AN80248

PSoC® 3 / PSoC 5 Improving the Accuracy of Internal Oscillators

Author: Antonio Rohit De Lima Fernandes
Associated Project: Yes
Associated Part Family: PSoC® 3, PSoC 5
Software Version: PSoC Creator™ 2.0 or higher
Related Application Notes: AN54439, AN60631

If you need help with this application note, please contact the author at anto@cypress.com

AN80248 shows how to improve the accuracy of the PSoC® 3 or PSoC 5 internal low-speed oscillator (ILO) and internal main oscillator (IMO) through run-time trim. Two components developed for this purpose greatly simplify the process of calibrating the ILO and IMO with respect to a reference time base. This application note assumes that you are familiar with the PSoC 3 or PSoC 5 architecture and the PSoC Creator design environment.

Contents

Introduction 1
ILO Basics 1
  Applications of a Trimmed ILO 2
IMO Basics 2
  Applications of a Trimmed IMO 2
Trimming Theory 3
Component Usage 3
  kHz_ILO_Trim 3
  IMO_Trim 4
  Test Project Description 5
Component Internal Implementation 7
  ILO Trim Design Considerations 7
  IMO Trim Design Considerations 8
Component Performance 9
Summary 9
Appendix A – IMO and ILO Frequency Variation Graphs 10
Appendix B – Identifying Nominal Clock Frequencies 11
Appendix C – main.c Code for the Test Project 12

Introduction

PSoC® 3 and PSoC 5 have a very powerful clocking system. This system offers the flexibility and performance to suit the needs of most embedded applications. It is comprised of clock sources and a clock distribution network. The clock sources available to you are: the internal main oscillator (IMO), external crystal oscillators (ECO) and internal low-speed oscillator (ILO). This application note describes the IMO and ILO as well a method to improve their accuracy through run-time calibration.

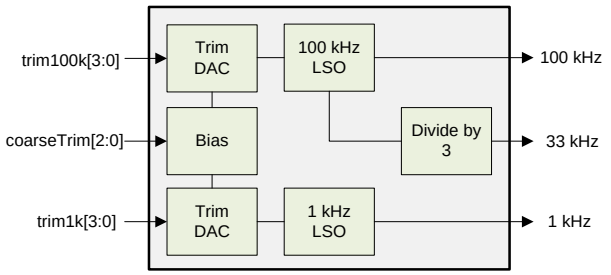
For more information on the clocking resources in PSoC 3 and PSoC 5, see AN60631 - PSoC 3 and PSoC 5 Clocking Resources. If you are new to PSoC 3 or PSoC 5, a good introduction is AN54181 - Getting Started with PSoC 3 or AN77759 - Getting Started with PSoC 5.

ILO Basics

The ILO is a low-speed, low-power and low-accuracy oscillator used as a kHz clock source, watchdog timer, and sleep timer. As shown in Figure 1, the ILO consists of two low speed oscillators (LSOs) and generates three frequencies – 1 kHz, 33 kHz, and 100 kHz. You can have any two of these frequencies output simultaneously.

After factory trim, the 1 kHz LSO has an accuracy range of -50%/+100%, while the 100 kHz LSO has a range of -55%/+100% over the entire operating range of voltage and temperature. Refer to Appendix A for graphs of ILO and IMO frequency variation versus temperature and voltage.

Figure 1. Simplified ILO Block Diagram



Run-time trimming can greatly improve the ILO's performance. This is done by adjusting the trim DACs and bias block shown in Figure 1 via dedicated trim registers. Figure 2 describes the trim registers in more detail.

Though you can use the methods presented here to trim both the 1 kHz and 100 kHz LSOs, this application note focuses on trimming the 1 kHz output. Henceforth, you can assume that all mentions of the ILO refer to the 1 kHz output of the ILO.

Figure 2. ILO Trim Registers and Their Description

ILO_TR0								
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SW Access: Reset	R/W: 1000				R/W:1000			
HW Access	R				R			
Name	tr_100k				tr_1k			

ILO_TR1								
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SW Access: Reset	NA: 00000					R/W:0	R/W:10	
HW Access	NA					R	R	
Name						ct_range	ctrim	

Bits	Name	Description	Maximum frequency	Minimum frequency
ILO_TR0[7:4]	tr_100k	Trim for 100 kHz output	4'b1111	4'b0000
ILO_TR0[3:0]	tr_1k	Trim for 1 kHz output	4'b1111	4'b0000
ILO_TR1[1:0]	ctrim	Coarse trim for ILO	2'b11	2'b00
ILO_TR1[2]	ct_range	Coarse trim range select	1'b0	1'b1

Applications of a Trimmed ILO

One of the major uses of the ILO is as a clock in sleep mode. A more accurate ILO allows for more precise event timing. This offers significant performance benefits, especially in PSoC 5 where the ILO is the only clock available in sleep mode.

When in sleep, PSoC 5 must wake up to refresh its internal power regulators at least every 256 ms. Hence, in practical applications, the PSoC 5 is configured for a sleep time of 128 ms so it does not oversleep at the lower bound of the ILO's accuracy range (-50%). A smaller sleep time results in more frequent wakeup events and thus higher average power consumption. This is undesirable in low-power applications. A more accurate ILO can thus significantly improve low-power performance in PSoC 5.

IMO Basics

The IMO is the main clock source of the PSoC. In PSoC 3, it provides clock outputs at 3, 6, 12, 24, 48 and 62.6 MHz while in PSoC 5 the IMO provides clock outputs at 3, 6, 12, 24 and 48 MHz. A factory trim value for each frequency range is stored in the device's flash memory. During the PSoC's boot-phase, the trim value corresponding to the speed setting of the IMO is loaded into the trim registers shown in Figure 3. The process of run-time trim modifies these registers, and does not affect the trim values stored in flash.

With factory trim, the IMO accuracy in CY8C38xx and CY8C36xx devices (across the range of operating temperature and voltage) varies from ±1% at 3 MHz to ±7% at 62.6 MHz. Please see the respective PSoC 3 family datasheets for more information.

For the PSoC 5 family, IMO accuracy varies between ±5% at 3 MHz and ±10% at 48 MHz. See Appendix A for graphs of IMO frequency variation versus temperature and voltage.

The IMO is generally used with the phase-locked loop (PLL) to generate a wide range of frequencies. Cypress recommends that you run the IMO at 3 MHz and use the PLL to generate the desired operating frequency. This is because the PLL introduces no additional error, and consumes less power than the IMO for frequencies greater than 24 MHz.

Similar to the ILO, the IMO contains an 11-bit trim DAC which is used to achieve a considerable accuracy improvement through run-time calibration.

Figure 3. IMO Trim Registers and Their Description

IMO_TR0								
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SW Access: Reset	R/W: 000			NA:00000				
HW Access	R			NA				
Name	imo_lsb							

IMO_TR1								
Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SW Access: Reset	R/W:00000000							
HW Access	R							
Name	imo_trim							

Bits	Name	Description	Maximum frequency	Minimum frequency
IMO_TR0[7:5]	imo_lsb	3 LSB of IMO trim	3'b111	3'b000
IMO_TR1[7:0]	imo_trim	8 MSBits of IMO trim	8'b11111111	8'b00000000

Applications of a Trimmed IMO

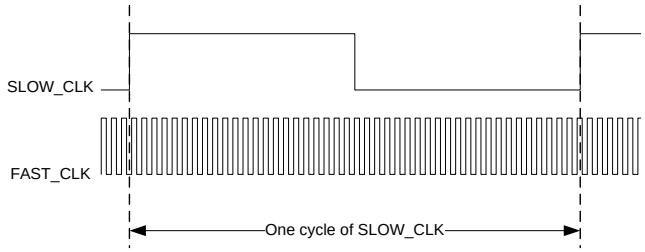
An accurate IMO has a wide range of applications, particularly for high-reliability communication. MHz crystals are generally more expensive than kHz crystals. Thus, calibrating the IMO with respect to a kHz crystal can reduce bill-of-material costs. For example, a high accuracy IMO eliminates the need for a MHz crystal for UART communication in PSoC 5.

Trimming Theory

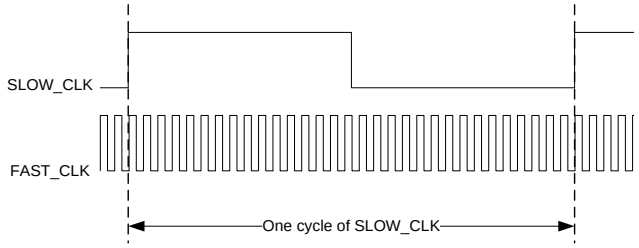
Trimming involves measuring the frequency of the clock of interest against an accurate time base. This reference time base may be slower or faster than your clock of interest.

Let us assume that the clock of interest is FAST_CLK and the reference is SLOW_CLK. If you count the number of FAST_CLK positive edges per cycle (or for a fixed number of cycles) of SLOW_CLK, there exist three cases:

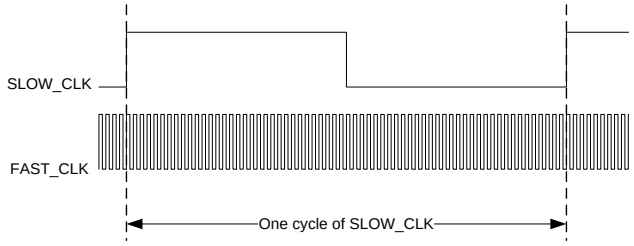
1. Edges Counted = Ideal number of edges
   FAST_CLK error = 0



2. Edges Counted < Ideal number of edges
   FAST_CLK error < 0

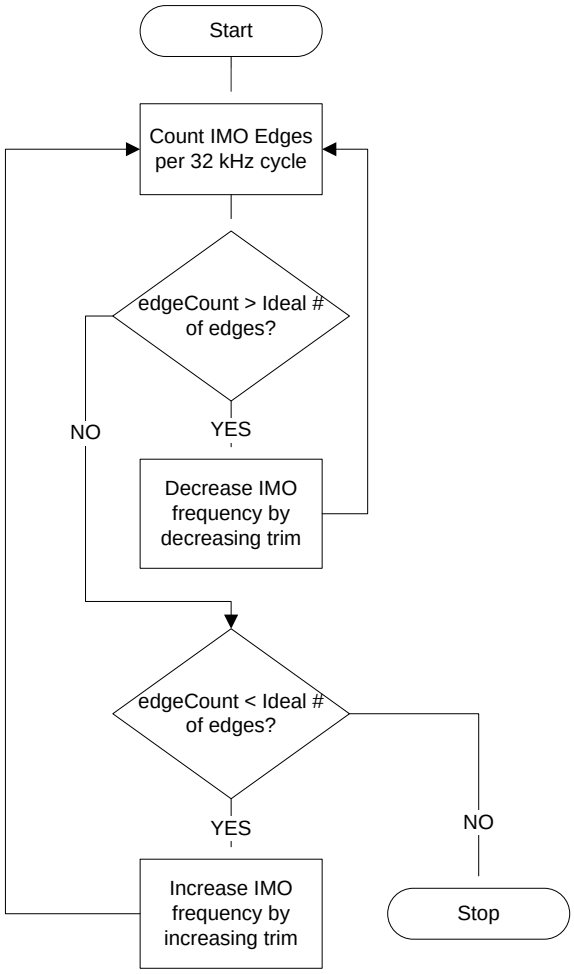


3. Edges Counted > Ideal number of edges
   FAST_CLK error > 0



This concept applies to trimming the IMO and the ILO. For example, say the SLOW_CLK is the 32 kHz crystal oscillator and the FAST_CLK is the IMO itself. The flowchart in Figure 4 shows the procedure to trim the IMO.

Figure 4. Theoretical Flowchart for IMO Trim



This method works for an accurate FAST_CLK and inaccurate SLOW_CLK as well. This is the case when you trim the ILO using a reference such as the IMO.

Component Usage

This section discusses the trimming components: kHz_ILO_Trim and IMO_Trim, their inputs, parameters, and APIs.

kHz_ILO_Trim

Symbol

As shown in Figure 5, the kHz_ILO_Trim component has only one input – the Reference clock from which the ILO derives the higher accuracy. The ILO frequency is supplied to the component internally.

Note The clock component in PSoC Creator can supply the ILO, 32 kHz XTAL or BUS_CLK frequencies without consuming digital clock dividers.

Figure 5. kHz_ILO_Trim Symbol



Parameters

You must define two parameters in the component customizer (Figure 6):

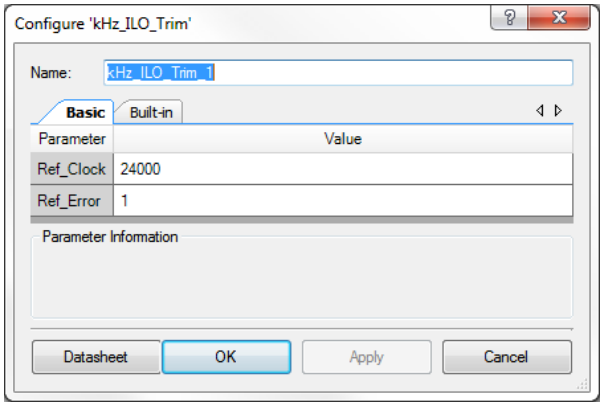
- Ref_Clock (kHz) – Frequency of the reference

  The upper bound for this frequency is limited to 32 MHz due to timing constraints, while the lower bound is determined by the acceptable level of error (see the section in this application note on Accuracy Attainable). It is recommended that this clock be the 3 MHz IMO directly or a clock derived from it.

- Ref_Error (%) – Error of the reference clock

  In the case of the 3 MHz IMO, the Ref_Error is 1 in PSoC 3. Appendix B shows where you can find the frequency and accuracy of system clocks.

Figure 6. kHz_ILO_Trim Component Customizer



APIs

The component comes with pre-defined APIs to simplify its use. For simplicity, assume that the component is named kHz_ILO_Trim.

- kHz_ILO_Trim_Start()
- kHz_ILO_Trim_Stop()
- kHz_ILO_Trim_BeginTrimming()
- kHz_ILO_Trim_CheckStatus()
- kHz_ILO_Trim_CheckError()
- kHz_ILO_Trim_RestoreTrim()
- kHz_ILO_Trim_Sleep()
- kHz_ILO_Trim_Wakeup()

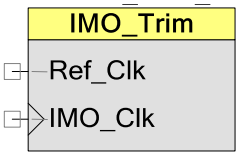
The `kHz_ILO_Trim_Start()` function initializes the component but does not start trim iterations. The `kHz_ILO_Trim_BeginTrimming()` function is called to initiate trimming. Once the calibration is complete, the correction iterations automatically stop. At any point, you can use the `kHz_ILO_Trim_CheckStatus()` function to check whether the ILO is within the desired accuracy range. The `kHz_ILO_Trim_CheckError()` function calculates the ILO error. If you want to revert to the original factory trim values, use the `kHz_ILO_Trim_RestoreTrim()` function. Sleep, Wakeup and Stop APIs are also available. Refer to the component datasheet for more information.

IMO_Trim

Symbol

As shown in Figure 7, the IMO_Trim component has two inputs: Ref_Clk and IMO_Clk. The 32 kHz crystal clock is typically used as Ref_Clk. IMO_Clk can be either the direct IMO output or a clock derived from the IMO via the PLL or clock dividers. The IMO_Clk terminal is external to the component so that you can reuse the associated clock.

Figure 7. IMO_Trim Symbol



Parameters

You must define three parameters in the component customizer (Figure 8):

- Averaging_Duration (cycles) – Number of Ref_Clk cycles for which IMO edges are counted.

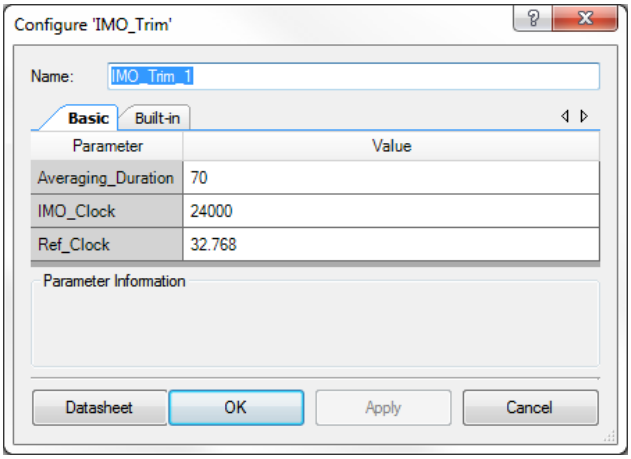
  Due to the component implementation, this parameter must be specified within the range of [2,127]. However, the actual limits depend on the values of IMO_Clock and Ref_Clock. A larger value for Averaging_Duration increases the clock stability but increases the time taken to settle to the final accuracy.

- IMO_Clock (kHz) – Frequency of IMO_Clk input.

  The lower limit for this input is the lowest IMO frequency – 3 MHz. The upper limit is 32 MHz due to timing considerations. Appendix B shows where you can find the exact frequency and accuracy of system clocks.

- Ref_Clock (kHz) – Frequency of the reference clock.

Figure 8. IMO_Trim Component Customizer



In a typical use-case, Ref_Clk is the 32.768 kHz crystal clock. Table 1 shows some valid combinations of IMO_Clk and Averaging_Duration values when Ref_Clk is the 32 kHz crystal. In the test project attached with this application note, we use an Averaging_Duration of 70 with a 24 MHz IMO_Clk.

Table 1. Table of Valid Averaging_Duration Values

Actual IMO (MHz)	IMO_Clk (MHz)	Averaging_Duration (Max cycles)
3 ± 1%	3	127
6 ± 2%	6	127
12 ± 3%	12	127
24 ± 4%	24	85
3 ± 1%	24*	88
3 ± 1%	32*	65
62.6 ± 7%	31.3**	64

* Via the PLL
** Via the clock divider

APIs

The APIs for IMO_Trim are exactly the same as those for kHz_ILO_Trim. For more information, see the IMO_Trim component datasheet.

- IMO_Trim_Start()
- IMO_Trim_Stop()
- IMO_Trim_BeginTrimming()
- IMO_Trim_CheckStatus()
- IMO_Trim_CheckError()
- IMO_Trim_RestoreTrim ()
- IMO_Trim_Sleep()
- IMO_Trim_Wakeup()

Test Project Description

Two types of projects are attached to this application note: a library project with the components and a test project to implement them. The name of the library component project is *Trim_Components.cyprj*. The test project is *AN80248.cyprj* and works with PSoC 3 by default.

You can add the Trim_Component library to any project. In PSoC Creator, select **Project** > **Dependencies** and specify the directory in which *Trim_Component.cyprj* is located. This can be done by clicking the folder icon under 'User Dependencies'. For more information on how to create and use library projects, see PSoC Creator help articles "Library Component Project" and "Basic Hierarchical Design Tutorial".

TopDesign Schematic

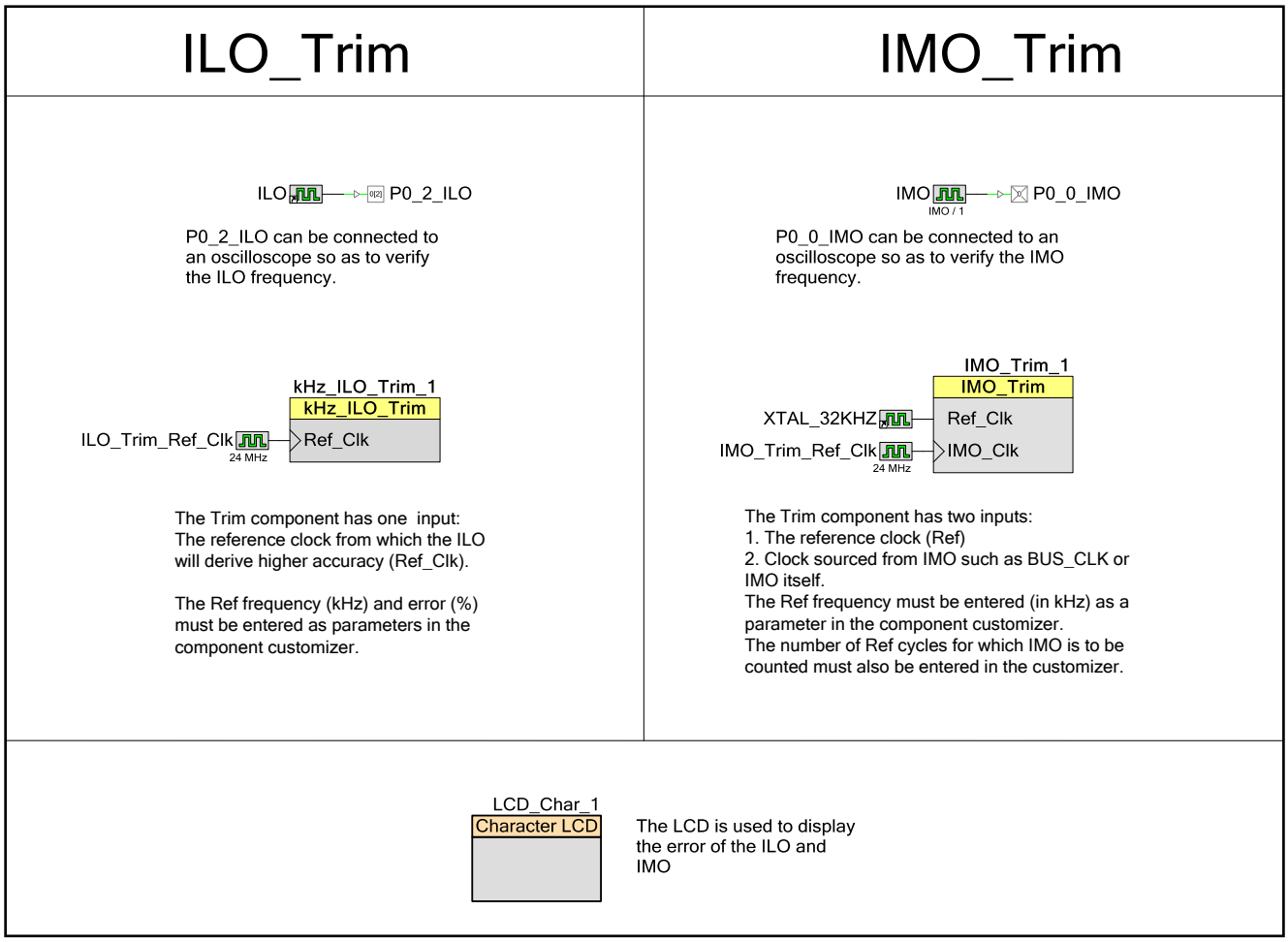
Figure 9 shows the TopDesign schematic of *AN80248.cyprj*. In this project, MASTER_CLK is set at 24 MHz and is derived from the 3 MHz IMO using the PLL.

A fixed-frequency 24 MHz clock sourced from the MASTER_CLK is the reference input to the ILO Trim component. You can verify the ILO frequency on P0[2].

The reference for IMO_Trim is the 32 kHz crystal. A fixed-frequency 24 MHz clock feeds IMO_Clk. You can verify the IMO frequency on P0[0].

The character LCD on P2[6:0] displays the IMO and ILO error in percentage.

Figure 9. PSoC Creator Schematic for Trim Component Test Project



main.c

At the beginning of the main block (see Appendix C), we initialize temporary variables to store and display IMO and ILO error, start the LCD, start the two components, and enable global interrupts.

In the infinite 'for' loop, the IMO and ILO frequency errors are calculated by using the CheckError function. If the IMO frequency error is calculated successfully, it is displayed on the character LCD as a percentage. IMO trimming is restarted if the IMO accuracy is outside a ±0.05% range.

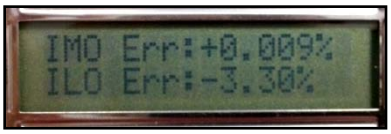
Similarly, the ILO frequency error is displayed on the LCD as a percentage. Typically, you would not use both CheckStatus and CheckError functions in the same design. However, for the purpose of illustration, we use both these functions in the attached project. ILO trimming is restarted if the CheckStatus function returns `kHz_ILO_Trim_1_IS_INACCURATE`.

Note The `kHz_ILO_Trim_1_CheckError()` function returns ILO error in parts per thousand, while the `IMO_Trim_1_CheckError()` function returns IMO error in parts per million.

Expected Output

- The frequency of the IMO output on P0[0] should be within ±0.05% accuracy.
- The frequency of the ILO output on P0[2] should be within ±7.5% accuracy.
- The LCD should show ILO, IMO accuracy in percent (Figure 10).

Figure 10. Example of Expected Output on LCD



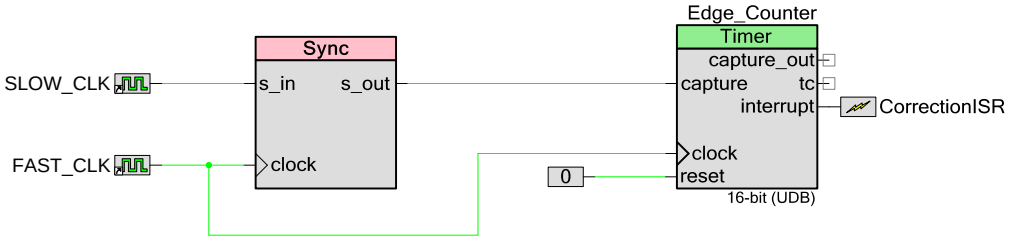
Component Internal Implementation

This section describes the internal working of the ILO and IMO Trim components. Figure 11 shows the generic PSoC Creator schematic used to trim either the ILO or IMO. The timer component is perfectly suited for edge counting and is configured in continuous 16-bit UDB mode. The timer starts at its period value and counts down to zero for every positive edge of FAST_CLK. Once at zero, the timer reloads and then continues this cycle.

The timer captures the 'count' value every 'Capture Count' cycles of SLOW_CLK and stores this in a 4-capture deep FIFO. Note that 'Capture Count' is a user-defined parameter in the Timer configuration window; its default value is 1. The number of edges of FAST_CLK per 'Capture Count' SLOW_CLK cycles is then calculated in firmware by subtracting consecutive captures. The Sync block ensures timing constraints are met within the timer.

Note In the projects attached to this application note when you trim the ILO, the ILO is SLOW_CLK and the IMO is FAST_CLK. When you trim the IMO, the clock from the 32 kHz crystal is SLOW_CLK, and the IMO is FAST_CLK.

Figure 11. Generic Schematic for Trimming



ILO Trim Design Considerations

When trimming the ILO, the ILO is the SLOW_CLK. The sources for FAST_CLK can be the 32 kHz crystal oscillator, the MHz oscillator or the IMO. This application note discusses the third case – trimming the ILO with the IMO (FAST_CLK) as reference.

Accuracy Attainable

The accuracy attainable by trimming the ILO depends on 3 factors:

- Ratio of FAST_CLK to SLOW_CLK
- Accuracy of the Reference
- Resolution of trim registers

Ratio of FAST_CLK to SLOW_CLK

The first step to trim the ILO is to count the edges of the FAST_CLK per cycle of the ILO. In other words,

$$edgeCount = \frac{\text{FAST_CLK}}{\text{ILO}}$$

Let the ILO error be z%, where z is small and can be either negative or positive. Then:

$$edgeCount = \frac{\text{FAST_CLK}}{\text{ILO} \times (1 + \text{z})}$$

If we denote the ratio of FAST_CLK to ILO by k, and apply a math result,

$$edgeCount \approx \text{k} \times (1 - \text{z})$$

Or

$$edgeCount \approx \text{k} - \text{k} \times \text{z}$$

Notice that k is the ideal number of edges that would be counted if the ILO were perfectly accurate. The term k×z arises due to the ILO error. For the timer to be able to resolve the change in edgeCount due to the k×z term, k×z must be greater than 1. So if an ILO error of z=0.1% must be resolved, then k should be greater than 1000 so that k×z is greater than 1. When trimming the 1 kHz ILO with the IMO as reference, this ratio is high enough so as not to introduce appreciable error.

Accuracy of the Reference

In the previous derivation, we assume that FAST_CLK is perfectly accurate. However, the error of the reference clock directly adds to the minimum error output from the trim procedure. The 3 MHz output is the most accurate (±1% in CY8C36xx and CY8C38xx devices) range of the IMO and hence is chosen to source FAST_CLK.

It is interesting to note that the IMO can directly source FAST_CLK or be sent through the PLL or clock dividers, which introduce no additional error. As discussed earlier, the upper bound for this frequency is limited to 32 MHz.

Note Even in the worst case, the 16-bit timer does not dictate the limit on FAST_CLK frequency. When ILO is at lower limit of accuracy (500 Hz), the maximum FAST_CLK frequency that the 16-bit timer can handle without overflow is 65536 × 500 ≈ 32.7 MHz > 32 MHz.

Resolution of Trim Registers

How closely the ILO frequency can approach 1 kHz depends on the resolution (i.e. the Hz/bit) of the trim registers. As shown in Figure 2, there are two trim ranges: fine trim and coarse trim. I have observed that the coarse

trim bit ct_range is not necessary when trimming the 1 kHz ILO. Thus, we are left with 6 bits (4 bits of tr_1k and 2 bits of ctrim) to trim the ILO.

From simulated ILO data over process, voltage and temperature, the maximum fine trim step is 128 Hz. This means that the closest that we can get to 1 kHz with certainty is ±128/2 = ±64 Hz or ±6.4%.

The effective accuracy of the trim procedure depends on the cumulative error from the 3 sources just discussed. Adding the three, the total error expected say for PSoC 3 is less than ±7.5%.Thus, we can trim the ILO to within a range of (925, 1075) Hz.

Trim Procedure

The trim procedure implementation is completely in firmware:

Start Function: Starts all the components, but disables the CorrectionISR. Similar to components in the Cypress Component catalog, there are associated Stop, Sleep and Wakeup functions.

BeginTrimming Function: Enables the CorrectionISR.

CorrectionISR: Is an interrupt service routine triggered every second timer capture. These two capture values are used to calculate the edgeCount. The edgeCount is then compared to the upper (+6.5% + Ref_Error%) and lower (-6.5% – Ref_Error%) bounds for counted edges. If the edgeCount is outside this range, then the frequency of the ILO is adjusted by a unit trim step based on the sign of the error. If the edgeCount is within this range, then the CorrectionISR disables itself. For simplicity, a sequential search method is employed instead of a binary/proportional search algorithm.

CheckStatus Function: Checks whether the ILO frequency has drifted outside the optimal frequency range. If it has, the BeginTrimming function can be called.

CheckError Function: Similar to the CheckStatus function, but calculates the ILO error in parts per thousand.

IMO Trim Design Considerations

When trimming the IMO, the FAST_CLK is an IMO-sourced clock (either the IMO directly or via the PLL/clock dividers) and the SLOW_CLK is the 32 kHz crystal oscillator output. The design considerations for the IMO trim component are very similar to that of the ILO.

Accuracy Attainable

The accuracy attainable by the IMO trim component depends on the same three factors as the ILO trim component:

- Ratio of IMO_CLK to the 32 kHz clock
- Accuracy of the 32 kHz crystal
- Resolution of trim registers

Ratio of IMO_CLK to 32 kHz Clock

Assume that the edges of the IMO are counted for every n (Averaging_Duration) cycles of the 32 kHz clock. Then:

$$edgeCount = \frac{\text{IMO_CLK}}{\frac{\text{32kHz_CLK}}{\text{n}}}$$

Let error of the IMO_CLK be z%, where z is small and can be either negative or positive. Then:

$$edgeCount = \text{IMO_CLK} \times (1 + \text{z}) \times \frac{\text{n}}{\text{32kHz_CLK}}$$

If we denote the ratio of IMO_CLK to 32kHz_CLK by k,

$$edgeCount = \text{k} \times \text{n} \times (1 + \text{z})$$

Or

$$edgeCount = \text{k} \times \text{n} + \text{k} \times \text{n} \times \text{z}$$

Note that k×n is the ideal number of edges which would be counted if the IMO were perfectly accurate. The term k×n×z arises due to the IMO error. For the timer to be able to resolve its impact on edgeCount, k×n×z must be greater than 1. For example, if we need to resolve an IMO error of z=±0.01%, then k×n should be greater than 10,000 so that k×n×z is greater than 1. In the component, n (Averaging_Duration) is a user-supplied parameter, while k depends on the frequency used for the IMO_CLK input.

Accuracy of the 32 kHz Crystal

The error of the 32 kHz crystal directly adds to the minimum error output from the trim procedure. Most crystals have an effective error in the range of ±0.01%.

Resolution of Trim Registers

As shown in Figure 3, there are 11 trim bits for the IMO. From empirical data, the step size (kHz/bit) across frequency, voltage and temperature is 0.06% of the IMO frequency per LSB of trim. This means that the closest we can get to a selected IMO frequency is ±0.03%.

Adding the three error sources, we can expect the accuracy of the IMO when using the trim component to be better than ±0.05%.

Trim Procedure

The firmware structure for trimming the IMO is exactly the same as that for the ILO. The only difference is in the values (frequencies, accuracy range) used within the routines.

Component Performance

Table 2 and Table 3 list the performance of the two trim components in terms of accuracy, lock-time, and resource usage.

Table 2. kHz_ILO_Trim Performance

Performance		
CPU cycles per iteration	314 (PSoC 3), 148 (PSoC 5)	
CPU percentage used*	0.65% (PSoC 3) 0.31% (PSoC 5)	
Accuracy wrt Ref_Clk	±6.5%	
Lock Time**	32 ms	
Code (bytes)	670 (Flash) + 22 (RAM)	
Resources	**Fraction**	**Percentage**
Datapath	2 of 24	8.3%
Status cells	1 of 24	4.2%
Control/count7 cells	1 of 24	4.2%
PLDs	2 of 48	4.2%
Macrocells	6 of 192	3.1%
Interrupts	1 of 32	3.1%
Digital clock dividers	0 of 8	0.0%
Sync blocks	1 of 92	2.2%

*Assuming 1 fine trim iteration every 2nd ILO cycle
** Worst lock time – from +100% error to minimum error

Table 3. IMO_Trim Performance

Performance		
CPU cycles per iteration	312 (PSoC 3), 147 (PSoC 5)	
CPU %age used*	0.31% (PSoC 3) 0.14% (PSoC 5)	
Accuracy wrt Ref_Clk	±0.05%	
Lock Time**	125 ms	
Code (bytes)	725 (Flash) + 24 (RAM)	
Resources	**Fraction**	**Percentage**
Datapath	2 of 24	8.3%
Status cells	1 of 24	4.2%
Control/count7 cells	2 of 24	8.3%
PLDs	3 of 48	6.3%
Macrocells	8 of 192	4.2%
Interrupts	1 of 32	3.1%
Digital clock dividers	0 of 8	0.0%
Sync blocks	1 of 92	2.2%

*Assuming 1 trim iteration every 140 32 kHz cycles
** Worst lock time – from +1% error to minimum error

Summary

The PSoC contains two internal clock sources – the ILO and IMO. Using the components presented in this application note, both the ILO and IMO can be trimmed during run time to improve their accuracy. The ILO was shown to achieve ±6.5% accuracy with respect to the reference clock, while the IMO could be trimmed down to ±0.05%. This adds a powerful feature to the unique, flexible clocking system in PSoC.

About the Author

Name:	Antonio Rohit De Lima Fernandes
Title:	Applications Engineer
Background:	B.E in Electrical and Electronics, BITS, Pilani, Rajasthan, India.
Contact:	anto@cypress.com

Appendix A – IMO and ILO Frequency Variation Graphs

Figure 12. ILO Frequency Variation vs. V_{DDD} in PSoC 3

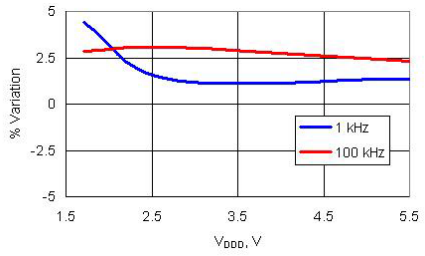


Figure 13. ILO Frequency Variation vs. T in PSoC 3

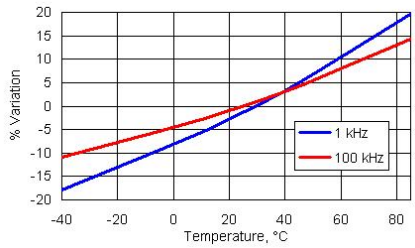


Figure 14. ILO Frequency Variation vs. V_{DDD} in PSoC 5

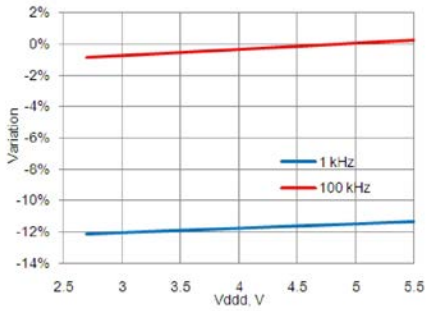


Figure 15. ILO Frequency Variation vs. T in PSoC 5

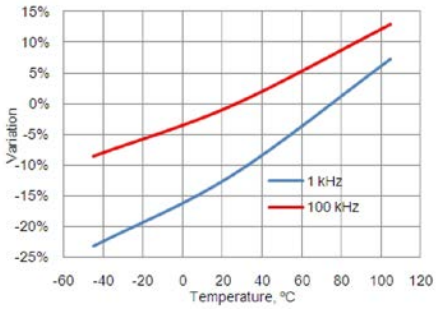


Figure 16. IMO Frequency Variation vs. V_{CCD} in PSoC 3

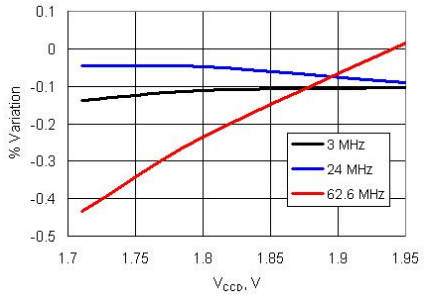


Figure 17. IMO Frequency Variation vs. T in PSoC 3

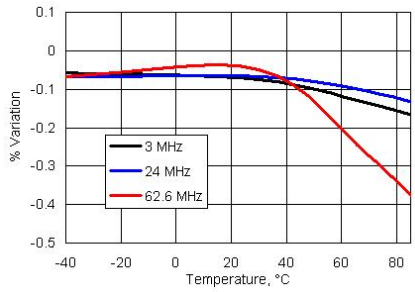


Figure 18. IMO Frequency Variation vs. V_{DDD} in PSoC 5

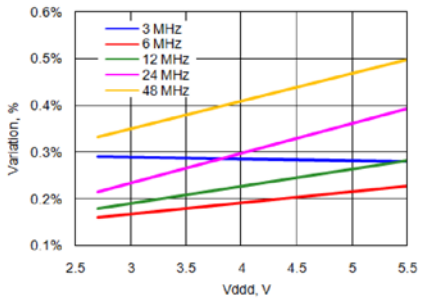
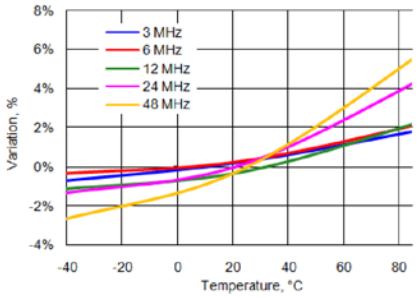


Figure 19. IMO Frequency Variation vs. T in PSoC 5



Appendix B – Identifying Nominal Clock Frequencies

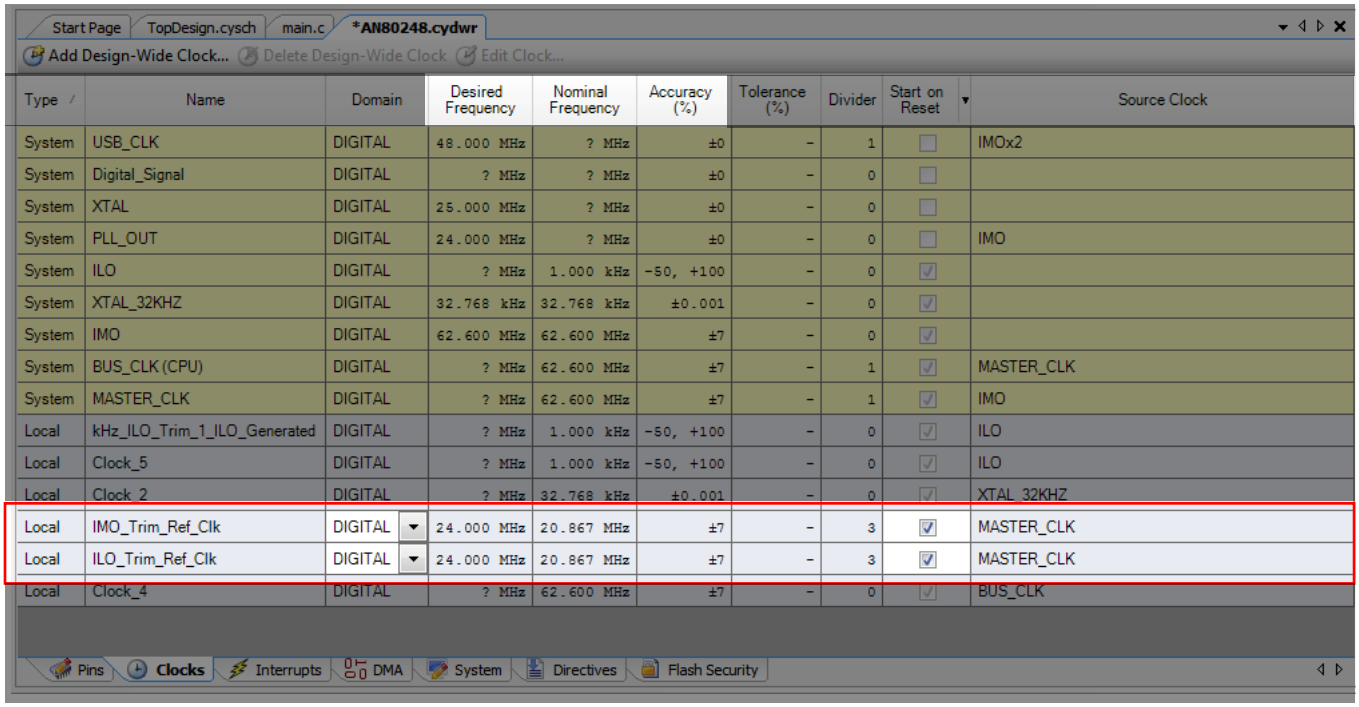
Figure 20 shows the Clocks tab of the design-wide resources file. The actual frequencies of all the clocks in your design appear under the **Nominal Frequency** column. There is also a **Desired Frequency** column which displays the expected/desired frequency of all fixed-frequency clocks in the system. If the desired frequency for a user-defined clock can be obtained through a 16-bit integer division of any of the enabled source clocks (MASTER_CLK, PLL, IMO, ILO, XTALs), then the desired and nominal frequency will be equal.

For the purpose of illustration, the IMO is set to 62.6 MHz, and the MASTER_CLK is sourced directly from the IMO. The PLL is disabled. As you can see in the figure, though the desired frequency for both IMO_Trim_Ref_Clk and ILO_Trim_Ref_Clk is 24 MHz, the nominal frequency is 20.867 MHz. The frequency value entered in the component customizer must be this *nominal frequency*.

Another important piece of information conveyed by Figure 20 is clock accuracy in percentage values. For each clock in your system, there is a corresponding accuracy value listed under the **Accuracy** column.

Note To make desired and nominal frequencies the same in this case, set the desired frequency to an integer-divided MASTER_CLK value. One example of such a setting is IMO_Trim_Ref_Clk and ILO_Trim_Ref_Clk configured for 31.3 MHz which is MASTER_CLK/2.

Figure 20. System Clocks Tab in .cydwr File



Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	Source Clock
System	USB_CLK	DIGITAL	48.000 MHz	? MHz	±0	-	1		IMOx2
System	Digital_Signal	DIGITAL	? MHz	? MHz	±0	-	0		
System	XTAL	DIGITAL	25.000 MHz	? MHz	±0	-	0		
System	PLL_OUT	DIGITAL	24.000 MHz	? MHz	±0	-	0		IMO
System	ILO	DIGITAL	? MHz	1.000 kHz	-50, +100	-	0	✓	
System	XTAL_32KHZ	DIGITAL	32.768 kHz	32.768 kHz	±0.001	-	0	✓	
System	IMO	DIGITAL	62.600 MHz	62.600 MHz	±7	-	0	✓	
System	BUS_CLK (CPU)	DIGITAL	? MHz	62.600 MHz	±7	-	1	✓	MASTER_CLK
System	MASTER_CLK	DIGITAL	? MHz	62.600 MHz	±7	-	1	✓	IMO
Local	kHz_ILO_Trim_1_ILO_Generated	DIGITAL	? MHz	1.000 kHz	-50, +100	-	0	✓	ILO
Local	Clock_5	DIGITAL	? MHz	1.000 kHz	-50, +100	-	0	✓	ILO
Local	Clock_2	DIGITAL	? MHz	32.768 kHz	±0.001	-	0	✓	XTAL_32KHZ
Local	IMO_Trim_Ref_Clk	DIGITAL	24.000 MHz	20.867 MHz	±7	-	3	✓	MASTER_CLK
Local	ILO_Trim_Ref_Clk	DIGITAL	24.000 MHz	20.867 MHz	±7	-	3	✓	MASTER_CLK
Local	Clock_4	DIGITAL	? MHz	62.600 MHz	±7	-	0	✓	BUS_CLK

Appendix C – main.c Code for the Test Project

```
void main()
{
    int16 errIMO, errILO;  /* temporary variables to store IMO, ILO error */
    char dispError[8];            /* buffer to print to LCD display */

    LCD_Char_1_Start();

    LCD_Char_1_Position(0,0);
    LCD_Char_1_PrintString("IMO Err:");

    LCD_Char_1_Position(1,0);
    LCD_Char_1_PrintString("ILO Err:");

    IMO_Trim_1_Start();
    IMO_Trim_1_BeginTrimming();

    kHz_ILO_Trim_1_Start();
    kHz_ILO_Trim_1_BeginTrimming();

    /* The components use interrupts - so enable global interrupts */
    CyGlobalIntEnable;

    for(;;)
        {
        errIMO = IMO_Trim_1_CheckError();       /* in parts per million */
        errILO = kHz_ILO_Trim_1_CheckError();   /* in parts per thousand */

        /* Similar to the CheckStatus function, the CheckError function returns
         * the value ISR_ON if the ISR is running, and NOT_READY if the Timer
         * FIFO is not full.
         */
        if((errIMO != IMO_Trim_1_ISR_ON) && (errIMO != IMO_Trim_1_NOT_READY))
        {
            /* Display IMO Error on LCD */
            LCD_Char_1_Position(0,8);
            sprintf(dispError, "%+.3f", (float)errIMO/10000); /* convert to % */
            LCD_Char_1_PrintString(dispError);
            LCD_Char_1_Position(0,14);
            LCD_Char_1_PrintString(" %");

            /* One method to ascertain whether trimming is required:
             * If the IMO accuracy is outside +/-0.05%, begin trimming.
             * IS_BETWEEN(VALUE, LOW, HIGH) is defined in the component's .h file.
             * It checks whether value is between the low, high bounds specified.
             */
            if(!IS_BETWEEN(errIMO, -500, 500))
            {
                IMO_Trim_1_BeginTrimming();
            }
        }

        if((errILO != kHz_ILO_Trim_1_ISR_ON) && (errILO != kHz_ILO_Trim_1_NOT_READY))
        {
            /* Display ILO Error on LCD */
            LCD_Char_1_Position(1,8);
            sprintf(dispError, "%+.3f", (float)errILO/10);    /* convert to percent */
            LCD_Char_1_PrintString(dispError);
            LCD_Char_1_Position(1,13);
            LCD_Char_1_PrintString("%  ");
```

```
        }

        /* Give a sufficient delay between CheckError and CheckStatus */
        CyDelay(100);

        /* The other method which can be used to check whether trimming is
         * required is by using the CheckStatus function. If the return value
         * is IS_INACCURATE, the BeginTrimming() function is called.
         */
        if(kHz_ILO_Trim_1_CheckStatus() == kHz_ILO_Trim_1_IS_INACCURATE)
        {
            kHz_ILO_Trim_1_BeginTrimming();
        }

/* This delay allows us to observe the LCD clearly */
        CyDelay(500);
    }
}
```

Document History

Document Title: PSoC® 3 / PSoC 5 Improving the Accuracy of Internal Oscillators - AN80248

Document Number: 001-80248

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3680731	ANTO	07/17/2012	New Application Note.
*A	3712498	ANTO	08/14/2012	Updated IMO Basics. Updated Component Usage (Updated kHz_ILO_Trim (Updated Parameters), updated IMO_Trim (Updated APIs), updated Test Project Description (Updated TopDesign Schematic, updated *main.c*, added Expected Output)). Updated Component Internal Implementation (Updated ILO Trim Design Considerations (Updated Trim Procedure)). Updated Component Performance. Updated Summary. Updated Appendix A – IMO and ILO Frequency Variation Graphs. Updated Appendix B – Identifying Nominal Clock Frequencies. Added Appendix C – main.c code for the Test Project.
*B	3793652	ANTO	09/25/2012	Updated document to clearly state that the ±1% spec for IMO is for CY8C36 and CY8C38 parts. Updated components with bug fixes. Uploaded a single example project for the AN.

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

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

PSoC is a registered trademark of Cypress Semiconductor Corp. PSoC Creator is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor | Phone : 408-943-2600
198 Champion Court | Fax : 408-943-4730
San Jose, CA 95134-1709 | Website : www.cypress.com

© Cypress Semiconductor Corporation, 2012. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.