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.

AN205375

FR, MB91460, Clock Calibration

This application note describes the functionality of the Clock Calibration function and gives an example.

Contents

1 Introduction....1
1.1 Key Features....1
2 Clock Calibration....1
2.1 Functionality....1
2.2 Block Diagram....2
2.3 Timing Diagram....2
2.4 Registers....3
3 Calibration and Measurement....4
3.1 Interpreting Measurement Results....4
3.2 Calibration of the Real Time Clock....4
4 Clock Calibration Examples....5
4.1 Calibration Measurement of the RC or Sub Clock....5
4.2 Calibrating the Real Time Clock....7
4.3 Calibrating the USART....11
5 Additional Information....13
Document History....14

1 Introduction

This application note describes the functionality of the Clock Calibration function and gives an example.

1.1 Key Features

Measurement of the following clocks against the Main Clock (`CLKMC`)

- RC Clock (CLKRC) (100 kHz)
- Sub Clock (CLKSC)

2 Clock Calibration

The Basic Functionality Of The Clock Calibration Function

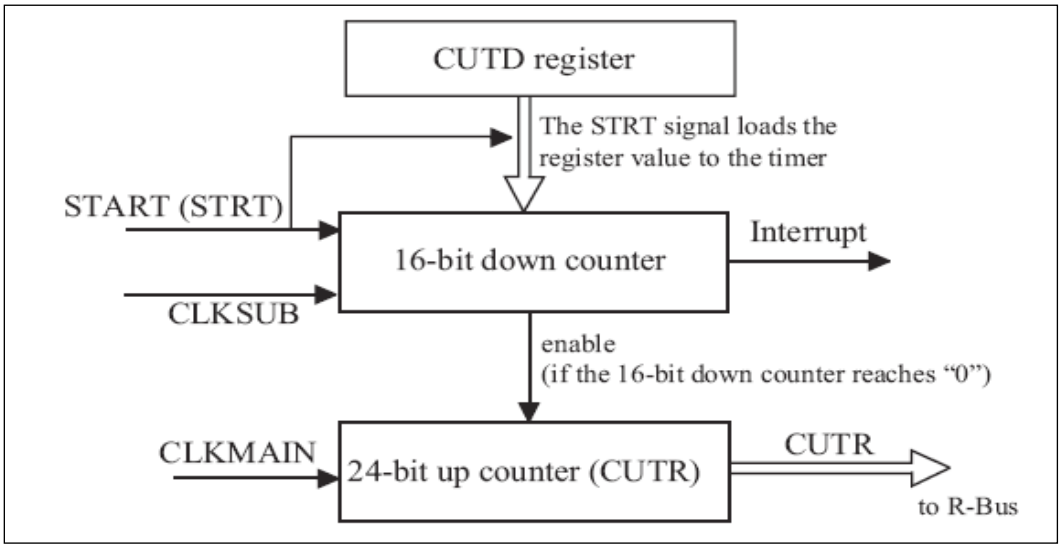
2.1 Functionality

With the Clock Calibration the RC or Sub Clock oscillation clocks a counter for a given duration value. In this duration time the calibration timer counter is clocked by the Main Clock (CLKMAIN).

This allows trimming other resources, which have reload-counter pre scalar (Real Time Clock, USART, etc.).

2.2 Block Diagram

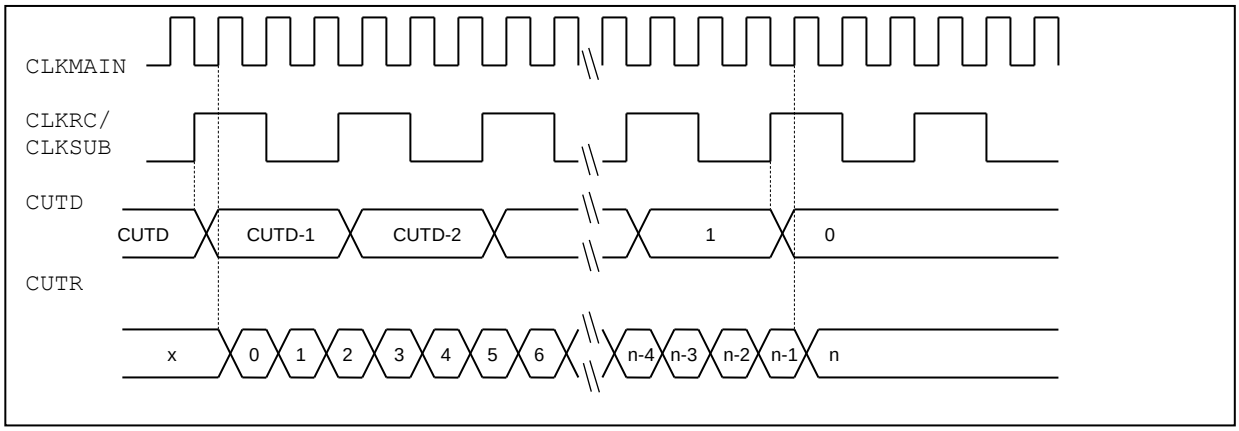
Figure 1. Clock Calibration Block Diagram



2.3 Timing Diagram

The following timing diagram shows the functionality of the Clock Calibration unit.

Figure 2. Timing diagram Clock Calibration Unit



Please note, that the length of the clock pulses is not drawn in a proportional manner.

2.4 Registers

2.4.1 Clock Calibration Unit Control Register (CUCR)

The CUCR consists of the following bits:

Bit No.	Bit Name	Initial Value	Value	Description
7,6, 5	-	0	-	*Unused bits*
4	STRT	0	0	No Calibration
			1	Start Calibration
2-3	-	0	-	*Unused bit*
1	INT	0	0	No calibration / Calibration ongoing Write: Clear bit
			1	Calibration finished
0	INTEN	0	0	Interrupt disabled
			1	Interrupt enabled

2.4.2 Clock Calibration Duration Timer Data Register (CUTD)

This 16-bit register contains the duration interval for the duration timer. This timer is clocked by the RC or Sub Clock as selected by CSCFG: CSC2

2.4.3 Clock Calibration Timer Data Register (CUTR)

This 24-bit register is clocked by the Main Clock (CLKMC). Its counting is gated by the duration of the clock to be calibrated.

Note: Calibration Unit is always supplied with 100 kHz independent of CSCFG: RCSEL bit setting

3 Calibration and Measurement

This chapter shows how to calculate Calibration Data

3.1 Interpreting Measurement Results

Assume the 32.768 kHz sub clock should be measured with an ideal Main Clock of 4MHz. The duration in CUTD was set to a value of 0x2000 = 8192, which should represent 0.25s at an ideal sub clock oscillation.

After measurement, the value of CUTR is e. g. 0x0F4002. What does this mean?

The Calibration Timer has counted to 0x0F4002 = 999426 in the duration time given by the sub clock. This represents (at ideal 4 MHz Main Clock reference) a duration time of 999426 / 4000000 = 0.24986s. Thus, the sub clock oscillation is a bit too fast.

3.2 Calibration of the Real Time Clock

Assume the Real Time Clock is clocked by the Sub Clock oscillator with the same frequency deviation given in 3.1.

At an ideal Sub Clock of 32.768 kHz the Sub Second Register (WTBR) would have the reload value 8192 (= 0x2000). Because the oscillation was measured as too fast, the new value can be calculated as:

$$WTBR_{CALIBRATED} = \frac{f_{RTC} \cdot f_{MAIN} \cdot t_{INTERVAL}}{4s^{-1} \cdot CUTR}$$

f_{RTC}: Clock source of Real Time Clock
f_{MAIN}: Main Clock (CLKMC)
$t_{INTERVAL}$: Interval time given by CUTD

With the example values above the calculation is:

$$WTBR_{CALIBRATED} = \frac{32768s^{-1} \cdot 4000000s^{-1} \cdot 0.25s}{4s^{-1} \cdot 999426} = 8196.70$$

The new calibrated Sub Second Register value is then: 8197

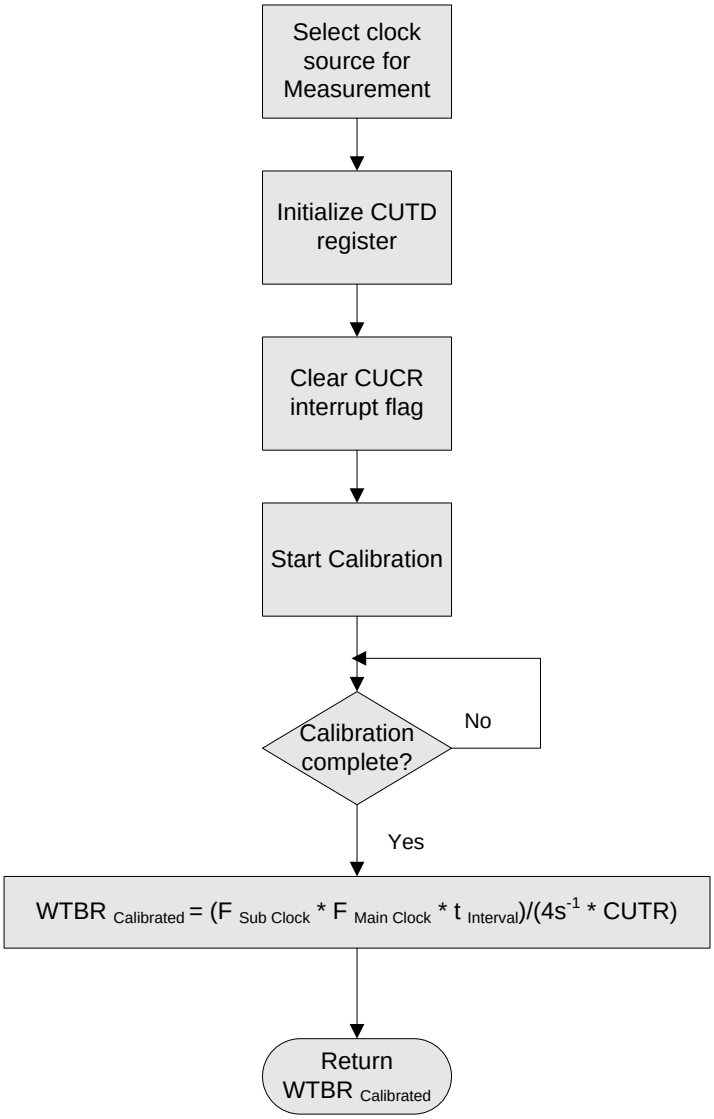
4 Clock Calibration Examples

Examples for the Clock Calibration usage.

4.1 Calibration Measurement of the RC or Sub Clock

The following example shows, how to measure the RC or sub clock. No interrupts are used.

Figure 3. Flowchart for Calibration Measurement



```
#include "mb91467d.h"

#define MAIN_CLOCK 4000000              // Main Clock in Hz
#define SUB_CLOCK 32768                 // Sub Clock in Hz
#define RC_CLOCK 100000                 // RC clock in Hz
#define INTERVAL 0.5                    // Measurement duration in s
#define CUTD_VALUE (INTERVAL * SUB_CLOCK)
#define CUTR  ((CUTR1<<16)|(CUTR2))

unsigned long ClockMeasure(unsigned int duration)
{
    float calibrate;
    unsigned long wtbr_cal2;
        CUCR = 0x00;        // measure sub clock, no interrupts
    //CUCR = 0x04;      // measure RC clock, no interrupts

    CUTD = CUTD_VALUE;
    CUCR_INT = 0;
    CUCR_STRT = 1;      // start calibration measurement
    while (!CUCR_INT); // wait for end of measurement

    CUCR_INT = 0;       // clear INT flag (is set even if no interrupts used)

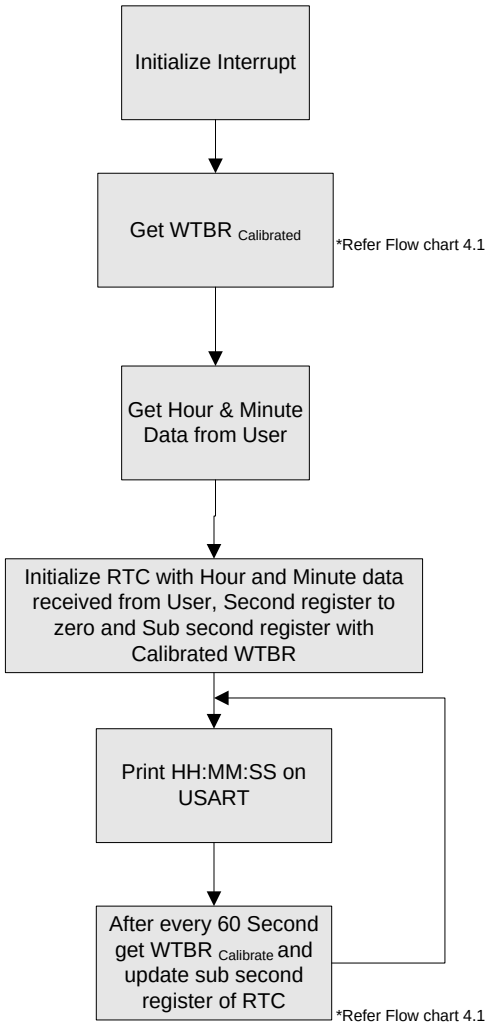
    calibrate = 0.5 + (((float) SUB_CLOCK * (float) MAIN_CLOCK
                    * (float) INTERVAL) / ((float) 4 * (float) CUTR));
    wtbr_cal2 = (unsigned long) calibrate;   //0.5 is added for rounding

    return wtbr_cal2;  // return measurement value
}
```

4.2 Calibrating the Real Time Clock

The following example shows how to calibrate the Real Time Clock clocked by the 32 kHz Sub Clock.

Figure 4. Flowchart for RTC Calibration



```
#include "mb91467d.h"
#define MAIN_CLOCK 4000000        // Main Clock in Hz
#define SUB_CLOCK 32768           // Sub clock in Hz
#define INTERVAL 0.5              // Measurement duration in s
#define CUTD_VALUE (INTERVAL * SUB_CLOCK)
#define CUTR_IDEAL (unsigned long) (INTERVAL * MAIN_CLOCK)
#define CUTR  ((CUTR1<<16)|(CUTR2))
#define UPDATE_INTERVAL 15 // 4s * 15 = 60s
#define WEIGHT (float)100
#define T_INTERVAL 60
char update_count;
unsigned long cal;
double dt, w;
void InitRTC(unsigned char hour, unsigned char minute, unsigned long subsecond)
{
    /* Real Time Clock Source */
    CSCFG_CSC &= 0xC;
    CSCFG_CSC |= 0x1;     // RTC Source = Sub Clock

    WTCR_ST = 1;
    WTCR_ST = 0;
   /* Disable interrupt */
    WTCR_INTE3 = 0;
    WTCR_INTE2 = 0;
    WTCR_INTE1 = 0;
    WTCR_INTE0 = 0;
    WTBR = subsecond;
    WTHR = hour;
    WTMR = minute;
   WTSR = 0;
    WTCR_UPDT = 1;
       while(!SSR05_RDRF);                    // wait for "RETURN"
       WTCR_ST = 1;                           // Start RTC
       (volatile) RDR05;                      // Flush RDR0
    return;
 }
unsigned long ClockMeasure(unsigned int duration)
{
   float calibrate;
   unsigned long wtbr_cal2;

   CUTD = duration;
   CUCR_INT = 0;
   CUCR_STRT = 1;
   while (!CUCR_INT);

   CUCR_INT = 0;
```

```
calibrate = 0.5 + (((float) SUB_CLOCK * (float) MAIN_CLOCK
                    * (float) INTERVAL) / ((float) 4 * (float) CUTR));
    wtbr_cal2 = (unsigned long) calibrate;

    return wtbr_cal2;
}
void StartRLT0(void)
{
    TMCSR0 = TMCSR0 | 0x0001;
}
void InitRLT0(void)
{
    TMCSR0 = 0x181A;
    TMRLR0 = 0x7A12;
}
void wait(long a)
{
    long i;

    for (i = 0; i < a; i++)
        __wait_nop();
}
void printdigits(unsigned char val)
{
    unsigned int val2;
    val2 = val / 10;
    Putch5(val2 + 0x30);
    val = val - (val2 * 10);
    Putch5(val + 0x30);
}
void printtime(void)
{
    printdigits(WTHR);
    Putch5(':');
    printdigits(WTMR);
    Putch5(':');
    printdigits(WTSR);
    Puts5("\n");
}
void main(void)
{
    unsigned char hh, hl, mh, ml, hour, minute;

    InitIrqLevels();
    set_il(7);                // allow all levels
    EI();                     // globally enable interrupts
        PORTEN = 0x3;
        InitUart5();                // UART
        Puts5 (welcome);            // Welcome User
        InitRLT0();
        StartRLT0()
    w = WEIGHT;
    dt = 0;
        cal = ClockMeasure(CUTD_VALUE);
```

```
Puts5("\nRTC/RC clock self calibration\n");
    Puts5("============================\n");
    Puts5("CUTR: 0x");
        printlong(CUTR);
    Puts5("\nCUTR: ");
    printdeclong(CUTR);
        Puts5("\nSub Second: 0x");
    printlong(cal);
    Puts5("\nSub Second: ");
    printdeclong(cal);
    Puts5("\n\nEnter Time (HHMM) <RETURN>: ");
         (volatile) RDR5;                        // Flush RDR0
    hh = Getch0();
    Putch5(hh);
    hl = Getch5();
    Putch5(hl);
    mh = Getch5();
    Putch5(mh);
    ml = Getch5();
    Putch5(ml);
         hour = (hl & 0x0F) + 10 * (hh & 0x0F);
    minute = (ml & 0x0F) + 10 * (mh & 0x0F);
    hour = (hl & 0x0F) + 10 * (hh & 0x0F);
    minute = (ml & 0x0F) + 10 * (mh & 0x0F);

    InitRTC(hour, minute, cal);

    InitSCT();
    Puts5("\n");

    while(1)
    {
    HWWD_CL = 0;
        printtime();
    }
}
// Recalibration of RTC
__interrupt void RLT0interrupt(void)
{
    if (update_count >= UPDATE_INTERVAL)
    {
         update_count = 0;

         dt += (((double)CUTR / (MAIN_CLOCK / 2)) * SUB_CLOCK / cal
              / 4 * (double)T_INTERVAL) - (double)T_INTERVAL;

         cal = ClockMeasure(CUTD_VALUE);
         cal += dt * WEIGHT;
           WTCR_INT0 = 0;

         while(!WTCR_INT0);       // wait for second
           WTBR0 = (0xFFFF & cal);// Set calibrated prescaler value
         WTBR1 = (cal >> 16);
    }
         update_count++;
    TMCSR0_UF = 0;
       }
```

4.3 Calibrating the USART

As per RS-232 standard, the maximum variance allowed for baud rate is 5%.

As per data sheet, for chip to chip, value of internal RC oscillator may vary from 1 MHz to 4 MHz for CLKRC of 2 MHz or it may vary from 50 kHz to 200 kHz for CLKRC of 100 kHz.

If we assume that CLKRC is operating at typical frequency of 100kHz and CLKRC is selected as clock source for CLKS1 and intern to CLKP1 than BGRn register value is calculated as…

$$BGRn = \left[\frac{CLKRC}{9600} - 1\right]$$

$$\therefore \quad BGRn = \left[\frac{100000}{9600} - 1\right]$$

$$\therefore \quad BGRn_{Typ} = 9$$

If we set the BGRn equal to 9 and if the actual frequency of CLKRC is 150 kHz than resulting baud rate will be

$$BaudRate = \left[\frac{CLKRC}{BGRn+1}\right]$$

$$\therefore \quad BaudRate = \left[\frac{150000}{9+1}\right]$$

$$\therefore BaudRate = 15000$$

This is far beyond the acceptable variance by RS-232 standard.

Considering above facts, when CLKRC is selected as clock source for CLKS1 and intern to CLKP1, one can use clock calibration unit to find the actual frequency of CLKRC and can set BGRn register value to match the required baud rate.

Let us consider an example; here we will assume that clock source CLKMC which is input to calibration timer is stable and its frequency is exactly 4 MHz.

Now we know that when we initialise CUTD to 0x0000 and start measurement, and when an underflow occurs, the measurement will take (0xFFFF + 1) * T_{CLKRC} seconds.

Let us assume that for the above measurement CUTR counts up to 0x145855.

From this information, we can find out value of CLKRC as follow

$$\frac{CUTD}{CLKRC} = \frac{CUTR}{CLKMC}$$

$$\therefore \quad \frac{50000}{CLKRC} = \frac{1333333}{4\,MHz}$$

$$\therefore \quad CLKRC = \frac{50000 * 4\,MHz}{1333333}$$

$$\therefore CLKRC = 150.0000\,kHz$$

One point of interest over here is, consider that when CUTD reaches to zero and when STRT is reset indicating end of calibration process, it may happen that Main Clock cycle is already finished 90% of its time and was about to increment CUTR, so CUTR instead of counting 1333334 has counted 1333333 cycles. (CUTR count is incremented at the end of Main clock cycle).

If we recalculate, using above formula we will get CLKRC as follow

$$\frac{CUTD}{CLKRC} = \frac{CUTR}{CLKMC}$$
$$\therefore \frac{50000}{CLKRC} = \frac{1333334}{4\,MHz}$$
$$\therefore CLKRC = \frac{50000 * 4\,MHz}{1333334}$$
$$\therefore CLKRC = 149.9999\,kHz$$

Now from above calculation, as we know the RC clock frequency, let us find out the BGRn register value for baud rate of 9600

$$BGRn = \left[\frac{CLKRC}{9600} - 1\right]$$
$$\therefore \quad BGRn = \left[\frac{150000}{9600} - 1\right] \text{ or } BGRn = \left[\frac{149999}{9600} - 1\right]$$
$$\therefore BGRn = 15 \text{ or } BGRn = 15$$

Let us also do reverse calculation to find out what is exact baud rate if we select BGRn value to 15.

$$BaudRate = \left[\frac{CLKRC}{BGRn + 1}\right]$$
$$\therefore BaudRate = \left[\frac{150000}{15+1}\right] \text{ or } BaudRate = \left[\frac{149999}{15+1}\right]$$
$$\therefore BaudRate = 9375 \text{ or } BaudRate = 9375$$

Here actual baud rate is 2.34% more than the required one, which is with in acceptable limits of RS232 Standards.

Also from the above calculation, the error due to one cycle count miss of CUTR will not affect the final baud rate, so one can safely ignore it.

Considering CLKRC may vary from 50kHz to 200kHz and the fact that for asynchronous serial communication BGRn must be set equal to or more than four; Maximum and minimum baud rate that can be set is calculated as follow

$$BaudRate = \left[\frac{CLKRC}{BGRn+1}\right]$$

$$\therefore BaudRate = \left[\frac{50000}{3+1}\right] \text{ or } BaudRate = \left[\frac{200000}{3+1}\right]$$

$$\therefore BaudRate = 12500\text{bps or } BaudRate = 50000\text{bps}$$

So, for all practical purpose where CLKRC is not known in advance one can still start with baud rate of 12500bps.

One more fact to be considered while calibrating USART is that CLKRC frequency is a temperature dependent

5 Additional Information

Information about CYPRESS Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software examples related to this application note is:

91460_Clock_Cal

It can be found on the following Internet page:

http://www.cypress.com/cypress-mcu-product-softwareexamples

Document History

Document Title: AN205375 - FR, MB91460, Clock Calibration

Document Number: 002-05375

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	02/26/2008	V1.0, First draft, HPi
			04/24/2008	V1.1, typo in title page corrected, MSt
*A	5066119	NOFL	02/10/2016	Converted Spansion Application Note “MCU-AN-300083-E-V11” to Cypress format
*B	5873274	AESATMP9	09/05/2017	Updated logo and copyright.
*C	6070279	NOFL	02/13/2018	Sunset Review Migrated to new template Updated links

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
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2008-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.