Application Report
SLAA489B–January 2011–Revised September 2019

UCS10 Guidance

ABSTRACT

This document describes erratum UCS10, its possible effect on applications, and suggested workarounds.

This erratum can be observed in certain MSP430F5xx and MSP430F6xx devices using the DCO output with the FLL enabled, producing occasional jumps in the output frequency of approximately 10% (see the device-specific errata sheet to determine if your specific device and revision is affected by UCS10). As a result of this behavior, any clock driven by DCO is affected for a short time until the FLL readjusts to the expected frequency.

Even when the processor is affected by this behavior, the effect on the application could be negligible in some cases. Affected applications can implement a workaround as suggested in this document, as best applicable to the particular application.

Contents

Trademarks

1 Description

1.1 Expected Behavior

As described in the MSP430F5xx and MSP430F6xx Family User's Guide, the FLL uses five DCO bits and five MOD (modulation) bits to adjust the output frequency by counting up or counting down a frequency integrator. The output of the frequency integrator that drives the DCO can be read in UCSCTL0 bits MOD and DCO (see Figure 1).

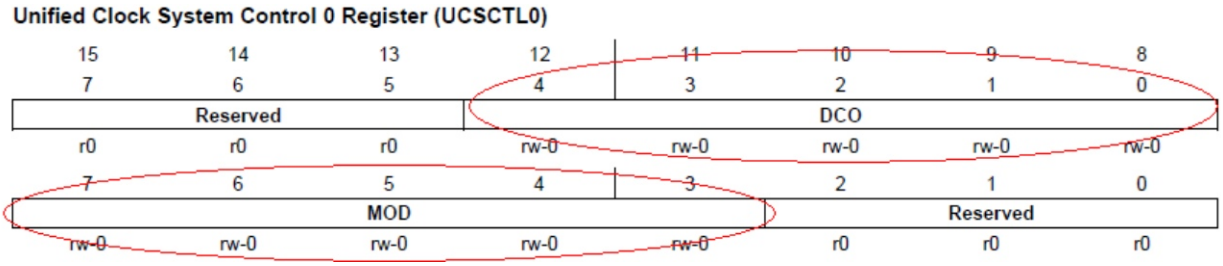

Figure 1. DCO and MOD Bits in UCSCTL0 Register

As part of its operation, the DCO register value is incremented when the modulation counter rolls over and, in the same way, it is decremented when the modulation decrements from 0 to the maximum count.

A change in the DCO bits represents a 2% to 12% change as documented in the data sheet:

S_{DCO}	Frequency step between tap DCO and DCO + 1	$S_{DCO} = f_{DCO(DCORSEL,DCO+1)}/f_{DCO(DCORSEL,DCO)}$	1.02	1.12	ratio

This step size, however, should be minimized under normal circumstances by the MOD bits changing from the maximum count to 0 or vice versa, allowing for a smooth transition between DCO steps.

1.2 FLL Behavior During FLL Jump

The FLL jump behavior is observed when the MOD counter is expected to roll over, and the DCO bits increment, but the MOD bits are decremented (see Figure 2).

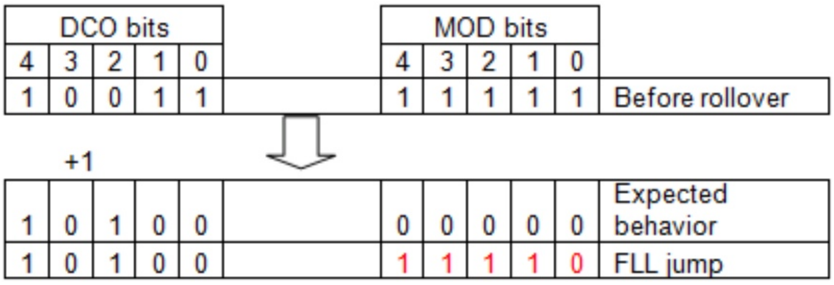

Figure 2. MOD and DCO Bits During FLL Jump

When this happens, because the DCO bits are changing to DCO+1, the output is expected to change up to 12%, per the data sheet.

Figure 3 shows the output frequency when this behavior occurs:

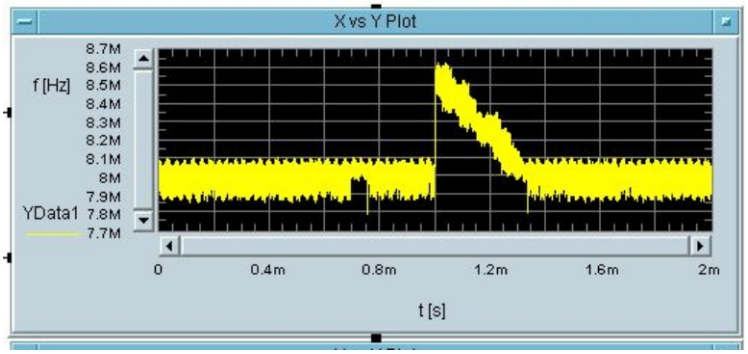

Figure 3. Output Frequency During Jump

In this case, the expected frequency from the DCO is 8 MHz but the output jumps up to approximately 8.6 MHz. The frequency returns to normal after approximately 250 µs.

Figure 4 shows an FFT of the output frequency during another occurrence of this behavior, in this case at 16 MHz.

Figure 4. FFT of Output Frequency

In both cases, it can be seen that the frequency jumps approximately 8% and then returns to normal after several FLL readjustments.

Because the FLL uses FLLREFCLK/FLLREFDIV as a reference, the time it takes to readjust the FLL depends on this reference frequency and not the DCO output.

The cases shown in Figure 3 and Figure 4 both use a 32.768-kHz reference (30.5-µs period) and take 8 to 10 cycles to return to normal. Using a slower or faster FLLREFCLK/FLLREFDIV results in a slower or faster settling time, respectively.

NOTE: This problem might happen only when the MOD bits are at the maximum value. Due to variations in the microcontrollers, the value of DCO+MOD bits at a particular frequency might be different from device to device.

At the same time, the DCO output is affected by temperature and supply voltage, so the DCO+MOD bits values are different depending on the environment. This can cause an intermittent affect in applications, where some devices appear to fail and others do not, when devices fail after some random time, or devices fail when temperature or supply voltage change.

Despite these variations, the problem can be replicated in all devices, and it presents itself depending on the factors described above, such as DCO output frequency and temperature.

2 Application Effects

Because the output of the DCO is affected, any clock being driven from the DCO is also affected, including CPU frequency and peripherals.

While synchronous operations (SPI) are affected, the impact should be minimal in most cases. Time dependant operations (PWM, RTC) or asynchronous operations (UART) might be impacted more severely.

2.1 Synchronous Communication (SPI)

The CLK frequency is affected but, because this clock is provided as part of the communication, the receiving end remains synchronized. It is important to consider the maximum frequency allowed by other devices in the bus to avoid violating their specifications when the frequency jumps.

2.2 Time-Dependent Operation (PWM, RTC)

PWMs, RTCs, and other time-dependent operations based on the DCO are affected by the same jump in frequency. The effects of the change in frequency are minimized when the period of these time-dependant operations is large enough to absorb the change, but the change has a greater effect in short time periods.

- With a periodic interrupt every 100 ms, a jump of approximately 8% that returns to normal after 250 µs represents a minimal change in the total percentage.
- A PWM at 4 kHz or higher, an 8% jump causes an undesirable effect for one or more cycles.

2.3 Asynchronous Communications (UART)

The level of effect on asynchronous communications such as UART depends on the frequency of their communications.

Because this kind of communications does not provide a CLK signal, devices are synchronized at the beginning of the frame and depend on the frequency being stable enough to sample the incoming data. When the FLL jumps, the frequency of the data changes and incorrect data can be sampled in or sent out, resulting in incorrect data or an overrun error. The problem is minimized when the communication frequency is low, and the problem is maximized when the communication frequency is high.

- If the baudrate is 115200 with a bit time of 8.6 µs and a frame time of approximately 86 µs, the clock jump affects a whole frame, causing an expected deviation of approximately 84% and producing data errors.
- If the baudrate is 19200 with a bit time of 52 µs and a frame time of 520 µs, the clock jump affects only part of the frame, causing a deviation of approximately 97%, which is acceptable for UART communication.

3 Workarounds

Applications affected by this behavior can implement the workarounds shown in Section 3.1 through Section 3.3.

3.1 Avoid Using DCO

Applications that do not use the DCO but use another clock source such as a crystal are not affected by this erratum.

3.2 Decrease Frequency of Affected Peripherals

The FLL jump change can be absorbed over time during long periods. Decreasing the frequency of the affected peripheral (for example, lowering the baudrate for UART) helps reduce the effect of the FLL jump.

3.3 Disable the FLL

If the FLL is disabled, the DCO+MOD bits do not change, and the problem does not happen. Some considerations are:

- There is a chance that the FLL could be disabled exactly when the DCO is out of frequency. It is important to compare the DCO against a known stable reference.
- The DCO drifts over temperature and voltage. It is important to readjust the DCO+MOD bits periodically. This period depends on the application but, in most cases, the temperature and voltage is not expected to change drastically, which allows for long periods between adjustments.
- While the FLL is being adjusted and until the frequency is checked against a reference, the functionality is not reliable and affected peripherals should be disabled.

3.3.1 Periodic FLL Trim Implementation

The flowchart in Figure 5 shows an implementation of a periodic FLL trim:

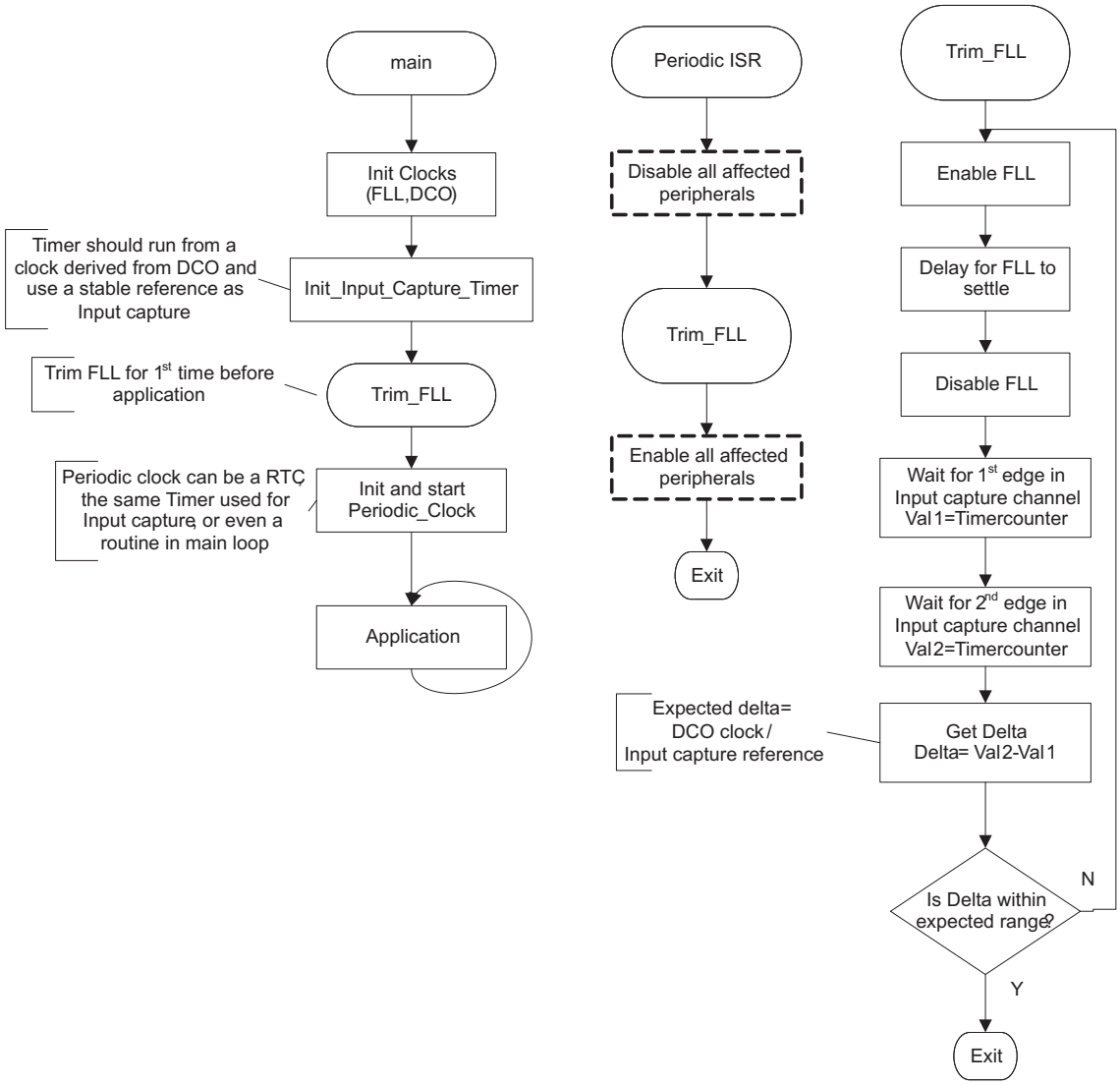

Figure 5. Flowchart of Periodic FLL Trim

3.3.1.1 Hardware

Any periodic timer or even a periodically executed routine in the main loop can be used as a time base to generate the periodic event used to readjust the FLL. Timer_A and Timer_B have input capture functionality and can be used for this implementation. This timer should run from a clock derived from the DCO (for example, SMCLK).

A stable reference is needed as an input for the timer. Taking advantage of the MSP430™ architecture, the reference for the input capture can be driven by an internal reference, thus removing any need for external connections.

The following example uses Timer_B0 in the MSP430F5438A as the source for periodic interrupts and input capture. Timer_B0 is driven by SMCLK, and the overflow routine is used to generate periodic interrupts. The input capture is connected internally to ACLK, which is sourced by an external 32.768-kHz crystal.

Figure 6 highlights the functionality used for Timer_B0:

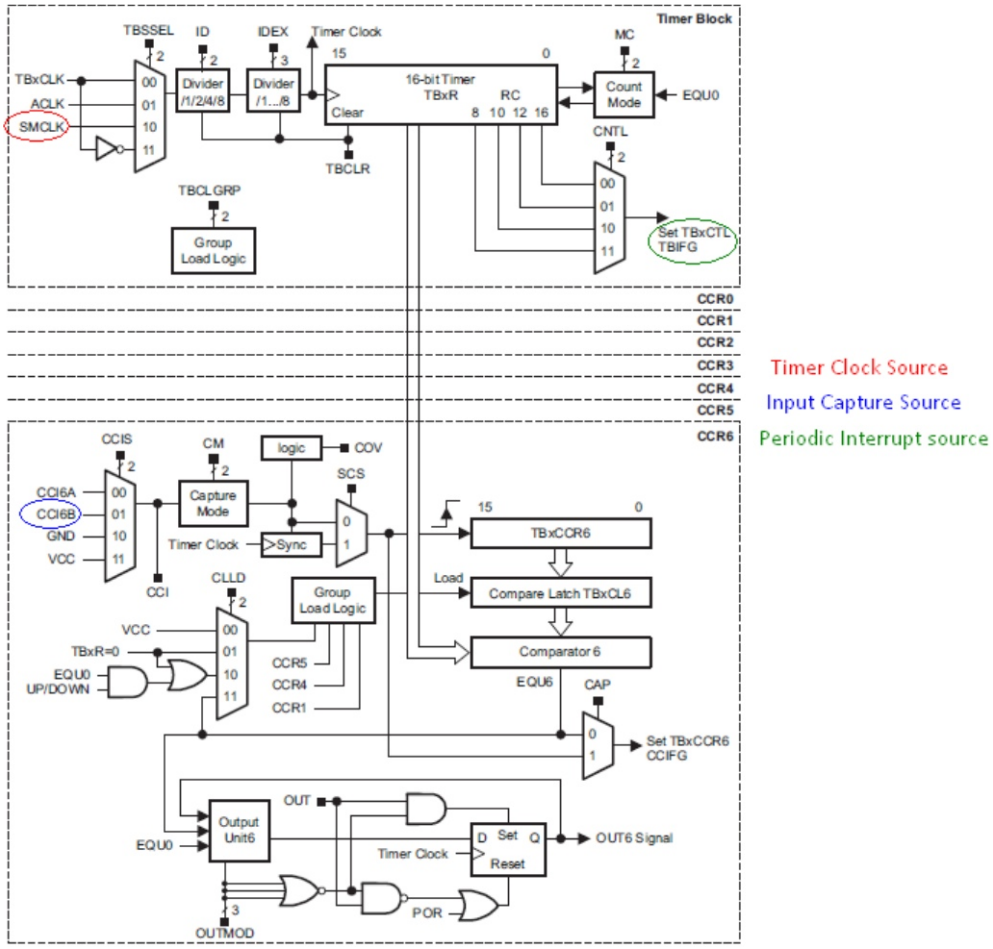

Figure 6. Timer_B0

The data sheet shows the internal connection from ACLK to CCI6B (see Figure 7).

Table 10. TB0 Signal Connections

INPUT PIN NUMBER		DEVICE INPUT SIGNAL	MODULE INPUT SIGNAL	MODULE BLOCK	MODULE OUTPUT SIGNAL	DEVICE OUTPUT SIGNAL	OUTPUT PIN NUMBER	
PZ/ZQW	PN						PZ/ZQW	PN
49/M11-P4.6	52-P4.6	TB0.6	CCI6A	CCR6	TB6	TB0.6	49/M11-P4.6	52-P4.6
		ACLK (internal)	CCI6B					
		DV_{SS}	GND					
		DV_{CC}	V_{CC}					

Figure 7. TB0 Signal Connections

NOTE: ACLK cannot be used as an internal capture input on devices affected by erratum TB20. For these devices, an external connection is required.

3.3.1.2 Software

Example 1. Clock Initialization - Periodic FLL Trim

```
P7SEL |= BIT0 + BIT1;                     // Port select XT1
  // Startup LFXT1 32 kHz crystal
  do{
    status = LFXT_Start_Timeout(XT1DRIVE_0, 50000);
  }while(status == UCS_STATUS_ERROR);


SELECT_FLLREF(SELREF__XT1CLK);          // Set DCO FLL reference = REFO
SELECT_ACLK(SELA__XT1CLK);              // Select XT1 as ACLK source
SELECT_SMCLK(SELS__DCOCLKDIV);          // Select DCO as SMCLK source
Init_FLL_Settle(16000,244);             // Init FLL at 16Mhz
```

Example 2. Timer Initialization - Periodic FLL Trim

```
// Start a timer to re-adjust fll , using SMCLK in continous mode
//    will interrupt every 16Mhz/65536
    TB0CTL = TBSSEL__SMCLK + MC__CONTINOUS;
// Use TB0.6 CCI6B which is internally connected to ACLK in Capture
//   on falling edge mode
    TB0CCTL6 = CM_2 + CCIS_1 + CAP;

    trim_fll();   // Adjust FLL for the first time before application

    TB0CTL |= TBIE;   // Enable Periodic timer after 1st trim
```

Example 3. Periodic Interrupt Service Routine- Periodic FLL Trim

```
// MCLK Frequency
#define MCLK_FREQ   16000000
// ACLK Frequency
#define ACLK_FREQ   32768

// Timer overflow Frequency
#define CLK__COUNT     (MCLK_FREQ/65536)

#pragma vector=TIMER0_B1_VECTOR
__interrupt void TimB0_Isr(void)
{
  static volatile unsigned long count_int =0;

switch(__even_in_range(TB0IV,14))
  {
    case  0: break;                          // No interrupt
    case  2: break;                          // CCR1 not used
    case  4: break;                          // CCR2 not used
    case  6: break;                          // CCR3 reserved
    case  8: break;                          // CCR4 reserved
    case 10: break;                          // CCR5 reserved
    case 12: break;                          // CCR6 reserved
    case 14:  // Overflow
    // Disable any affected peripherals here (i.e. UART, PWMs)
    if (count_int++ > (CLK__COUNT))
    {   // Trim only after 1 second
        count_int=0;
        trim_fll();
    }
    TB0CTL &= ~TBIFG;
             break;
```

Example 3. Periodic Interrupt Service Routine- Periodic FLL Trim (continued)

```
    default: break;
  }
```

Example 4. trim_fll( ) Routine - Periodic FLL Trim

```
// Expected Delta
#define EXPECTED_DELTA (MCLK_FREQ/ACLK_FREQ)
// Tolerance (10 = ~2%)
#define TOLERANCE 10

void trim_fll(void)
{
    unsigned short val1, val2;
    do
    {
        __bic_SR_register(SCG0);       // Re-enable FLL
        __delay_cycles(500000);         //Delay waiting for FLL to settle: Max = 16Mhz/32768 * 32 * 32
        __bis_SR_register(SCG0);      // Disable FLL

        TB0CCTL6 &= ~CCIFG;        // Wait for 1st input capture
        while (!(TB0CCTL6&CCIFG))
          ;
        val1 = TB0CCR6;                     // Store value of 1st edge
        TB0CCTL6 &= ~CCIFG;                 // Wait for 2nd edge
        while (!(TB0CCTL6&CCIFG))
          ;
        val2 = TB0CCR6;                     // Store value of 2nd edge

     // Compare vs the expected Delta and repeat if value is out of range
    }while ( ((val2-val1) > (EXPECTED_DELTA+TOLERANCE)) || ((val2-val1) < (EXPECTED_DELTA-
TOLERANCE)));
```

3.3.1.3 Additional Comments

The routine shown in Example 4 adjusts the DCO clock output on every periodic interrupt regardless of its current actual status.

An alternative implementation could check the current status of the DCO clock using the timer's input capture and trim the clock only when the value is out of range. This could shorten the time spent in this routine, at least when the DCO is within the expected range.

3.3.2 Software FLL Implementation

The FLL can also be adjusted in software. It can be a more CPU-intensive solution compared to the example described in Section 3.3.1, but it offers the advantage that the bug never appear, because the DCO and MOD bits are adjusted manually.

Note that this software FLL implementation does not adjust DCORSEL if the DCO bits underflow or overflow; it just limits the values. Calling Init_FLL_Settle( ) sets DCORSEL within the expected range during initialization.

The flowchart in Figure 8 shows an implementation of a software FLL trim:

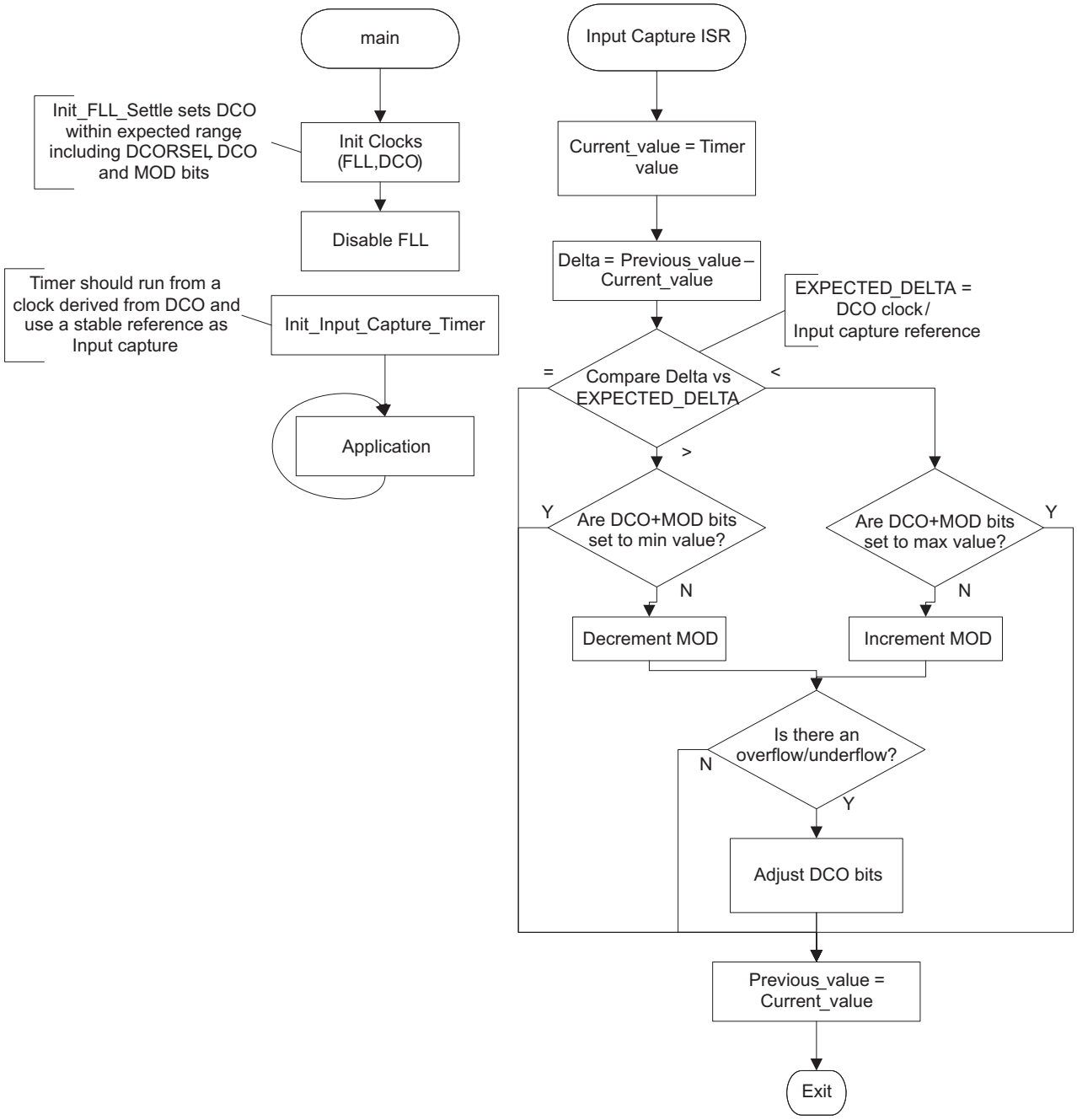

Figure 8. Flowchart of Software FLL Trim

3.3.2.1 Hardware

Hardware for this example is similar to the example in Section 3.3.1. This example uses Timer_B0 in the MSP430F5438A for input capture. Timer_B0 is driven by SMCLK, and the overflow routine is used to generate periodic interrupts. The input capture is connected internally to ACLK, which is sourced by an external 32.768-kHz crystal.

3.3.2.2 Software

Example 5. Clock Initialization - Software FLL

```
// MCLK Frequency: MCLK/SMCLK frequency
#define SMCLK_FREQ   16000000
//#define SMCLK_FREQ   (32768*473)
// ACLK Frequency: XT1 frequency
#define ACLK_FREQ   32768

// Expected Delta: this is the expected value when comparing ACLK vs SMCLK
#define EXPECTED_DELTA ((SMCLK_FREQ/ACLK_FREQ))

P7SEL |= BIT0 + BIT1;                     // Port select XT1
  // Startup LFXT1 32 kHz crystal
  do{
    status = LFXT_Start_Timeout(XT1DRIVE_0, 50000);
  }while(status == UCS_STATUS_ERROR);


SELECT_ACLK(SELA__XT1CLK);              // Select XT1 as ACLK source
SELECT_SMCLK(SELS__DCOCLKDIV);          // Select DCO as SMCLK source
// Initialize the FLL at 16Mhz
Init_FLL_Settle(SMCLK_FREQ/1000,(SMCLK_FREQ/32768));

// Now that Hardware FLL is close to target, disable and continue
//  with Sofware FLL
  __bis_SR_register(SCG0);      // Disable FLL loop control
```

Example 6. Timer Initialization - Software FLL

```
// Start a timer to Re-adjust fll , using SMCLK in continous mode
    //  will interrupt on every input catpure
    TB0CTL = TBSSEL__SMCLK + MC__CONTINOUS;
    // Use TB0.6 CCI6B which is internally connected to ACLK in Capture on
    //  falling edge mode
    TB0CCTL6 = CM_2 + CCIS_1 + CAP + CCIE;
    TB0CTL |= TBIE;   // Enable Periodic timer after 1st trim
```

Example 7. Input Capture ISR - Software FLL

```
#pragma vector=TIMER0_B1_VECTOR
__interrupt void TimB0_Isr(void)
{
    static unsigned char first_pass = 0;
    static unsigned short previous, current, ucs_dco_mod;

    switch(__even_in_range(TB0IV,14))
    {
        case  0: break;                          // No interrupt
        case  2: break;                          // CCR1 not used
        case  4: break;                          // CCR2 not used
        case  6: break;                          // CCR3 reserved
        case  8: break;                          // CCR4 reserved
```

Example 7. Input Capture ISR - Software FLL (continued)

```
        case 10: break;                          // CCR5 reserved
        case 12:    //CCR6: 32.768Khz edge
            current = TB0CCR6;
            // Get current dco_mod value ignoring reserved bits
            ucs_dco_mod = (UCSCTL0 & 0x1FF8);
            if (first_pass != 0)
            {
                // Wait until the second pass to start trimming FLL
                if (EXPECTED_DELTA < (current - previous))
                {
                    // Too fast, slow down
                    if (ucs_dco_mod != 0x00)
                    {
                        // Decrement MOD+DCO bits if they haven't reached low limit
                        ucs_dco_mod -= MOD0;
                    }
                 /* else
                    {   // current DCORSEL settings don't support this frequency
                        // DCORSEL must be adjusted properly before using SW FLL
                    } */
                }
                else if (EXPECTED_DELTA > (current - previous))
                {
                    // Too slow
                    if (ucs_dco_mod != 0x1FF8)
                    {
                        // Increment MOD+DCO bits if they haven't reached high limit
                        ucs_dco_mod += MOD0; // Increment MOD bits
                    }
                 /* else
                    {   // current DCORSEL settings don't support this frequency
                        // DCORSEL must be adjusted properly before using SW FLL
                    } */

                }
                // Write the value to the register
                UCSCTL0 = (ucs_dco_mod & 0x1FF8);
            }
            else
            {
                // Just set the flag and wait for next time
                first_pass = 1;
            }
            previous = current;
            TB0CCTL6 &= ~CCIFG;
            break;
        default: break;
    }
}
```

4 Summary

The information in this document provides the reader with information that can be applied to assessing the susceptibility of an application to UCS10 erratum described herein. In addition to the information presented here, the performance of the application being assessed should also be considered as an important data point in determining risk. Whether or not failures occur in the field or at production of specific end equipment can serve as an additional indicator of robustness and the likelihood that UCS10 affects the specific application.

5 References

1. MSP430F5xx and MSP430F6xx Family User's Guide

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from March 15, 2011 to September 16, 2019	Page
• Added the note that begins "ACLK cannot be used as an internal capture input..." in Section 3.3.1.1, *Hardware*	7

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265