EEPROM POR Defaults

Data	Description
S	I^{2}C start condition
40h	Device address $R/\overline{W} = 0$
30h	Select Command register
01h	Store device configuration
P	I^{2}C stop condition

To restore factory defaults, execute the following steps:

Table 10. Restoring Factory Defaults

Data	Description
S	I^{2}C start condition
40h	Device address $R/\overline{W} = 0$
30h	Select Command register
02h	Restore factory defaults
P	I^{2}C stop condition

Front Panel Display using I^{2}C Expander

A front panel display is a typical Expander application. Utilization of an expander device for a front panel display is suitable for several reasons:

- It occupies only two digital lines on the microprocessor to control several I/O Expander pins, giving the ability to create various indicating systems, keyboard, data display, etc. Note that these interface lines can be used by many other I^{2}C devices, not just the Expander.
- It simplifies access to Expander resources. The register-based internal architecture allows direct control of integrated resources.
- It has an efficient interrupt mechanism. The processor can carry out useful tasks without polling the device about incoming events. The Expander utilizes only one interrupt input line wired to its INT output pin to trigger interrupts.
- PWM User Modules can be useful to control gradual brightness of lights or volume of alarms in indicating systems.
- The presence of EEPROM makes it possible to save panel settings in non-volatile memory and return to these settings after power on. EEPROM can be used as ordinary data storage in the system. Also, integrated Expander EEPROM scales down PCB area in cases of its (EEPROM) necessity.

A simple front panel display is shown in Figure 4.

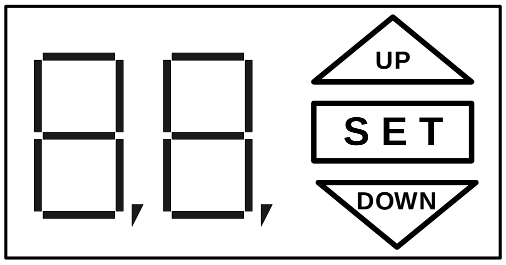

Figure 4. Front Panel Display

This display consists of two LED-based, 7-segment decimal point displays and three control buttons. The UP button increments the display value, the DOWN button decrements the display value. The SET button stores the current display value in EEPROM. After the Expander powers on, this stored value is loaded into the display.

A simplified schematic of an Expander is shown in Figure 5.

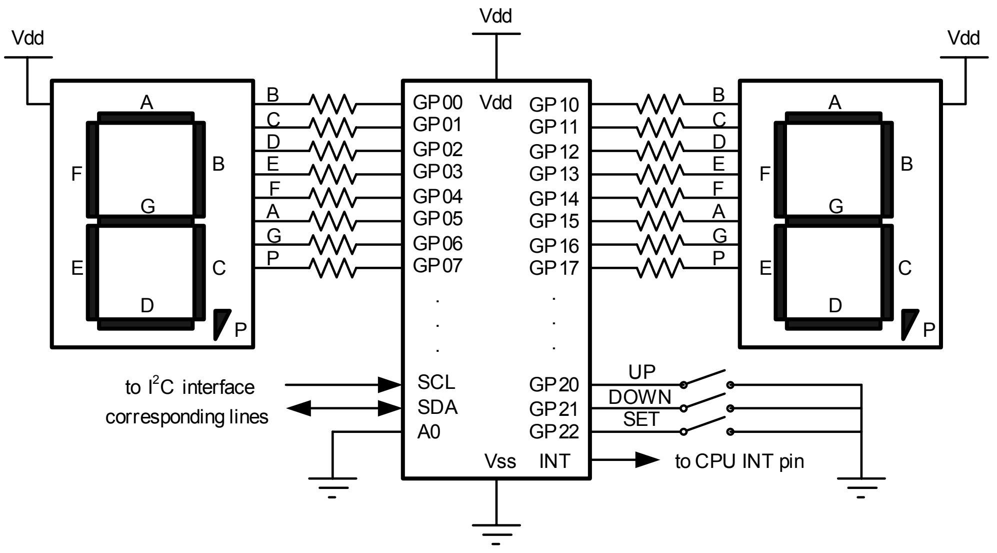

Figure 5. Simplified Application Schematic

In the schematic, the multi-port device address is 0100000 and the EEPROM address is 1010000.

The static indicating system utilizes two ports, 0 and 1. The drive mode for these port pins is configured as Open Drain Low. The drive mode for the button's pins must be configured as Pull Up.

The proposed application is based on the CPU processing interrupts. The interrupt requests (IRQs) are sent to the CPU after any button changes state. In order to recognize which button has triggered the IRQ and determine interrupt signal edge, the CPU reads two registers, the Port 2 Interrupt Status register and Input Port 2 register. This information is enough for the system to make a decision and continue accordingly.

All algorithmic work concerning panel processing activity is executed by the I^2C master, in this case, the CPU. The master's algorithm may be implemented as shown in Figure 6.

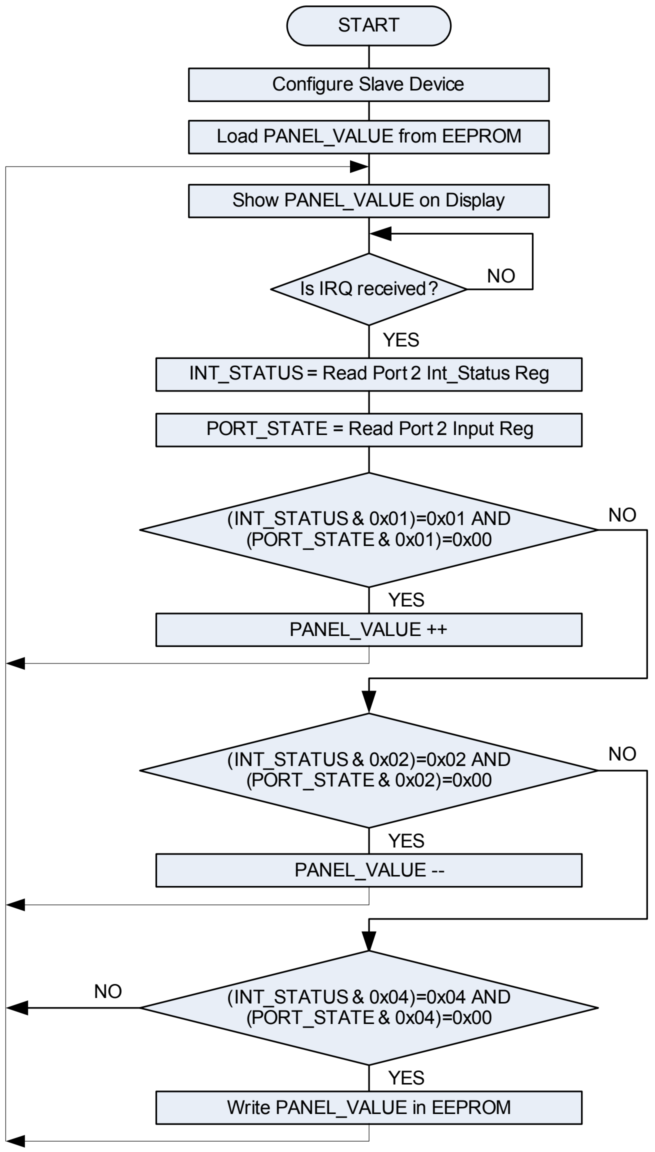
START
Configure Slave Device
Load PANEL_VALUE from EEPROM
Show PANEL_VALUE on Display
Is IRQ received?
NO
YES
INT_STATUS = Read Port 2 Int_Status Reg
PORT_STATE = Read Port 2 Input Reg
(INT_STATUS & 0x01)=0x01 AND (PORT_STATE & 0x01)=0x00
NO
YES
PANEL_VALUE ++
(INT_STATUS & 0x02)=0x02 AND (PORT_STATE & 0x02)=0x00
NO
YES
PANEL_VALUE --
NO
(INT_STATUS & 0x04)=0x04 AND (PORT_STATE & 0x04)=0x00
YES
Write PANEL_VALUE in EEPROM

Figure 6. Simplified Application Flowchart

The processing blocks on the flowchart contain functions that communicate with the Expander. Below, these blocks are detailed on the I^2C bus transaction level.

Configure Slave Device

- s40 18 00 FF 00 00 00;
  port 0 configuration
- s40 20 FF;
  set Open Drain Low drive mode (port 0)
- s40 18 01 FF 00 00 00;
  port 0 configuration
- s40 20 FF;
  set Open Drain Low drive mode (port 1)
- s40 18 02 F8 00 00 00 00;
  port 2 configuration
- s40 2D 43 4D 53 02;
  enable EEPROM

Load PANEL_VALUE from EEPROM

- sA0 00 00 sA1 x

Show PANEL_VALUE on Display

- s40 08 LOW_NIBBLE
- s40 09 HIGH_NIBBLE
- LOW_NIBBLE and HIGH_NIBBLE are set in display digit code, which is determined using the table below.

Table 11. Display Digit Code

Digit	Port Data	Digit	Port Data
0	C0	9	88
1	FC	A	84
2	92	B	A1
3	98	C	C3
4	AC	D	B0
5	89	E	83
6	81	F	87
7	DC	DOT Mask	7F
8	80	SPACE	FF

LOW_NIBBLE = table [PANEL_VALUE&0x0F];
HIGH_NIBBLE = table [PANEL_VALUE>>4];

Read Port 2 Int_Status Reg

- s40 12 s41 x

Read Port 2 Input Reg

- s40 0A FF s40 02 s41 x

Write PANEL_VALUE in EEPROM

- sA0 00 00 PANEL_VALUE

Conclusion

Different aspects of the CY8C95xx Expander device were considered on the basis of various examples. The nuances of its operation and questions of its configuration were discussed. The Expander's distinguishing features (EEPROM and PWM) were touched upon. The system interrupt processes were taken into consideration. And, in the end, practical application was examined in the form of a front panel display.

About the Author

Name:	Andrew Smetana
Title:	Electronic Engineer
Education:	Andrew earned his Master of Science diploma in 2004 from National University "Lviv Polytechnic " (Ukraine). His interests include various aspects of embedded systems development.
Contact:	smetana@ukrwest.net

Appendix: LPT-to-I2C Adapter Schematic

This adapter can be used with the IIC Bus Universal program, which can be downloaded freely at http://www.boerde.de/~matthias/delphi/. Note that if you plan to use the Expander in 3.3V systems, you'll need to use gates that are able to tolerate 5V input at a 3.3V supply level, for example, a hex inverter such as MC74LCX06 from ON-Semiconductor or SN74LVC06A from Texas Instruments.

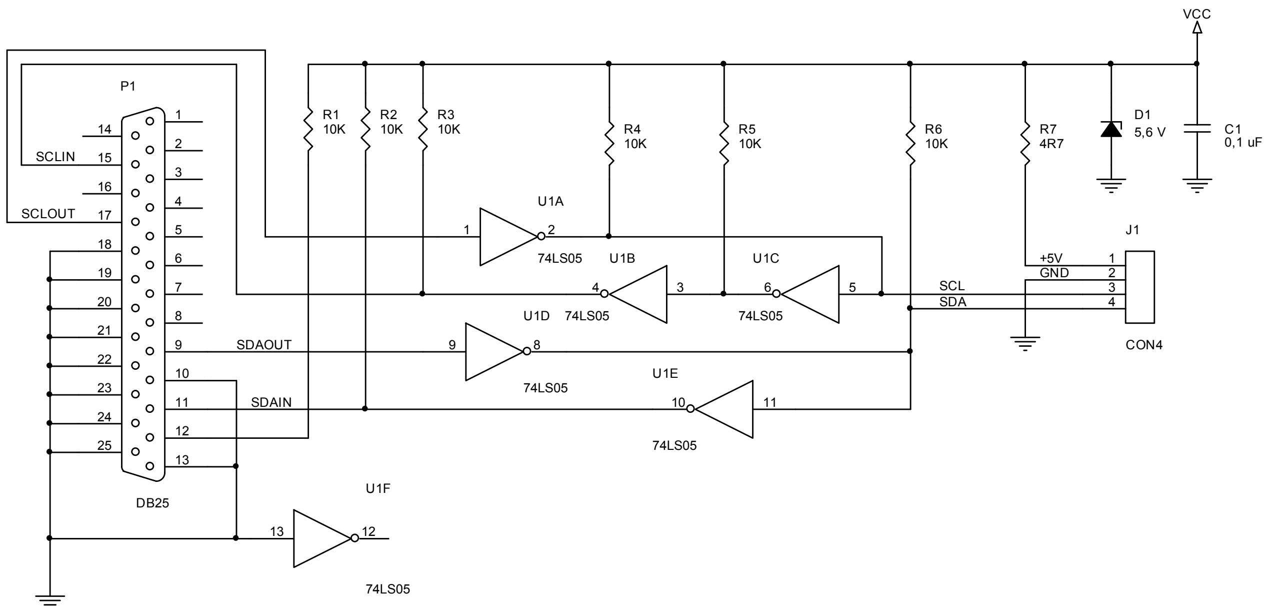

Figure 7. LPT-to-I^{2}C Adapter Schematic

Cypress Semiconductor
2700 162nd Street SW, Building D
Lynnwood, WA 98087
Phone: 800.669.0557
Fax: 425.787.4641

http://www.cypress.com/