

# Spread-Spectrum Clock Source Using an MSP430

Mark Heminger

MSP430 - Signal Chain Applications Team

## ABSTRACT

While spread-spectrum clocking has long since been used in processor and memory clock trees, there are many other clocked systems, such as power supplies or switch-mode amplifiers, that continue to use a single-frequency clock. This can, in turn, generate significant EMI and can make meeting governmental regulations for EMI challenging. These regulations typically set a limit on peak energy within a narrow frequency range. Spread-spectrum techniques vary the period of the clock to distribute the energy over several frequencies, thus reducing the amplitude of the central peak.

This document describes a method of generating a spread-spectrum clock source using a low-cost low-power MSP430 microcontroller from Texas Instruments.

This application report includes source code that can be downloaded from this link: <http://www-s.ti.com/sc/psheets/sl原因291/sl原因291.zip>.

## Contents

1	System Overview .....	2
2	Configuring the Spread-Spectrum Clock Source .....	3
3	Performance Measurements .....	6
4	Processor Selection.....	11
5	Firmware Description .....	12
6	Acknowledgements .....	13
Appendix A	Glossary .....	14
Appendix B	C Source Code .....	15

## List of Figures

1	Pin Assignment of MSP430Fx11x1A (Top View).....	2
2	Schematic for Run-Time Configuration .....	3
3	Output Spectrum in Random Hopping Mode .....	6
4	Output Spectrum For Four Different Spreading Settings in Random Hopping Mode .....	7
5	Output Spectrum For Four Different Spreading Settings in Linear Sweep Mode .....	8
6	Start-Up Frequency Transient .....	9
7	Response to Power-Supply Voltage Change.....	10
8	MSP-FET430P120 Flash Emulation Tool.....	12

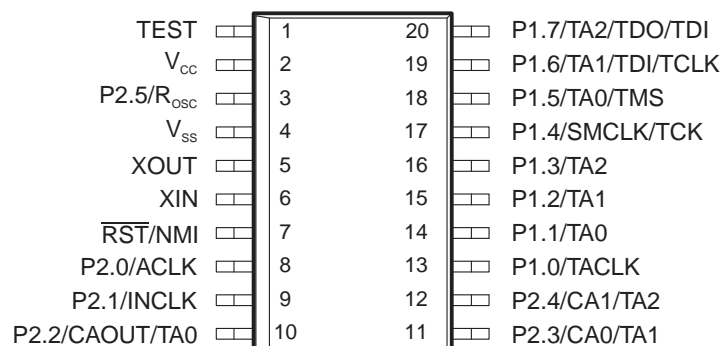
## List of Tables

1	Recommended FOD Settings .....	4
2	Low-Cost MSP430 Processor Family .....	11
3	Firmware File Descriptions.....	12
4	Firmware Function Descriptions.....	13

## 1 System Overview

The Texas Instruments MSP430 family of microcontrollers use digitally controlled oscillators (DCOs) to generate their internal clock signals. By periodically adjusting the DCO settings, the frequency of the master clock can be varied, thus spreading its average spectral energy over a range of frequencies and reducing the amplitude of the spectral peaks.

By varying the DCO and using the resulting clock to drive external systems such as power supplies or power amplifiers, it is possible to extend the benefits of average spectral energy spreading to these systems. Such power systems were the original target for this project, but they are certainly not its only application.



**Figure 1. Pin Assignment of MSP430Fx11x1A (Top View)**

### 1.1 Crystal Input

Frequency accuracy for the DCO is provided by reference to a low-cost 32.768-kHz watch crystal.

### 1.2 Clock Outputs

The dithered clock output comes out from the SMCLK pin of the MSP430. It is generated by an internal DCO. The center frequency of the DCO is an integer multiple of the crystal frequency, 32 768 Hz. The DCO operates at discrete frequencies, but the DCO frequencies are independent of the crystal frequency and are spaced much closer than integer multiples of the crystal frequency.

Frequency spreading is accomplished by offsetting the output from the center frequency. A new frequency offset is chosen as often as 1.95 ms (every 64 cycles of the 32 768-Hz clock). The frequency update rate may be slowed down to prevent interactions in the audible range. A control algorithm gradually adjusts the DCO frequency offsets to keep the average output spectrum centered near the center frequency.

The ACLK pin provides an unmodulated clock at 32 768 Hz (unless modified by the FOD setting described in [Section 2.4](#)).

### 1.3 Modulator Output

Each period of the DCO clock can occur at only a limited range of values. To improve the resolution and reduce EMI, the DCO jumps between two adjacent frequencies as often as once per period, over a cycle no longer than 32 periods. Therefore, even when the offset frequency is not being varied, the DCO modulator still provides high-frequency spreading.

## 2 Configuring the Spread-Spectrum Clock Source

### 2.1 Compile-Time vs Run-Time Configuration

The operational settings for the clock generator can be chosen at either compile time or at run time. To choose run-time configuration, the compiler flag `READ_JUMPERS` must be `#defined` in the `settings.h` file. In that case, the settings are determined during the MCU power-up sequence by reading jumpers on the GPIO pins and by measuring the resistance ratio of two precision resistors.

If the `READ_JUMPERS` flag is not defined, the operational settings are defined at compile-time using constants defined in the `settings.h` file.

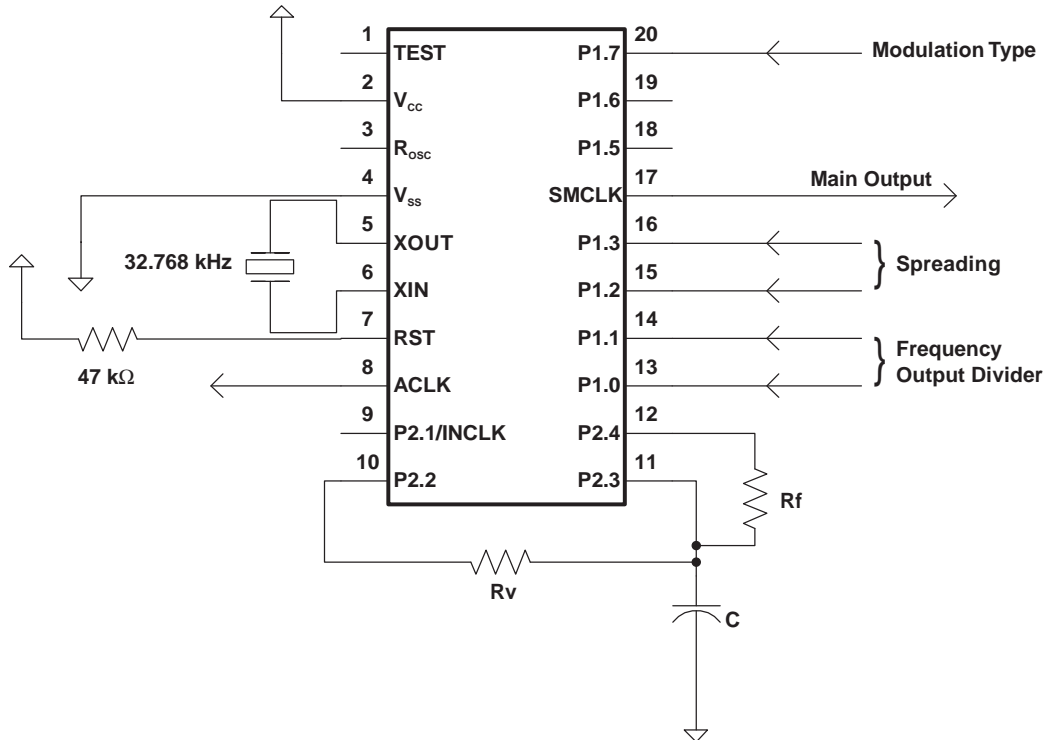


Figure 2. Schematic for Run-Time Configuration

### 2.2 Center Frequency Programming

The center frequency of the DCO is an integer multiple of the crystal frequency (normally 32.768 kHz). It is possible to get finer resolution on the center frequency by using the frequency output divider (FOD), described in [Section 2.4](#).

**Run-Time Configuration:** The center frequency is programmed at power up using one low-cost capacitor and two precision resistors. The ratio of the resistors is used to calculate an integer center frequency multiplier (CFM) according to the formula:

$$\text{CFM} = \text{ROUND}(\text{MAX\_FREQ\_MULT} \times R_v/R_f)$$

$$\text{Center frequency} = \text{CFM} \times 32768 \text{ Hz/FOD}$$

The default value for `MAX_FREQ_MULT` is 128, allowing a maximum frequency of approximately 4.2 MHz. The component values should be chosen so that  $R_f \times C$  is approximately 1 ms. Resistor  $R_v$  generally should be chosen to be smaller than  $R_f$ . It is possible to push the frequency limit slightly upward if the power-supply voltage is sufficiently high. Typical values are:

$$C = 0.1 \mu\text{F} + 80\%/-20\%$$

$$R_f = 10 \text{ k}\Omega \pm 0.1\%$$

$$R_v = 5 \text{ k}\Omega \text{ to } 10 \text{ k}\Omega \pm 0.1\%$$

## Configuring the Spread-Spectrum Clock Source

Resistor Rf is connected between pins P2.4 and P2.3. Resistor Rv is connected between pins P2.2 and P2.3. The capacitor C is connected between P2.3 and ground.

**Compile-Time Configuration:** The center frequency multiplier is calculated during the initialization sequence using the constant MASTER\_FREQ, declared in the settings.h file, as an input term.

### 2.3 Frequency Spreading Range

The range of the frequency spreading offset away from the center frequency of the DCO is an integer multiple of the crystal frequency (normally 32.768 kHz). It is possible to get finer resolution on the frequency spreading range by using the FOD, which is described in [Section 2.4](#).

**Run-Time Configuration:** The amount of frequency spreading is set using two digital input pins: P1.2 and P1.3. The amount of frequency spreading is determined using the formula:

$$F_{hi}/F_{lo} = SDCO^{(FSS/32)}$$

Where:

FSS = Frequency spreading steps

For the MSP430x11x1 family of processors, the value of SDCO ranges from 1.07 to 1.16, with a rated typical value of 1.12.

JUMPER SETTINGS		FREQUENCY SPREADING STEPS (FSS)	OUTPUT SPREADING		
P1.3	P1.2		MIN	TYP	MAX
0	0	1	0.2%	0.3%	0.5%
0	1	8	1.7%	2.9%	3.8%
1	0	16	3.4%	5.8%	8.8%
1	1	32	7.0%	12.0%	16.0%

**Compile-Time Configuration:** The frequency spreading range is initialized using the constant FSS declared in the settings.h file.

### 2.4 Frequency Output Divider

The frequency outputs can be divided down to generate low frequencies with finer resolution on the center frequency and frequency spreading range. By default, the center frequency and spreading steps are integer multiples of 32 768 Hz. By using the Frequency Output Divider (FOD) it is possible to make the resolution as fine as 4096 Hz but with a reduced maximum frequency.

$$\text{Center frequency} = (32\,768 \text{ Hz}/\text{FOD}) \times \text{CFM}$$

$$\text{Spreading step size} = (32\,768 \text{ Hz}/\text{FOD})$$

$$\text{Spreading range} = (32\,768 \text{ Hz}/\text{FOD}) \times \text{FSS}$$

The master clock, MCLK, that controls the execution speed of the processor core is not affected by FOD settings; only the output signals on SMCLK and ACLK are slowed down. This ensures that the processor still operates quickly enough to perform its operations when operating at low output frequencies.

[Table 1](#) shows the recommended FOD setting based on the desired center frequency. In all the cases shown, the master frequency would be set in the 2 MHz to 4.2 MHz range.

**Table 1. Recommended FOD Settings**

CENTER FREQUENCY	RECOMMENDED FOD SETTING
>2 MHz	÷1
1 MHz to 2 MHz	÷2
0.5 MHz to 1 MHz	÷4
<500 kHz	÷8

The FOD also affects the rate at which new setpoint frequencies are chosen. Different combinations of CFM and FOD can be used to move any undesired low-frequency modulation effects.

**Run-Time Configuration:** The frequency output divider is set using two digital input pins, P1.0 and P1.1.

JUMPER SETTINGS		FREQUENCY OUTPUT DIVIDER
P1.1	P1.0	
0	0	÷1
0	1	÷2
1	0	÷4
1	1	÷8

**Compile-Time Configuration:** The frequency spreading range is initialized using the constant OUT\_DIV declared in the settings.h file.

OUT_DIV	FREQUENCY OUTPUT DIVIDER
0	÷1
1	÷2
2	÷4
3	÷8

## 2.5 Frequency Update Interval

The time between changes of the frequency setpoint is

$$\text{STEP\_INTERVAL} \times \text{WDTIS} \times \text{FOD} / \text{CRYSTAL\_FREQUENCY}$$

Where:

CRYSTAL\_FREQUENCY is 32.768 kHz

WDTIS is the watchdog timer interval select (hardcoded to 64)

STEP\_INTERVAL is a constant, set to 3

FOD is the frequency output divider, set by two jumpers. Possible values are 1, 2, 4, 8.

According to this formula, the frequency setpoint update rate is 170.66 Hz/FOD.

The repeat rate of the generated output frequency pattern is two times the frequency spreading setting, FSS, when running in linear frequency sweep mode.

**Compile-Time Configuration:** The frequency update interval is set using the constant STEP\_INTERVAL declared in the settings.h file.

**Run-Time Configuration:** There is no option to change the frequency update interval at run time.

## 2.6 Modulation Mode

The MSP430 can be configured to modulate the frequency spreading using either a linear sweep mode or a random hopping mode. In random hopping mode, the frequency offset hops to discrete frequencies in a pseudo-random order. To ensure frequency consistency over both a medium-term and long-term time frame, every sequence of up to 32 frequency offset values generates the same distribution. However, the order in which the offsets occur is different for each sequence. The pseudo-random order of the offsets repeats itself every 255 times through the sequence.

**Run-Time Configuration:** Input pin P1.7 is used to select between linear sweep mode and random hopping mode.

JUMPER SETTING P1.7	MODULATION TYPE
0	Random hopping
1	Linear sweep

**Compile-Time Configuration:** The modulation mode is set using the constant MOD\_TYPE declared in the settings.h file. Allowable values are MOD\_HOP and MOD\_SWEEP.

### 3 Performance Measurements

This section contains measurements taken using an MSP430F1111A processor with a target center frequency of 3.014656 MHz.

#### 3.1 Spectral Distribution

CFM = 92

FOD (0) = ÷1

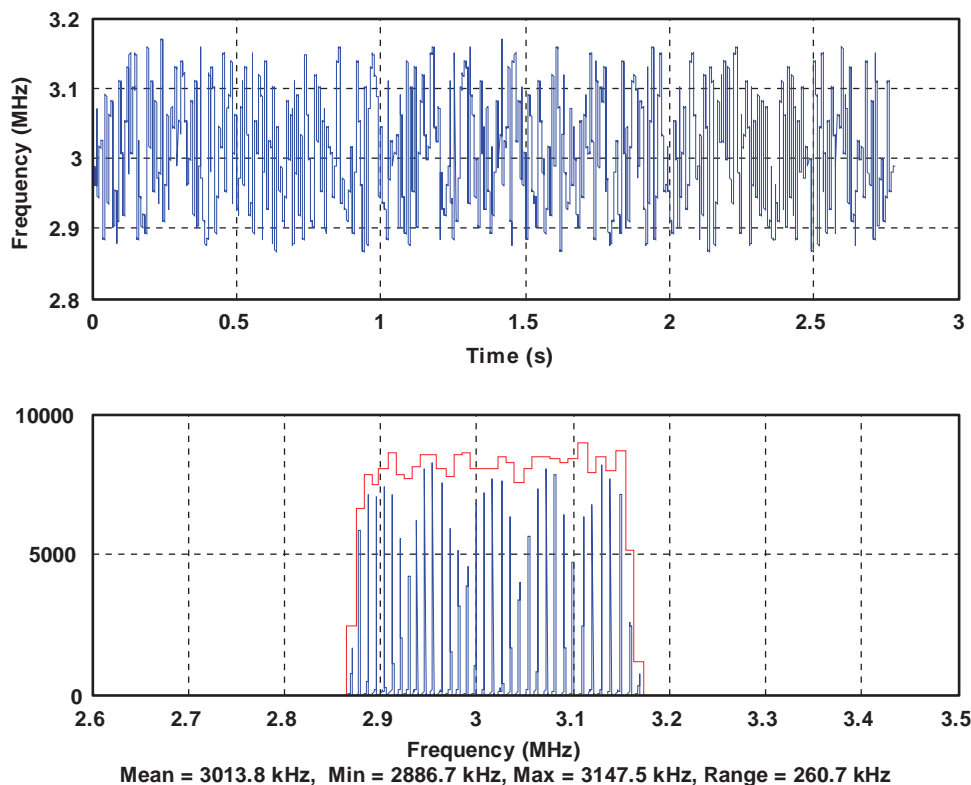
Center frequency = 3.014656 MHz

FSS (3) = 32 steps

Mode = Random hopping

Figure 3 shows the output frequency and spectrum while operating in random hopping mode with 32 steps of spreading. The spectrum was measured using a time-interval analyzer with a time resolution of about 0.1 ns. The DCO operates at discrete intervals with a granularity of about 1.1 ns. The narrow blue peaks in Figure 3, Figure 4, and Figure 5 show the spectral energy in each of the timer interval analyzer's narrow resolution frequency bins, while the red bars show the total energy in each of the DCO output frequency bins.

Figure 4 shows random hopping operation with several different spreading settings. Figure 5 shows operation with linear sweeping modulation using several different spreading settings.



**Figure 3. Output Spectrum in Random Hopping Mode**

### 3.1.1 Random Hopping Spectrum

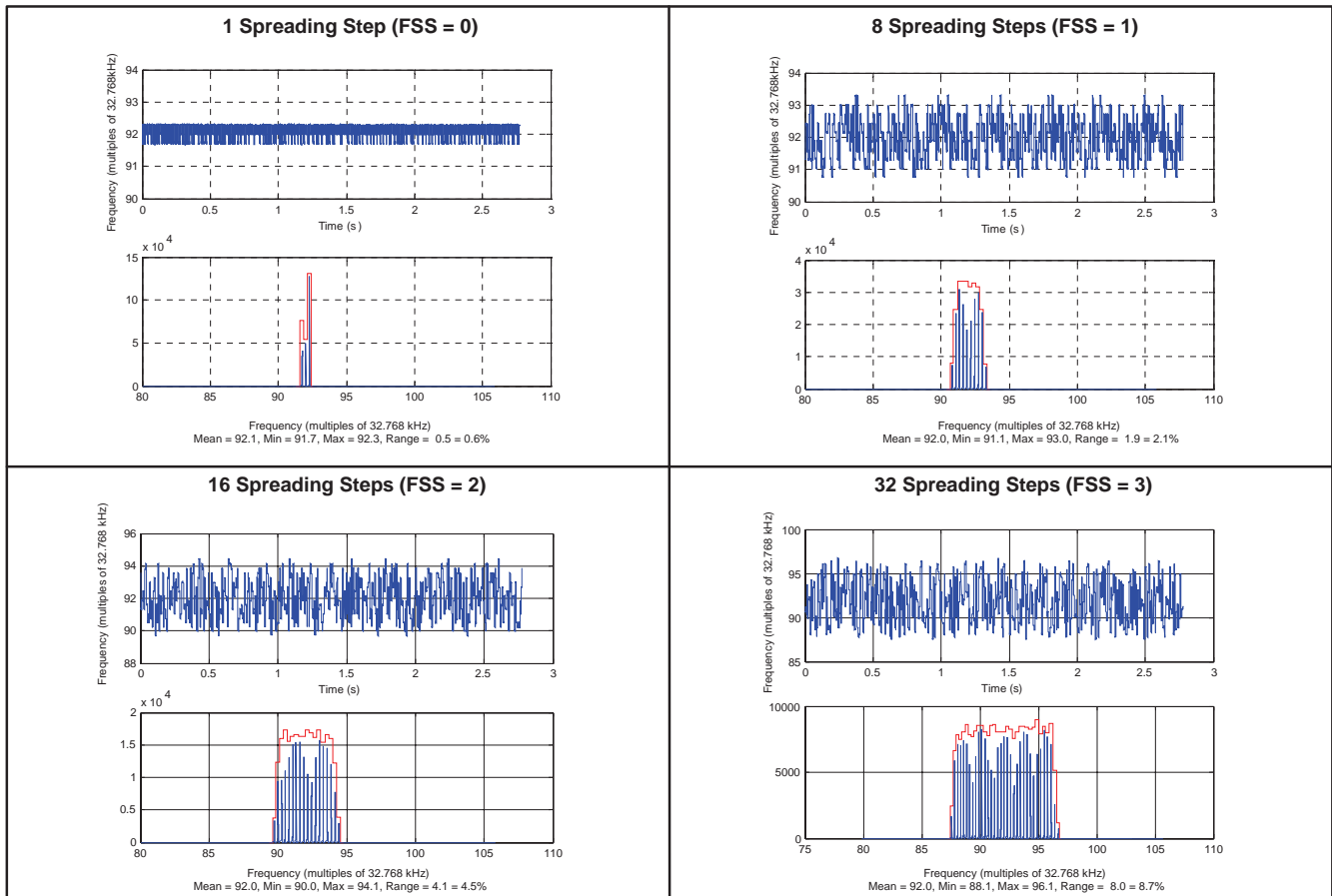


Figure 4. Output Spectrum For Four Different Spreading Settings in Random Hopping Mode

### 3.1.2 Swept Frequency Spectrum

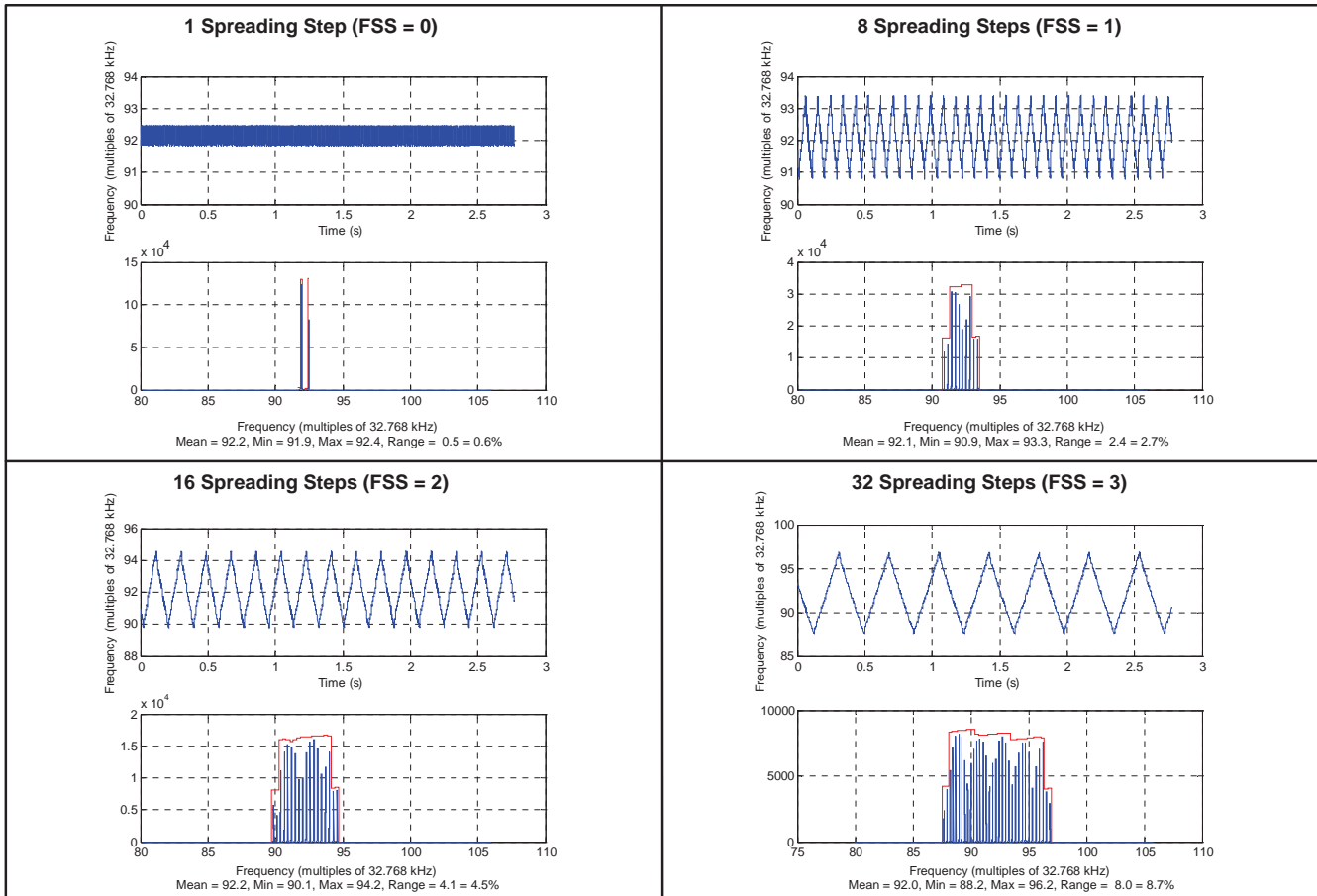
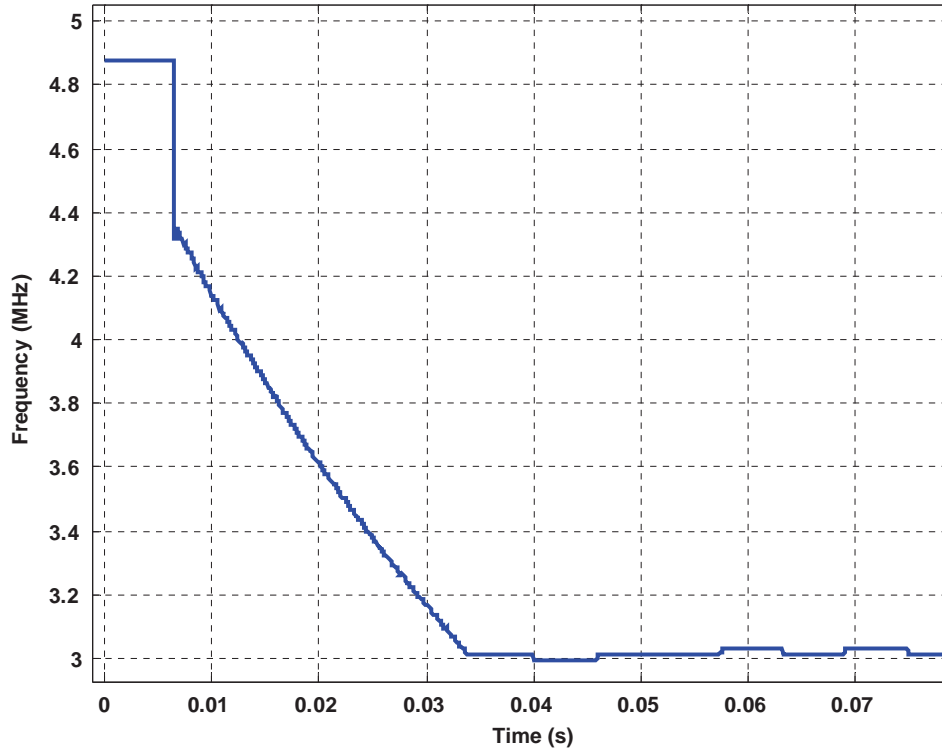


Figure 5. Output Spectrum For Four Different Spreading Settings in Linear Sweep Mode

### 3.2 Start Up

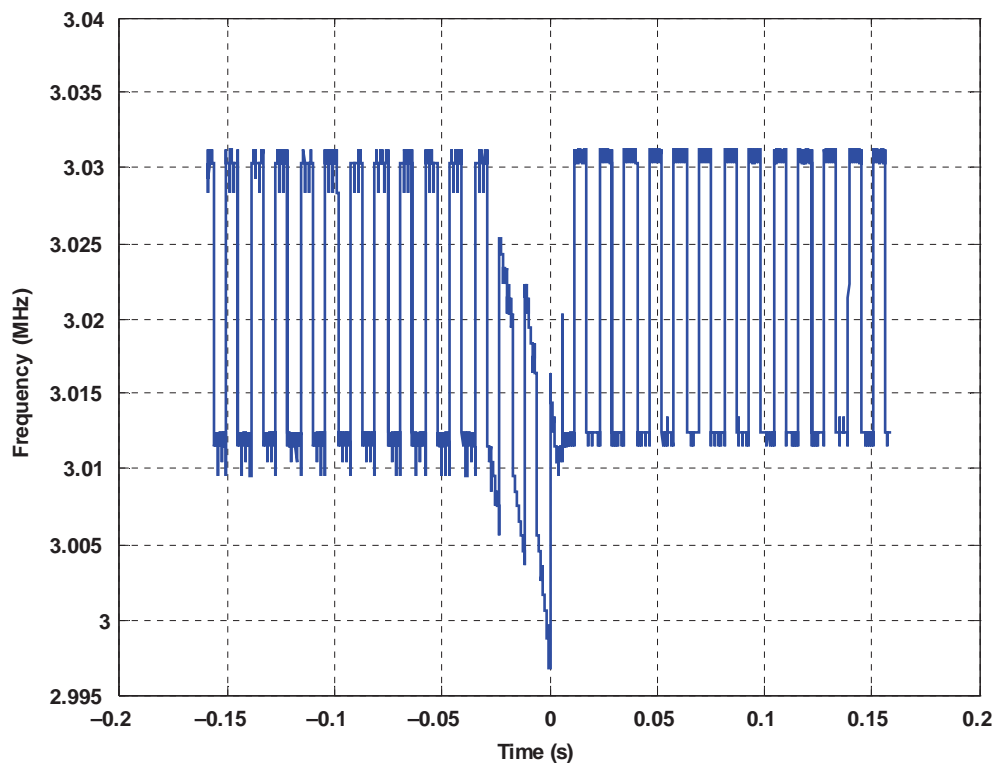
During the start-up sequence, the clock is set briefly to its maximum speed, then adjusted until it reaches the desired center frequency. [Figure 6](#) shows this sequence. Note that the amount of time required to reach the desired center frequency depends on the desired frequency—a lower frequency requires a longer start-up time. Setting a lower value for the constant `INIT_MOD_VALUE` in the `settings.h` file can reduce the start-up time when using lower frequencies.



**Figure 6. Start-Up Frequency Transient**

### 3.3 Voltage and Temperature Disturbance Rejection

The frequency of the DCO is sensitive to power-supply voltage, and varies by 0% to 10% per volt. The control algorithm in the WatchDogISR routine adjusts the DCO settings to keep the average frequency at the desired center frequency. Figure 7 shows the response when the power-supply voltage drops from 3.5 V to 3 V. After the initial transient, while the voltage is actively falling, the control algorithm quickly stabilizes the frequency back to its target value.



**Figure 7. Response to Power-Supply Voltage Change**

The frequency of the DCO is also sensitive to temperature, varying by about 0.38% per degree Celsius change in temperature. The control algorithm should be able to easily compensate for this change.

## 4 Processor Selection

The lowest-cost members of the MSP430 family of processors are shown in [Table 2](#). Processors with Flash memory are good for code development and low-volume production. For large-volume production, costs may be reduced by switching to a ROM-based processor.

**Table 2. Low-Cost MSP430 Processor Family**

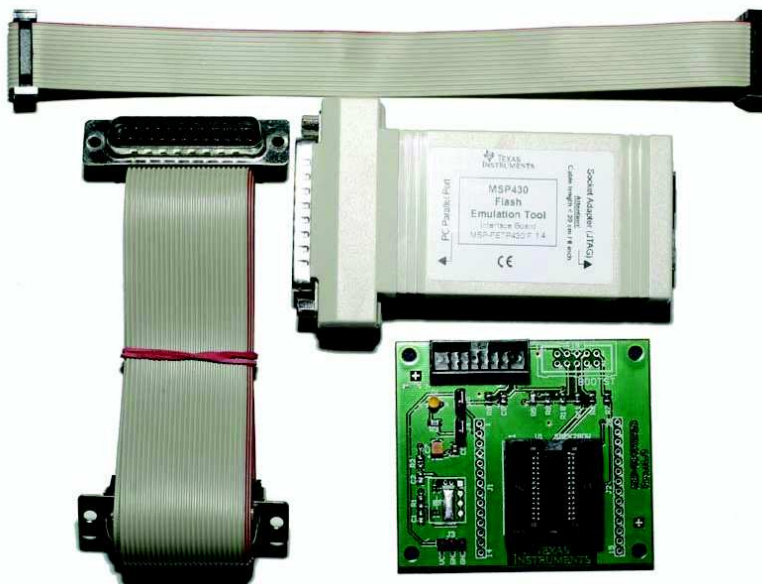
PART NUMBER	TYPE OF PROGRAM MEMORY	PROGRAM MEMORY (kB)	RAM (Bytes)	PIN/ PACKAGES	APPROX. 1KU PRICE (US\$)	DESCRIPTION
MSP430C1101	ROM	1 kB	128	20SOIC 20TSSOP 24QFN	0.60	16-Bit Ultra-Low-Power Microcontroller, 1-kB ROM, 128-B RAM, Comparator
MSP430F1101A	Flash	1 kB	128	20SOIC 20TSSOP 20TVSOP 24QFN	0.99	16-bit Ultra-Low-Power Microcontroller, 1-kB Flash, 128-B RAM, Comparator
MSP430C1111	ROM	2 kB	128	20SOIC 20TSSOP 24QFN	1.10	16-Bit Ultra-Low-Power Microcontroller, 2-kB ROM, 128-B RAM, Comparator
MSP430F1111A	Flash	2 kB	128	20SOIC 20TSSOP 20TVSOP 24QFN	1.35	16-bit Ultra-Low-Power Microcontroller, 2-kB Flash, 128-B RAM, Comparator
MSP430F2001	Flash	1 kB	128	14TSSOP 14PDIP 16QFN	0.55	16-bit Ultra-Low-Power Microcontroller, 1-kB Flash, 128-B RAM, Comparator
MSP430F2011	Flash	2 kB	128	14TSSOP 14PDIP 16QFN	0.70	16-bit Ultra-Low-Power Microcontroller, 2-kB Flash, 128-B RAM, Comparator

The compiled size of the firmware determines which processor must be used. The '1101 processors each contain 1 kB of program memory, while the '1111 parts contain 2 kB. When the firmware is configured to use compile-time configuration, the code fits snugly in the 1-kB available memory. Run-time configuration of the center frequency adds roughly 250 bytes of code, forcing a move into a processor with 2 kB of memory.

The software in this application report was developed using the 'F1111A processor. Since that time, TI has released the 'F20x1 members of the MSP430 family of processors. These new processors have lower cost and additional features over the 'F11xx processors and are available in smaller packages. Note that these smaller packages do not have the GPIO pins P2.2 through P2.4, so only the compile-time version of the software works with them. The `READ_JUMPERS` compiler flag must be undefined in the software.

## 5 Firmware Description

This source code was developed in C language running on an MSP-FET430P120 Flash Emulation Tool using the CrossWorks development environment from Rowley Associates (<http://www.rowley.co.uk/msp430/index.htm>). The code should work with other compilers but may require some small changes to the interrupt declarations.



**Figure 8. MSP-FET430P120 Flash Emulation Tool**

The source code consists of three files: `hopper.c`, `settings.h`, and `main.h`. [Table 3](#) describes the content of each file. [Table 4](#) describes each of the functions in `hopper.c`, including their calling relationships.

**Table 3. Firmware File Descriptions**

FILE NAME	CONTENTS
<code>hopper.c</code>	All the executable code
<code>settings.h</code>	Settings for the executable
<code>main.h</code>	Commonly used constants, macros, and type definitions

**Table 4. Firmware Function Descriptions**

FUNCTION NAME	DESCRIPTION	CALLS	CALLED BY
Main	Configure everything, then drop into low-power idle loop and allow periodic interrupts to call WatchDogISR.	ConfigureEverything, WatchDogISR (indirectly)	
WatchDogISR	Watchdog Timer interrupt-service routine. Periodically choose a new frequency for the main clock, then adjust the DCO register to move toward that new frequency.	CaptureElapsedTime, Shuffle, ChangeDCOControl, JumpDCOControl	Triggered by watchdog timer in interval timer mode, generally while main() is waiting in low-power idle mode
ChangeDCOControl	Change the DCO register by up to one count. Handle rollovers properly.	JumpDCOControl	ConfigureClock, WatchDogISR
JumpDCOControl	Change the DCO register by multiple counts. Handle rollovers properly.		ChangeDCOControl, WatchDogISR
CaptureElapsedTime	Captures a timestamp from Timer_A and calculates the elapsed time since the previous capture		ConfigureClock, WatchDogISR
ConfigureEverything	Configure hardware registers and initialize variables	ConfigureClock, __delay_cycles, ConfigureIO, MeasureResistorRatio, ConfigureTimerA, ConfigureSpreading,	Main
ConfigureClock	Configure the main clock oscillator	CaptureElapsedTime, ChangeDCOControl	ConfigureEverything
ConfigureIO	Configure the I/O pins as either dedicated function or general-purpose I/O		ConfigureEverything
ConfigureTimerA	Configure Timer_A to count SMCLK ticks		ConfigureEverything
ConfigureWatchDog	Configure the WatchDog Timer to run in interval timer mode		ConfigureEverything
MeasureResistorRatio	Read the ratio of the resistors used to set the clock frequency	MeasureChargeTime	ConfigureEverything
MeasureChargeTime	Pulses GPIO lines to discharge and charge an external R/C network. Uses Timer_A capture unit to measure charge time.	__delay_cycles	MeasureResistorRatio
ConfigureSpreading	Select the frequency-spreading modulation scheme based on the GPI jumper settings		ConfigureEverything
Shuffle	Rearrange the order of the entries in the frequency-spreading table in pseudo-random order		ConfigureSpreading, WatchDogISR

## 6 Acknowledgements

The author would like to thank Patrick Rowland, who proposed using an MSP430 to generate a spread-spectrum clock source, provided a starter code set, and provided invaluable direction during the course of the project. Mark Buccini provided the starter code for the clock-calibration routine.

**Appendix A Glossary**

ACLK	Auxiliary clock
CFM	Center frequency multiplier
DCO	Digital controlled oscillator
EMI	Electromagnetic interference
FOD	Frequency output divider
FSS	Frequency spreading steps
GPIO	General purpose input/output
MCLK	Master clock
MCU	Microcontroller unit
MSP430	TI low-cost low-power microcontroller brand
SDCO	Step size of DCO
SMCLK	Sub-system master clock

## Appendix B C Source Code

The C source code discussed in this application report is available for download from this link: <http://www-s.ti.com/sc/psheets/slaa291/slaa291.zip>. The following files are included:

- **settings.h**

This file contains the configuration settings used by hopper.c.

- **main.h**

This file contains commonly used constants, macros, and type definitions.

In the source listing, the prefix of the variable name indicates the format of the variable, as described in this table:

TYPE	PREFIX	DESCRIPTION
u8	uc	unsigned char
u16	u	unsigned 16-bit integer
q16	q	signed 16-bit integer
u32	ul	unsigned 32-bit integer

- **hopper.c**

This file contains the main program. The main() function configures everything, then drops into a perpetual power-saving idle loop. The actual work is done by the watchdog timer interrupt service routine, WatchDogISR().

## 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 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.

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.

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

<b>Products</b>		<b>Applications</b>	
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>	Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>	Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>	Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>	Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>	Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>	Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>	Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
		Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
		Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
		Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

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

Copyright © 2006, Texas Instruments Incorporated