Application Report
SLAA336A – October 2006 – Revised August 2007

Using the DCO Library

Lane Westlund
MSP430 Applications

ABSTRACT

This document serves as an overview describing the use of the DCO library within existing C or assembly projects. This library encapsulates routines used for setting the DCO to a specific speed based on a multiplication of a known clock, such as a 32-kHz crystal. These functions are written in assembly to be optimized for the MSP430 but can be called from any C program that includes their header files.

1 Introduction

It is often necessary to have the digitally controlled oscillator (DCO) of an MSP430 tuned to a specific frequency. Due to device variances, simply setting the DCO register values to a constant value across all devices is not a method that can ensure accurate results. In order to have accuracy, the DCO must be tuned based on a known frequency. In 4xx devices, this is done automatically using the FLL. The purpose of the FLL is to keep the DCO running at a certain multiple of ACLK. In 1xx and 2xx devices, which do not have the FLL, a similar tuning process can be accomplished using software and timers. By using this software FLL to periodically calibrate the DCO, a high degree of accuracy can be achieved, enabling the DCO to run at any arbitrary frequency, provided the V_{CC} requirements are met.

Included with this application report are library files which are intended to be compiled in a larger project. These files include functions necessary for setting the DCO to a multiple of an external crystal.

Also included in the library are a series of code examples, intended to illustrate various methods of initializing and using the library.

2 Tuning the DCO – Theory

Every MSP430 comes with a DCO. The frequency at which the DCO oscillates can be adjusted by setting the DCO registers. In this way, the DCO can be tuned by incrementally changing the registers and comparing the resultant frequency against a known frequency. When the speed of a slower clock source is known, such as a 32-kHz watch crystal, the DCO speed can be adjusted until a specific number of DCO cycles occur in one cycle of the slower clock.

To accomplish this, a Capture/Compare Register of Timer_A is initialized in Capture mode. In this mode, it captures the value of Timer_A when a low-to-high transition occurs on an internal signal. In this case, the internal signal is ACLK. When SMCLK is driving the timer, the captured value becomes the number of SMCLK cycles since the last ACLK low-to-high transition.

When the number of SMCLK cycles are known, the DCO can be adjusted and measured again using the same capture method previously outlined. In this way, the DCO can be tuned to a specific multiple of the known ACLK frequency. For increased accuracy, the ACLK signal can be divided by eight to increase granularity.

3 Safety Factors

In order to ensure reliable and safe use of the DCO library, several safety measures have been added to the library.

In order to prevent the library from becoming trapped in an infinite loop, a maximum number of 10,000 loops are allowed. If the desired frequency is not set after 10,000 iterations, the function returns 0xFF, in order to signify the timeout has occurred. It should be noted that this timeout is loop count based and not time based. This means that a timeout does not occur if no ACLK signal is present.

On 2xx devices, the DCO speed should not exceed the specified 16 MHz. As an additional safety factor, 2xx devices will not increment the DCO settings above the value given in the factory-calibrated 16-MHz register settings. If this value is reached, the function exits, returning 0×02 to indicate the DCO is set to the fastest setting.

In order to ensure a more consistent time for setting the DCO, a tolerance has been built into the setDCO routine. The routine exits if the measured DCO is exactly set to the desired multiplier or if the DCO speed is one MCLK slower.

4 DCO_Library – Usage From C

```
#include <msp430x11x1.h>
#include "DCO_Library.h"

void main(void)
{
  volatile unsigned int I;
  int result;
  WDTCTL = WDTPW +WDTHOLD;                  // Stop Watchdog Timer
  P1DIR |= 0x12;                            // P1.1 and P1.4 outputs
  P1SEL |= 0x10;                            // P1.4 SMCLK output
  P2DIR |= 0x01;                            // P2.0 output
  P2SEL |= 0x01;                            // P2.0 ACLK output
  for( I = 0; I < 0xFFFF; I++){}            // delay for ACLK startup

  result = setDCO(TI_2MHz);

  if( result == DCO_SET_TO_SLOWEST )        // returned result if DCO registers hit min
  {
    while(1);                               // trap the CPU if hit
  }
  else if( result == DCO_SET_TO_FASTEST )   // returned result if DCO registers hit max
  {
    while(1);                               // trap the CPU if hit
  }
  else if( result == DCO_TIMEOUT_ERROR )    // result if DCO takes >10000 loops
  {
    while(1);                               // trap the CPU if hit
  }
  while(1)
  {
    P1OUT |= 0x02;                          // P1.1 = 1
    P1OUT &= ~0x02;                         // P1.1 = 0
  }
}
```

The file DCO_Library.h must be included in order to gain access to the definitions and functions when using the Library from a C program.

The appropriate device header file must also be included in the library. This requires editing the DCO_Library.s43 file. The correct device header file must be included to accommodate differences between ACLK connections in Timer_A2 and Timer_A3 devices.

This program is a demonstration of setting the DCO to 2 MHz by tuning it to a 32-kHz watch crystal.

After a delay to ensure the startup of the 32-kHz crystal, the function setDCO is called. This function takes one parameter, an integer describing the desired frequency of the DCO. In order to tune the DCO, this routine divides ACLK by eight, then increases or decreases the DCO settings until the number of DCO cycles per ACLK cycle is equal to the number passed to the setDCO routine.

Put another way, the resultant DCO frequency is:

DCO = parameter × (32768/8)

For convenience and code legibility, the included header file contains many predefined values that set the DCO to specific value. For safety, the numbers have always been rounded down, so that the frequency does not exceed that of the definition.

The setDCO routine returns a character that indicates the result of the operation. These values are given definitions in the header file for clarity. Using this return value, the routine can indicate that it has set the DCO to the fastest or slowest setting possible without matching the frequency, timed out when trying to set the DCO, or set the DCO to the desired frequency without error.

5 DCO_Library – Usage From Assembly

```
#include  <msp430x11x1.h>
EXTERN    TI_SetDCO
;-------------------------------------------------------------------------------
            RSEG    CSTACK                  ; Define stack segment
;-------------------------------------------------------------------------------
            RSEG    CODE                    ; Assemble to Flash memory
;-------------------------------------------------------------------------------
RESET       mov.w   #SFE(CSTACK),SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupPx     bis.b   #012h,&P1DIR            ; P1.1,4 output direction
            bis.b   #010h,&P1SEL            ; P1.4 = SMCLK
            bis.b   #001h,&P2DIR            ; P2.0 output direction
            bis.b   #001h,&P2SEL            ; P2.0 = ACLK
                                            ;
            mov.w   #0xFFFF, r15            ; Delay for ACLK startup
Loop        dec.w   r15
            jnz     Loop

            mov.w   #488, r12               ; value for 2MHz
            call    #TI_SetDCO
            cmp.w   #0x00, r12
            jz      Mainloop
            cmp.w   #0xFF, r12
            jz      TimeoutError
            cmp.w   #0x02, r12              ; compare returned value
            jl      SlowestError            ; 1 = DCO set to slowest
            jz      FastestError            ; 2 = DCO set to fastest setting
            jn      TimeoutError            ; 0xFF = loop timeout error
                                            ;
Mainloop    bis.b   #002h,&P1OUT            ; P1.1 = 1
            bic.b   #002h,&P1OUT            ; P1.1 = 0
            jmp     Mainloop                ; Repeat
SlowestError
            nop                             ; set breakpoint here
            jmp     SlowestError
FastestError
            nop                             ; set breakpoint here
            jmp     FastestError
TimeoutError
            nop                             ; set breakpoint here
            jmp     TimeoutError
;-------------------------------------------------------------------------------
            COMMON  INTVEC                  ; Interrupt Vectors
;-------------------------------------------------------------------------------
            ORG     RESET_VECTOR            ; POR, ext. Reset
            DW      RESET
            END
```

In assembly, the usage of the library is the same as in C. In this case, R12 is used for passing and returning parameters.

Since the header file is not included, the predefined values are not used. Instead, the numerical values can be placed in the assembly code.

6 CCE Usage

With the CCE version of the library, an extra modification is needed. Besides modifying the included header file, the correct value for DEVICE_TYPE must also be set. This is to ensure that the correct Timer_A registers are used in the code. To set the correct device, uncomment the line that sets device type to a value of 1, 2, or 3, depending on whether the device is a 1xx, 2xx Timer_A3, or 2xx Timer_A2, respectively.

7 Code Size

	Size (bytes)
DCO Library	184

8 Included Library Files

In the zip file accompanying this application report, there are two directories: source_CCE and source_IAR. The files in these directories are functionally equivalent and only contain minor changes to allow for compiling using CCE or IAR, respectively.

8.1 DCO_Library.s43

This file includes all needed functions and variables to DCO tuning.

8.2 DCO_Library.h

This file includes the definitions for the functions and variables used in DCO_Library.s43. This file must be included in any C program that uses the library. It also includes a series of precomputed parameters for the setDCO function.

9 Function Description

9.1 setDCO(int delta)

This function causes the DCO to be set to a specific frequency. This frequency is based on ACLK, which is typically a 32-kHz crystal. From a 32-kHz crystal, the DCO speed becomes:

DCO = delta × (32768/8)

It should be noted that passing a delta of 1 or 0 results in incorrect behavior.

This function returns a status integer. There are several values used to describe the state of the DCO after calling this function. These values are given easy-to-remember definitions in the header file:

- 0 == DCO_NO_ERROR : The DCO was set to the desired frequency.
- 1 == DCO_SET_TO_SLOWEST: The DCO registers were decremented to the slowest setting without reaching the desired frequency.
- 2 == DCO_SET_TO_FASTEST: The DCO registers were incremented to the fastest setting without reaching the desired frequency. On 2xx devices, this also means that the setDCO function incremented the DCO until its settings were equal to the built-in 16-MHz calibrated values.
- 0xFF == DCO_TIMEOUT_ERROR: The DCO did not reach the desired frequency after conducting 10,000 measurement loops.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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