

**Compensating for oscillator frequency variations in UCD3138**

**Application Note**

|  |  |  |
| --- | --- | --- |
| Version | Date | Comment |
| 0.1 | 6/28/2013 | First version |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Revision History

# Contents

## Table of Contents

[1 Revision History 2](#_Toc360191817)

[2 Introduction: 3](#_Toc360191818)

[This application notes describes one way to compensate for the variations in the frequency of the main oscillator clock in UCD3138. 3](#_Toc360191819)

[3 External crystal circuit 3](#_Toc360191820)

[4 Requirement details 5](#_Toc360191821)

[5 Frequency measurement using timer capture 5](#_Toc360191822)

[6 DPWM timing compensation 6](#_Toc360191823)

[7 Timer interrupt frequency compensation 7](#_Toc360191824)

[8 UART baud-rate compensation 7](#_Toc360191825)

[9 IMPORTANT NOTICE 8](#_Toc360191826)

[10 Contents 9](#_Toc360191827)

[10.1 Table of Contents 9](#_Toc360191828)

[10.2 Figures 9](#_Toc360191829)

[11 Documentation and References 10](#_Toc360191830)

[11.1 References 10](#_Toc360191831)

## Figures

# Introduction:

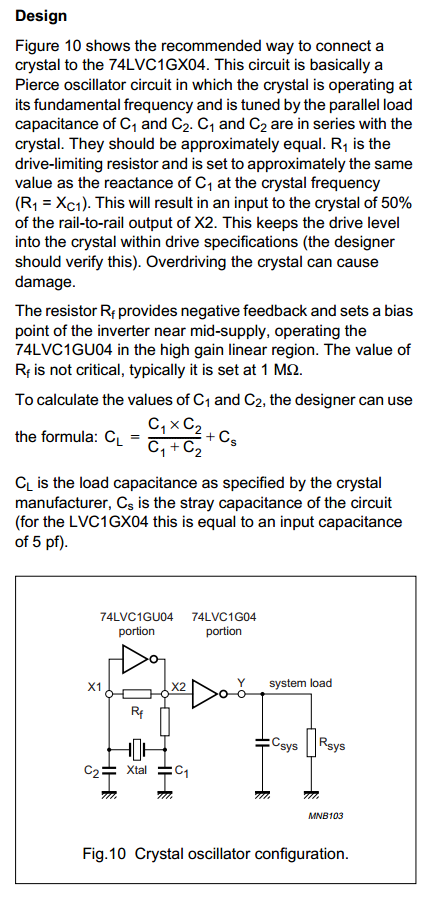
# This application notes describes one way to compensate for the variations in the frequency of the main oscillator clock in UCD3138.

To achieve this goal, we will use an external low cost 32.768 KHZ watch crystal and a crystal driver (or inverter gate). By connecting the stable 32KHZ external clock generated by this circuit to the timer capture (TCAP) pin in UCD3138, we will be able to adjust several internal controls to compensate for internal clock variations.

# External crystal circuit

To drive the Crystal oscillator a SN74LVC1GX04 crystal driver by Texas Instruments is used. The Crystal used is a 9H03200031 by TXC corporation taiwan.

The external components used are C1= C2 = 27 pF (NPO or COG type capacitors), Rf= 4.5 MΩ (metal film 1%), Rs = 182 K Ω (metal film 1%).



# Requirement details

Oscilator frequency compensation is required to provide three advantages.

The following three parameters should stay intact regardless of the internal oscillator frequency variations

* The DPWM waveforms frequency and other DPWM related timings (dead times)
* The frequency of system TICK (16-bit timer interrupt)
* The frequency (physical bud-rate) of UART communication

Please note, the default setting in UCD3138 of:

MiscAnalogRegs.CLKTRIM.bit.HFO\_LN\_FILTER\_EN = 1;

Leads to larger drift in the oscillator frequency as a function of operating temperature. This is actually fine for the purpose of proving that the compensation routines are functional in this application note.

But after testing the code and proving that even larger amount of drift can be compensated, it is recommended to change the configuration to:

MiscAnalogRegs.CLKTRIM.bit.HFO\_LN\_FILTER\_EN = 0;

# Frequency measurement using timer capture

We would like to acquire a value that is proportional to the frequency of internal oscillator.

To do so, we will read the capture data register of the 24bit-timer inside the standard interrupt.

Since the standard interrupt is invoked roughly every 100µs, this should provide us with a measurement over three (or four) periods of the external crystal oscillator.

1/ 32.768 KHZ = 30.5176 µs

100 µs / 30.5176 µs ~ 3 times

The 24-bit timer is initialized to capture on rising edge:

TimerRegs.T24CAPCTRL.bit.EDGE = 1;

Then the following routine is added to the standard timer interrupt:

if( TimerRegs.T24CAPCTRL.bit.CAP\_INT\_FLAG)

{

t24\_latched = TimerRegs.T24CAPDAT.bit.CAP\_DAT;

if(t24\_latched > t24\_latched\_previous)

{

t24\_latched\_diff = t24\_latched - t24\_latched\_previous;

}

else

{

t24\_latched\_diff = t24\_latched + 0xFFFFFF - t24\_latched\_previous;

}

if( t24\_latched\_diff > 1660) // if counted four times instead of three

{

t24\_latched\_diff = (t24\_latched\_diff \* 3) >> 2;

}

t24\_latched\_previous = t24\_latched;  
}

The variable “t24\_latched\_diff” is containing our measurements result and its value is usually around 1430 in room temperature.

The 1430 value is the result of measuring the internal 15.625 MHZ internal frequency that the all UCD3138 timers are driven by for three periods of the external 32.768KHZ clock ( 3 X 30.5176 µs = 91.553 µs).

91.553 µs/ (1/15.625 MHZ) = 91.553 µs/ ( 64 ns) = 1430.52

The variable “t24\_latched\_diff” therefore, provides a resolution of better than 0.1% and is directly proportional to the internal oscillator frequency of UCD3138.

# DPWM timing compensation

The variable “t24\_latched\_diff” is used as the input into the DPWM timing compensation routine.

The DPWM timing compensation routine is toplogy specific. Please check the availability of this code for your specific power stage and configuration of interest.

The original code is developed for phase shifted full bridge – peak current mode EVM. The routine resides inside the

void deadtime\_adjust(void) function in the file deadtime\_adjust.c

The entire code is quite long, but following is a part of the code that represets the concept well.

#define PERIOD\_SCALER 2798 // (2798 >> 4 ) == 174.88

#define PERIOD\_SHIFTER 4 // (2798 >> 4 ) == 174.88

period\_dummy = (((PERIOD\_SCALER \* t24\_latched\_diff) >> PERIOD\_SHIFTER) / pmbus\_dcdc\_config[0].switching\_frequency);

period\_new = ((period\_new \* 7) >> 3) + (period\_dummy >> 3);

if(period\_new > period)

{

period = period + 1;

}

else if(period\_new < period)

{

period = period - 1;

}

half\_period = period >>1;

pwm\_slavesync = half\_period \*16; //scale to 250ps/bit in registers

pwm\_period = period \*16;

pwm0\_ev1 = bridge\_delay \* 4; // scale to 250ps/bit (bridge\_delay/4 \*16)

pwm0\_ev2 = half\_period \* 16; //scale to 250ps/bit (\*16)

pwm0\_ev3 = half\_period \* 16 + (bridge\_delay - 30 + sr\_rising\_bridge\_delay) \* 4;

pwm0\_ev4 = (bridge\_delay + sr\_falling\_bridge\_delay) \* 4;

pwm0\_ev5 = bridge\_delay \* 4; //scale to 250ps/bit (bridge\_delay/4 \*16)

pwm0\_ev6 = half\_period \* 16;

pcm\_blank = PCM\_BLANK;

Dpwm0Regs.DPWMEV1.all = pwm0\_ev1;

Dpwm0Regs.DPWMEV2.all = pwm0\_ev2; // This has to match 3A EV2 for curr limit to match (readjust at end)

Dpwm0Regs.DPWMEV3.all = pwm0\_ev3;

Dpwm0Regs.DPWMEV4.all = pwm0\_ev4;

Dpwm0Regs.DPWMBLKABEG.all = 0;

Dpwm0Regs.DPWMBLKAEND.all = ( bridge\_delay + pcm\_blank) \* 4;

Dpwm0Regs.DPWMBLKBBEG.all = pwm0\_ev5 + deadtime\_adi\_lag ;

Dpwm0Regs.DPWMBLKBEND.all = pwm0\_ev6;

Dpwm0Regs.DPWMSAMPTRIG2.all = pwm\_slavesync;

Dpwm0Regs.DPWMPHASETRIG.all = pwm\_slavesync;

Dpwm0Regs.DPWMPRD.all = pwm\_period;

------------------------------------------------------------------------------------------------------------------------------------------------

The concept is quite simple. The variable “period\_new” is calculated based on the fresh measurement of “t24\_latched\_diff” from timer capture. If the newly calculated period is bigger than the current setting of period, then period is incremented by a single count.

Then all the other events, blanking intervals, sample trigger positions and other timings are re-calculated as a function of the newly incremented period. The same is through if the newly calculated period is smaller than the current setting of period. The only difference is that in this case the period will be decremented by one count.

# Timer interrupt frequency compensation

Since the variations in the oscillator clock frequency are limited to +/-5% a linear compensation function should be sufficient .

The following C statement called in the main loop will keep the interrupt frequency fixed at 10KHZ:

TimerRegs.T16PWM0CMP0DAT.all = (t24\_latched\_diff \* INT\_FREQ\_*SCALER*) >> INT\_FREQ\_SHIFTER;

When for 10KHZ:

#define INT\_FREQ\_*SCALER* 1136 // (1136 >> 10) == 1.1094

#define INT\_FREQ\_SHIFTER 10 // (1136 >> 10) == 1.1094

1430 \* 1.1094 = 1586; 1586 \* 64 nS ~ 100 µS

If the interrupt frequency is different than 10KHZ, the INT\_FREQ\_*SCALER and* INT\_FREQ\_SHIFTER will need to be changed accordingly.

# UART baud-rate compensation

Since the variations in the oscillator clock frequency are limited to +/-5% a linear compensation function should be sufficient .

The value of baud setting in Uart registers can be calculated by:

Baud rate register setting = ( F\_iclk / (8 \* Baudrate)) - 1

For the baud rate of 4800 (Typical for EVMs), The UARTBAUD is calculated as:

UARTBAUD = (15.625 MHZ /( 8 \* 4800 ))- 1 = 407

The following C statement called in the standard interrupt will keep the baud ratefixed at 4800 BPS:

Uart1Regs.UARTLBAUD.all = (BAUD\_*SCALER* \* t24\_latched\_diff) >> BAUD\_SHIFTER ;

When for baud rate of 4800 BPS:

#define BAUD\_*SCALER* 583 // (583 >> 11) == 0.2847

#define BAUD\_SHIFTER 11 // (583 >> 11) == 0.2847

1430 \* 0.2847 = 407; 407 -> 4800 BPS according to UCD3138 baud rate reference table

If the UART baud rate is different than 4800, the BAUD\_*SCALER and* BAUD\_SHIFTER will need to be changed accordingly.

# IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI’s standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI’s publication of information regarding any third party’s products or services does not constitute TI’s approval, license, warranty or endorsement thereof.

Reproduction of 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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI’s products or services with *statements different from or beyond the parameters* stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.

[www.ti.com/sc/docs/stdterms.htm](file:///C:\projects\hpa\TechnicalInfo\Cyclone\GUI%20Userguides\www.ti.com\sc\docs\stdterms.htm)

Mailing Address:

Texas Instruments

Post Office Box 655303

Dallas, Texas 75265

Copyright ©2002, Texas Instruments Incorporated

# 

# Documentation and References

## References

# SN74LVC1GX04 Crystal Oscillator Driver data sheet

<http://www.ti.com/product/sn74lvc1gx04>