The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

F²MC - 8FX Family, MB95200 Series, Keyboard Development using Matrix

There are three methods to design a keyboard function using the MB95200 series MCU: external interrupt, AD and matrix. This document describes how to use the matrix method to develop a keyboard function and illustrates the method with an example.

Contents

1 Introduction

There are three methods to design a keyboard function using the MB95200 series MCU: external interrupt, AD and matrix.

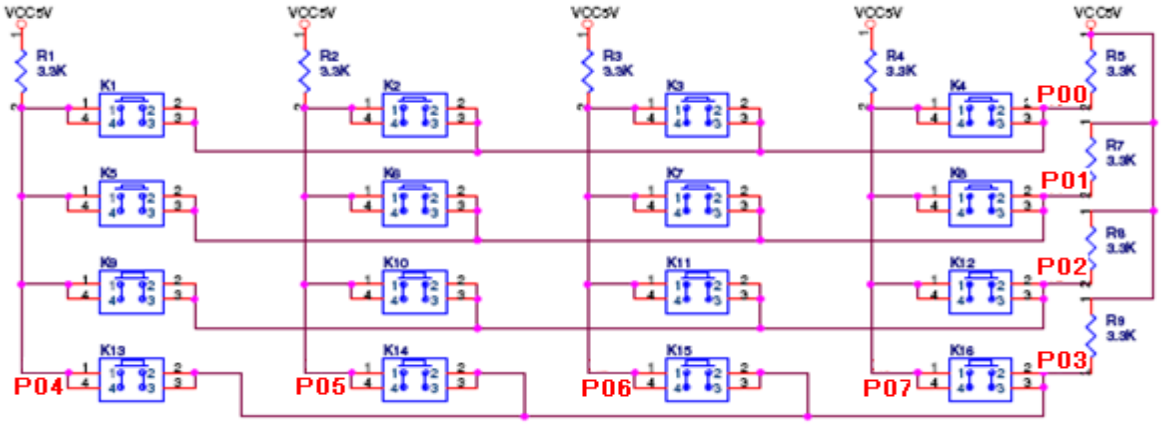
This document describes how to use the matrix method to develop a keyboard function and illustrates the method with an example.

2 Hardware Design

This chapter introduces how to use the matrix method to create a keyboard function with hardware circuit.

There are two methods to eliminate jittering, software method and hardware method. To use the hardware method, a capacitor should be added to the circuit.

Figure 1. Circuit for Keyboard Design Using Matrix



As indicated in the figure above, P04 ~ P07 are used as output pins and P00 ~ P03 as input pins to scan the keyboard status.

3 Keyboard Development

This chapter introduces how to develop the keyboard using matrix.

3.1 Function of Matrix Circuit

The Keyboard is an important tool for the input system. This reference design is used to show how to save I/O pins when adopting keyboard with many keystrokes.

A real-time monitoring method for polling the keyboard status is also used. The period of polling is a fixed value. It means any movement slower than this value will be caught by the monitoring software. It will ensure the high responding action of the keyboard.

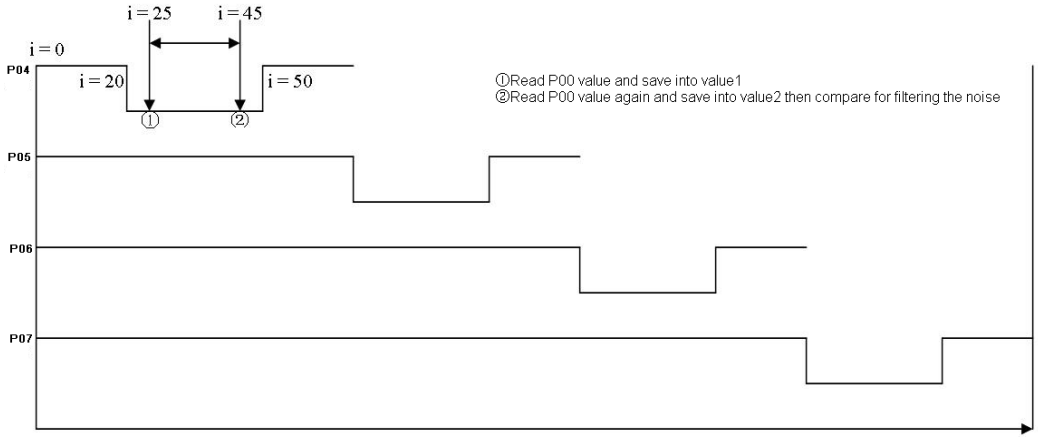
3.2 Jittering Elimination

Jittering elimination is a problem for keyboard design. There are two methods to resolve this problem. One is to add a capacitor to the hardware circuit. The other is to design a delay by using the 8/16-bit composite timer.

3.3 Jittering Elimination Example

The timing chart below is an example of jittering elimination. This example is a 4×4 keyboard designed by matrix. X0, X1, X2 and X3 takes turns to scan the keyboard for about 50 ms each. So it takes about 200 ms to scan the whole keyboard.

Figure 2. Timing Chart for Jittering Elimination



4 Resource Usage

This chapter introduces evaluation steps of normal run status.

To monitor the keyboard status, I/O pins are necessary. This chapter will introduce the usage of I/O port. Please refer to Chapter 9 of the MB95200 Series Hardware Manual for detailed register setting.

4.1 Register Introduction

4.1.1 Port Data Register (PDR)

This register contains bit information of corresponding input or output pins. The values are output, if the Port Direction Register is set to output mode.

Please note that the resource output control bit overwrites the PDR bit value.

PDRx_yz	Pin Function
0	Pin state low (VSS)
1	Pin state high (VDD)

4.1.2 Data Direction Register (DDR)

This register contains the bit information of the corresponding pins if they act as input or output.

DDRx_yz	Peripheral Function Output	Pin Function
0	Disable	Port Input
1	Disable	Port Output
invalid	Enable	Peripheral Function Output

4.1.3 Pull-up Control Register (PUL)

This register connects an internal pull-up resistor to a port pin.

PULx_yz	Pull-up Resistor
0	Disable
1	Enable

Please see datasheet for the resistor value.

4.1.4 Input Level Selection Register (ILSR)

With this register one of the following input levels can be chosen.

ILSR	Input Level	VIL	VIH
0x04	CMOS	0.3 VCC	0.7 VCC
0x00	Hysteresis	0.3 VCC	0.7 VCC

Please note that this function is available only in PDR0_P04.

4.2 General Introduction

I/O pins which are necessary in this system can be classified into two groups: output pins and input pins. The output pins are used to output high voltage while the input pins are used to scan the keyboard status.

4.2.1 How to Set Input Pin

If a pin functions as an input port, the corresponding bit in the Data Direction Register should be set to “0”.

To set an externally connected source to high-Z state, please use an external pull-up or pull-down resistor or set the corresponding bit in the Pull-up Register

There are three types of input modes: digital input, ADC input and peripheral function input:

- Digital input means the port is used as general I/O.
- ADC input means the port is used for analog input only.
- Peripheral function input means the port is used by a peripheral function as input, such as external interrupt input.

4.2.2 How to Set Output Pin

There are two kinds of output modes: digital output and peripheral function output.

- Digital output means the port is used as general I/O.
- Peripheral function output means the port is used for peripheral resource output such as output of 8/16-bit compound timer.

4.2.3 Pull-up Register

The P0 and PG ports, when in input-mode, can enable an internal pull-up resistor (about 50

K Ω, please see datasheet for the exact value) by programming the pull-up register (2.2.3).

The initial value of “0” disconnects the internal pull-up resistor. Writing “1” to the corresponding bit in the PULx register enables the resistor.

If the port is used for output, the value of the register-bit has no meaning and the pull-up resistor is disabled (Exception: For I2C pins SDA and SCL, the setting remains. Also for UART output SOT the internal pull-up can be used if not provided by line driver).

The pull-up resistor is disabled when the microcontroller is in stop or timer mode.

The resistor is also disabled if the pin is used for ADC input.

If an external pin is used by the external bus interface, the internal pull-up register cannot be used.

5 Software Design

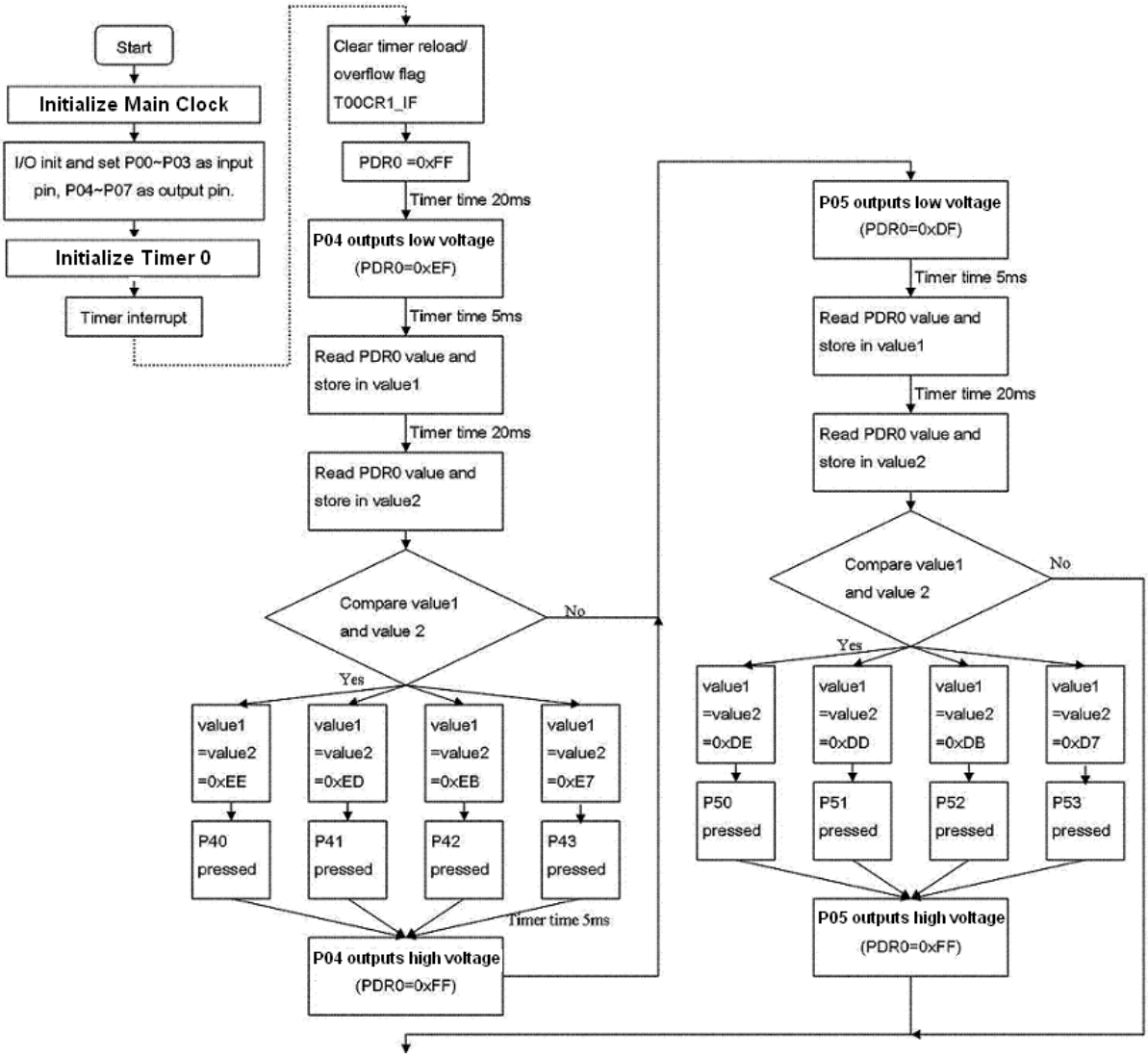
This chapter describes how to develop a keyboard by matrix.

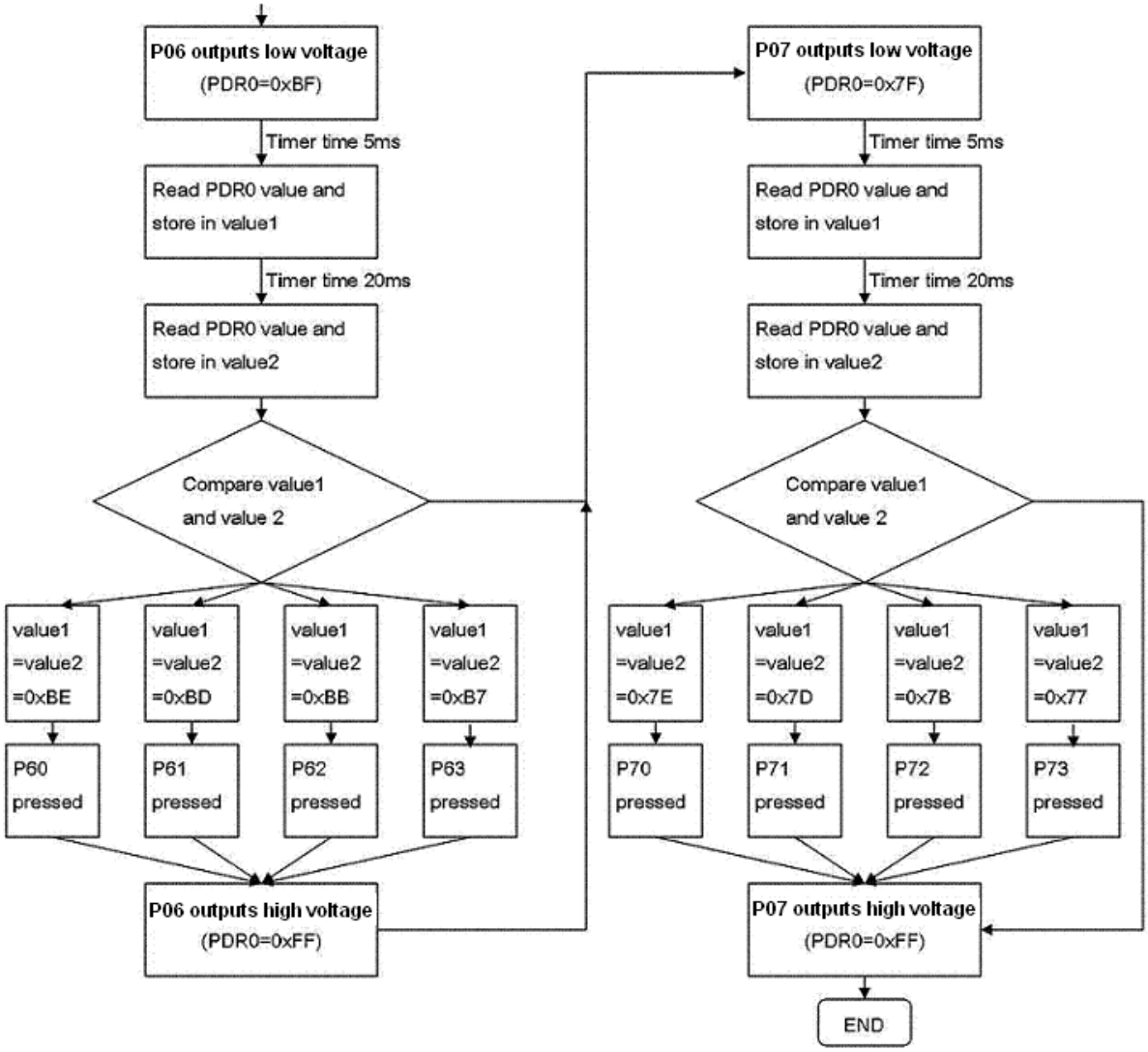
5.1 Software Design

To realize this function, first initialize the I/O register, and setup the AD input forbid, then enable the timer interrupts to eliminate jittering. The output I/O pin outputs high voltage and MCU waits for the input signal. Refer for to the hardware circuit, if some keys pressed, the MCU will scan this information and transact the corresponding key function.

The flow chart is illustrated as below:

Figure 3. Flow Chart of Keyboard Development by Matrix




P06 outputs low voltage
(PDR0=0xBF)
Timer time 5ms
Read PDR0 value and store in value1
Timer time 20ms
Read PDR0 value and store in value2
Compare value1 and value 2
value1 =value2 =0xBE
value1 =value2 =0xBD
value1 =value2 =0xBB
value1 =value2 =0xB7
P60 pressed
P61 pressed
P62 pressed
P63 pressed
P06 outputs high voltage
(PDR0=0xFF)
P07 outputs low voltage
(PDR0=0x7F)
Timer time 5ms
Read PDR0 value and store in value1
Timer time 20ms
Read PDR0 value and store in value2
Compare value1 and value 2
value1 =value2 =0x7E
value1 =value2 =0x7D
value1 =value2 =0x7B
value1 =value2 =0x77
P70 pressed
P71 pressed
P72 pressed
P73 pressed
P07 outputs high voltage
(PDR0=0xFF)
END

6 Sample Code

This chapter illustrates keyboard development using Matrix.

Based on MB2146-410-01, the following code is intended to illustrate how to create a keyboard function with I/O port. Port 0 is used for detecting key status. The default level of input is high.

```
/*                              SAMPLE CODE                                */
/*-------------------------------------------------------------------------*/
/* Give a example for basic I/O matrix */
#include "../MB95200_IO/mb95200.h"
void timer_init(void)
{
       T01DR = 0x13;       // 5000us
       T00DR = 0x88;
       TMCR0 = 0x10;       // 16-bit
       T00CR0 = 0x81;      // interval timer with continuous mode
       T00CR1 = 0xA0;      // disable output, start timer
}
/*P04 SCAN*/
void IO_restart(void)      // PDR0 restart
{
       PDR0 = 0xFF;
}
void P04_low(void)         // Pin P04 begin to scan
       {
              PDR0_P04 = 0;
       }
void Port_Value1(void)     // Read and store PDR0 in value 1
       {
              port_value1 = PDR0;
       }
void Row_one_process(void) // key judge
{
       port_value2 = PDR0; // Read and store PDR0 in value 2
       if(port_value2 == port_value1)
       {
              switch(port_value2)
              {
                     case 0xEE: SW1();
                                         break;
                     case 0xED: SW2();
                                         break;
                     case 0xEB: SW3();
                                         break;
                     case 0xE7: SW4();
                                         break;
              }
       }
}
/* P05 scan*/
void P05_low(void)         // Pin P05 begin to scan
{
       PDR0_P05=0;
}
```

```
void Row_two_process(void)       // key judge
{      port_value2 = PDR0;
       if(port_value2 == port_value1)
       {
              switch(port_value2)
              {
                     case 0xDE: SW5();
                                          break;
                     case 0xDD: SW6();
                                          break;
                     case 0xDB: SW7();
                                          break;
                     case 0xD7: SW8();
                                          break;
              }
       }
}
/* P06 scan*/
void P06_low(void)               // Pin P06 begin to scan
{
       PDR0_P06=0;
}
void Row_three_process(void)     // key judge
{
       port_value2 = PDR0;
       if(port_value2 == port_value1)
       {
              switch(port_value2)
              {
                     case 0xBE: SW9();
                                          break;
                     case 0xBD: SW10();
                                          break;
                     case 0xBB: SW11();
                                          break;
                     case 0xB7: SW12();
                                          break;
              }
       }
}
/* P07 scan*/
void P07_low(void)               // Pin P07 begin to scan
{
       PDR0_P07=0;
}
void Row_four_process(void)      // key judge
{
       port_value2 = PDR0;
       if(port_value2 ==port_value1)
       { switch(port_value2)
              {
                     case 0x7E: SW13();
                                          break;
                     case 0x7D: SW14();
                                          break;
                     case 0x7B: SW15();
                                          break;
                     case 0x77: SW16();
                                          break;
              }
       }
```

```
void SW1(void)              // Key Process
{
. . .
}
. . .
. . .
void SW16(void)
{
. . .
}
. . .
__interrupt void Timer_Interrupt (void) //key scan process
{
       T00CR1_IF=0;
       timer_counter++;
       if(timer_counter==0)       // PDR0 restart
       {
              IO_restart();
       }
       if(timer_counter==4)       // P04 begin to scan
       {
              P04_low();
       }
       if(timer_counter==5)       // Read and store PDR0
       {
              Port_Value1();
       }
       if(timer_counter ==9)      // Read and store PDR0 and judge keypressed
       {
              Row_one_process();
       }
       if(timer_counter==10)      // PDR0 restart
       {
              IO_restart();
       }
       if(timer_counter==14)      // P05 begin to scan
       {
              P05_low();
       }
       if(timer_counter==15)      // Read and store PDR0
       {
              Port_Value1();
       }
       if(timer_counter==19)      // Read and store PDR0 and judge keypressed
       {
              Row_two_process();
       }
       if(timer_counter==20)      // PDR0 restart
       {
              IO_restart();
       }
       if(timer_counter==24)      // P06 begin to scan
       {
              P06_low();
       }
       if(timer_counter==25)      // Read and store PDR0
       {
              Port_Value1();
       }
```

```
        if(timer_counter==29) // Read and store PDR0 and judge keypressed
        {
                Row_three_process();
        }
        if(timer_counter==30) // PDR0 restart
        {
                IO_restart();
        }
        if(timer_counter==34) // P07 begin to scan
        {
                P07_low();
        }
        if(timer_counter==35) // Read and store PDR0
        {
                Port_Value1();
        }
        if(timer_counter==39) // Read and store PDR0 and judge keypressed
        {
                Row_four_process();
        }
        if(timer_counter==40) // PDR0 restart
        {
                IO_restart();
        }
        if(timer_counter>=41) // Timer_counter restart
        {
                timer_counter=0;
        }
}
void clock_Select(void)
{
        SYCC = 0x00;
        WATR = 0x00;
        STBC = 0x01;
        SYCC2= 0xF4;
}
void main(void)
{
        InitIrqLevels();
        __EI();
        timer_counter =0;
        DDR0 = 0xF0;          //P00~P03= in P04~P07=out
        AIDRL = 0xFF;         //Port input enable
        clock_Select();       //Main clock select
        timer_init();         //Timer initialize
```

7 Performance Evaluation

To eliminate jittering, there are hardware method and software method. Though the code of hardware method is very simple, only using the timer calcula graph, its effect is not good, because the system may not be able to scan the edge. By comparison the software method is better.

8 Additional Information

For more information about how to use MB9595200H/210H EV-board, BGM Adaptor and

SOFTUNE, please refer to SKT MB2146-410A-01-E User Manual, or visit websites:

http://www.cypress.com/documentation/software-and-drivers/f2mc-8fx-mb95200h210h-series-starter-kit-mb2146-410a-01-e-setup

Document History

Document Title: AN205510 – F²MC-8FX Family, MB95200 Series, Keyboard Development using Matrix

Document Number: 002-05510

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	RLIU	03/05/2009	Original version
*A	5245179	RLIU	04/27/2016	Migrated Spansion Application Note “MCU-AN-500038-E-10” to Cypress format.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com