Application Report
SPNA217–May 2015

Sine Wave Generation Using PWM With Hercules™ N2HET and HTU

Charles Tsai

ABSTRACT

This application report illustrates how to generate sine waves using the versatile programmable high-end timer (N2HET) and its companion data transfer unit (HTU). The code example can be run in either the Hercules hardware development kit (HDK) or the LaunchPad™ development kit. The application report shows the N2HET program examples, the steps to setting up the N2HET and HTU registers as well as basic system settings utilizing the HalCoGen.

This document assumes that you have some basic understanding of the N2HET terms as well as some understanding of both the HET integrated development environment (IDE) and HalCoGen tools.

Project collateral and source code discussed in this application can be downloaded from the following URL: http://www.ti.com/lit/zip/spna217.

Contents

List of Figures

List of Tables

Hercules, LaunchPad are trademarks of Texas Instruments.
All other trademarks are the property of their respective owners.

1 IntroductionF

N2HET is a fifth-generation Texas Instruments (TI) advanced intelligent timer coprocessor module based on the very long instruction word (VLIW) instruction set architecture. The instruction set, based mostly on very simple, but comprehensive instructions provides sophisticated timing functions for real-time applications. The high resolution hardware channels allow greater accuracy for widely used timing functions such as pulse width modulation (PWM) generation. In the microcontroller, there is also the dedicated High End Timer Transfer Unit (HTU) that moves data efficiently between the N2HET RAM and the system RAM. The purpose of this application report is to illustrate how to use the N2HET coupled with the HTU to create a sine wave in an efficient manner without CPU's intervention.

1.1 Sine Wave and PWM Overview

Pulse Width Modulation (PWM) is a method of encoding a voltage onto a fixed frequency carrier wave. The frequency of the PWM will be fixed while the duty cycle will vary between 0% and 100%. The percentage of the on-time will be proportional to the output signal voltage. For example, a 0% duty cycle produces a 0 V output while a 100% duty cycle produces a peak-to-peak voltage V_{p-p} equal to the Vccio, which is the I/O supply voltage to the microcontroller. The nominal Vccio is 3.3 V in Hercules microcontrollers. A 50% duty cycle would have produced an output voltage equal to 1.65 V. The PWM method is a low cost way of implementing a digital-to-analog converter (DAC). By time-varying the duty cycle percentage, a sine waveform can be generated.

The basic idea of generating a sine waveform using the PWM method is to first digitize the sine wave and encode the duty cycle corresponding to each sample point, see Figure 1. In Figure 1 the sine wave is digitized over 12 sample points. 12 samples means that each sample is taken at a angular step of 30^{o} of a circle. The sine value of each sample and its corresponding duty cycle is shown in Table 1.

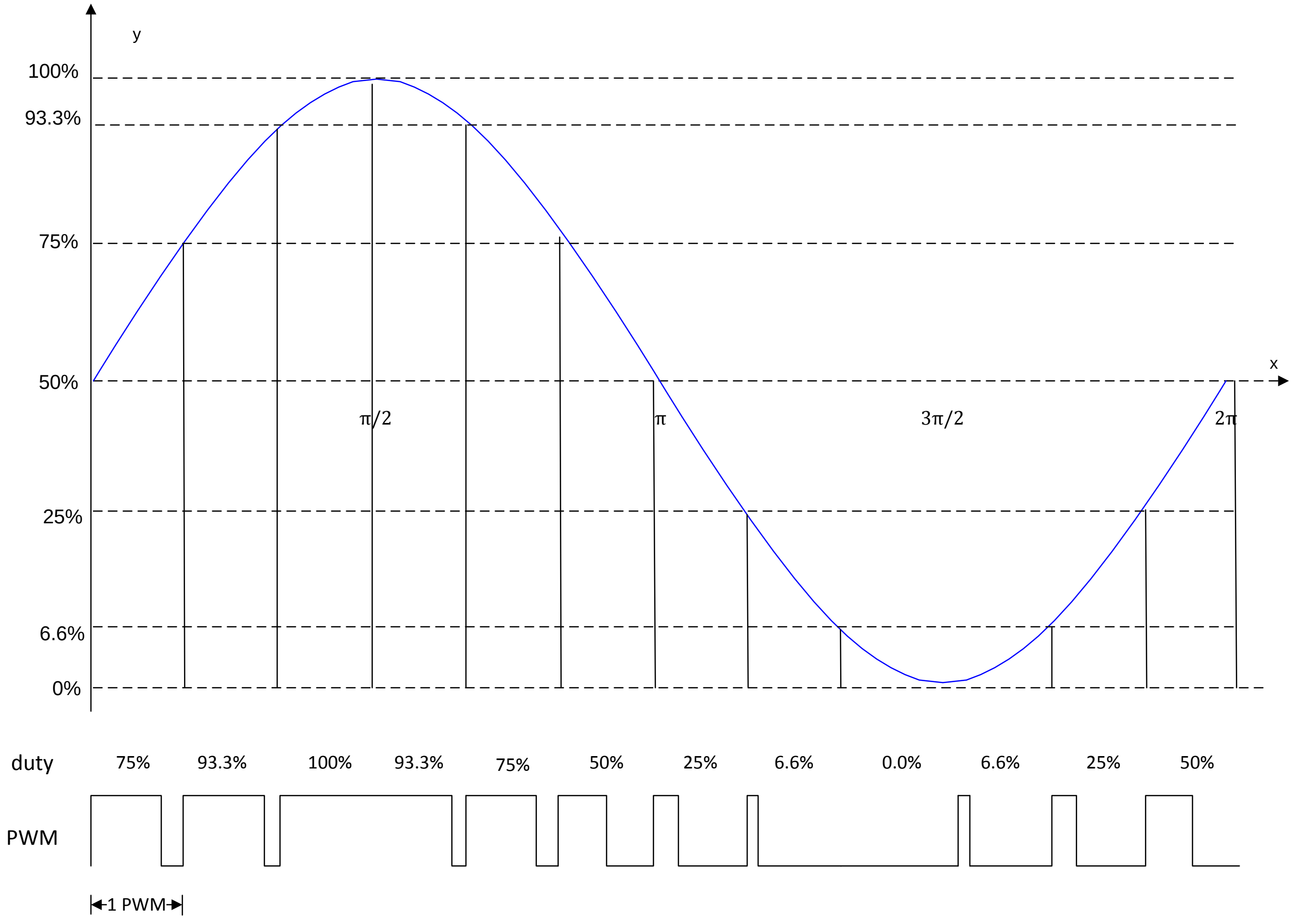


Figure 1. 12 Sample Points Sine Wave With the Corresponding Modulated PWM

Table 1. 12-Sample Digitized Sine Table

Degree	Radian	sine(Radian)	Normalize to 0	%
0	0.0000	0.0000	1.0000	0.50
30	0.5233	0.4998	1.4998	0.75
60	1.0467	0.8658	1.8658	0.93
90	1.5700	1.0000	2.0000	1.00
120	2.0933	0.8666	1.8666	0.93
150	2.6167	0.5011	1.5011	0.75
180	3.1400	0.0016	1.0016	0.50
210	3.6633	-0.4984	0.5016	0.25
240	4.1867	-0.8650	0.1350	0.07
270	4.7100	-1.0000	0.0000	0.00
300	5.2333	-0.8673	0.1327	0.07
330	5.7567	-0.5025	0.4975	0.25

1.2 Hardware Dataflow

The key to generating a sine wave is the ability to continuously generating a time-varying PWM. Time-varying PWM means that the duty cycle will change from one sample point to another. The duty cycle at each sample point is a representation of the sine amplitude. Two things are necessary in order to accomplish this. First, store the different duty cycles needed to reconstruct the sine wave digitally. These can be pre-calculated duty cycles in a lookup table stored in the system RAM. Then, a mechanism to continuously transfer these duty cycles to N2HET RAM is needed. One way to do this is to use the CPU to read the sine lookup table and then write to the N2HET RAM by using interrupt. Each time the N2HET finishes one PWM cycle it generates an interrupt to the CPU. In the ISR, the CPU will transfer the new duty cycle. However, this is not the most efficient way as it takes away CPU's bandwidth from performing other critical tasks. The interrupt also involves saving and restoring context; this can take some cycles to complete. The ISR latency may become the limiting factor in determining the fastest sampling frequency.

In this application report, the on-chip HTU module is used to handle this chord. HTU not only has the capability to transfer data from the lookup table to the N2HET RAM but also has the capability to do this task in a circular fashion repeatedly. This is exactly what is needed because the task is not generating one sine cycle but a continuous sine wave. The HTU can automatically repeat from the first element of the lookup table after the lookup table is completely transferred. Figure 2 illustrates the dataflow of the HTU transfer.

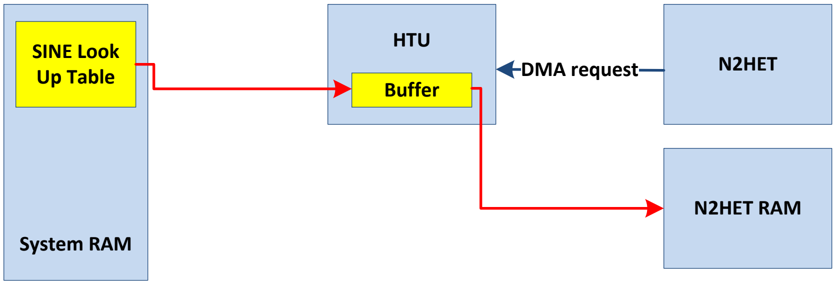


Figure 2. HTU Dataflow

2 Sine Wave Frequency Calculation

There is one important relationship established from Figure 1; the frequency of the sine wave depends on the PWM period and the number of sample points. This relationship can be written in Equation 1.

$$F_O = \frac{F_{PWM}}{samples} \quad (1)$$

where:

F_O = The sine wave frequency

F_{PWM} = The PWM frequency

samples = The number of samples to digitize the sine wave

It is understood from the equation that if the output sine wave frequency is increased, either the PWM frequency needs to be increased or the number of sample points needs to be decreased. Decreasing the number of sample points increases quantization errors. The sine wave produced using only 12 points will not be smooth after passing the PWM through a Low-Pass Filter (LPF). The higher the number of sample points, the less quantization errors. Therefore, don't keep the number of sample points too low in order to synthesize a nice sine wave. In this application report, 128 sample points are used. You can choose another number of samples to suit your application need.

Since the number of sample points to 128 is fixed, the only parameter played with in order to change the output frequency is by changing the PWM frequency. However, there is a limitation on the fastest F_{PWM} you can get. The F_{PWM} is a function of the reference clock (VCLK2) to N2HET and the prescaler factor for the Loop Resolution Period (LRP).

This relationship can be written as shown in Equation 2:

$$F_{PWM} = \frac{1}{(SINE\_FREQ\_DIVIDER * LRP)} \tag{2}$$

where:

SINE_FREQ_DIVIDER = Number of LRP, SINE_FREQ_DIVIDER >= 1

$$LRP = lr * hr * VCLK2 \tag{3}$$

where:

hr = 1, the hr (high resolution) prescaler factor is fixed in this application report

lr = 16, 32, 64, or 128. lr is the low resolution prescaler factor.

The equation for F_{PWM} can be re-written as shown in Equation 4.

$$F_{PWM} = \frac{1}{(SINE\_FREQ\_DIVIDER * lr * VCLK2)} \tag{4}$$

Substituting Equation 4 into Equation 1, will give you Equation 5.

$$F_O = \frac{1}{(SINE\_FREQ\_DIVIDER * lr * VCLK2 * samples)} \tag{5}$$

2.1 Example 1

Suppose the following:

VCLK2 = 11.11ns (90 MHz)

lr = 16

SINE_FREQ_DIVIDER = 1

samples = 128

The fastest F_{PWM} will be:

$$F_{PWM} = \frac{1}{(16 * 11.11\,ns)} \cong 5625.6\,KHz \tag{6}$$

The fastest sine wave frequency will be:

$$F_O = \frac{F_{PWM}}{samples} = \frac{5625.6\,KHz}{128} \cong 44\,KHz \tag{7}$$

2.2 Example 2

Suppose the following:

VCLK2 = 12.5ns (80 MHz)

lr = 128

SINE_FREQ_DIVIDER = 1024

samples = 128

The F_{PWM} will be:

$$F_{PWM} = \frac{1}{(1024 * 128 * 12.5\,ns)} \cong 610.4\,Hz \tag{8}$$

The sine wave frequency will be:

$$F_O = \frac{F_{PWM}}{samples} = \frac{610.4\,Hz}{128} \cong 4.7\,Hz \tag{9}$$

2.3 How to Obtain a Desired Sine Wave Frequency

By playing with the SINE_FREQ_DIVIDER and the lr prescaler, it is possible to obtain the desired sine wave frequency as you can also express:

$$SINE\_FREQ\_DIVIDER = \frac{1}{F_O * samples * lr * VCLK2} \tag{10}$$

Suppose the following:

F_O = 120 Hz, the desired output sine wave frequency

VCLK2 = 12.5 ns (80 MHz)

lr = 128

samples = 128

The SINE_FREQ_DIVIDER will be:

$$SINE\_FREQ\_DIVIDER = \frac{1}{(120 * 128 * 128 * 125\,ns)} \cong 41 \tag{11}$$

Table 2. Examples of Sine Wave Frequency (F_O) vs. Different SINE_FREQ_DIVIDER and lr Prescaler With VCLK2 = 90 MHz

	SINE_FREQ_DIVIDER									
lr	**1**	**2**	**3**	**4**	**5**	**16**	**32**	**64**	**128**	**256**
16	44 KHz	22 KHz	14.6 KHz	11 KHz	8.8 KHz	2.7 KHz	1.4 KHz	687 Hz	343 Hz	172 Hz
32	22 KHz	11 KHz	7.3 KHz	5.5 KHz	4.5 Hz	1.4 KHz	687 Hz	343 Hz	172 Hz	86 Hz
64	11 KHz	5.5 KHz	3.6 KHz	2.7 KHz	2.2 Hz	687 Hz	343 Hz	172 Hz	86 Hz	43 Hz
128	5.5 KHz	2.7 KHz	1.8 KHz	1.4 KHz	1.1 KHz	343 Hz	172 Hz	86 Hz	43 Hz	21 Hz

3 Low-Pass Filter (LPF)

The PWM generated out of the microcontroller needs to pass through a LPF to remove the high frequency components. There is no one size fits all solution when it comes to the types of filters to use. The N2HET demonstrated in this application report can generate a wide range of sine wave frequencies. Therefore, a filter with the proper pass-band bandwidth should be used accordingly. For simplicity reason, a first order passive low-pass filter using only the low-cost RC components is used, as shown in Figure 3. This first order low pass filter has its cut-off frequency at about 724Hz. Therefore, the sine wave that is demonstrated is in the pass-band that is slower than the cut-off frequency. The focus of this application report is to show how to write the N2HET code to produce a sine wave. This document does not touch on the optimum filter to use. Extra information on the analysis of filters is provided in Section 7.

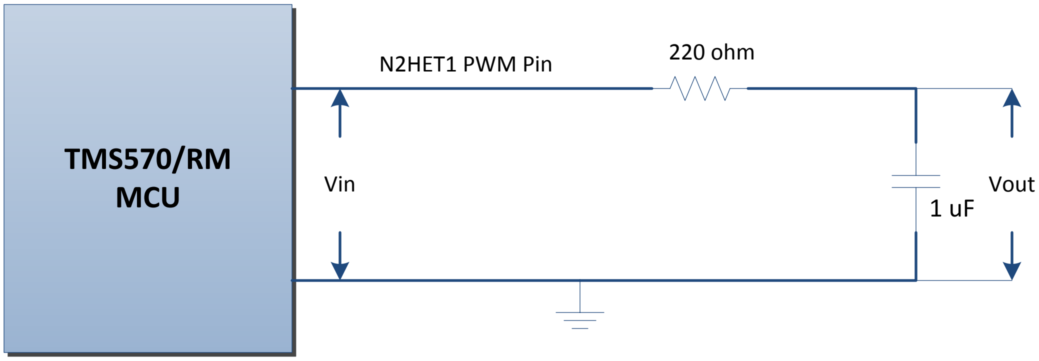


Figure 3. Low-Pass Filter

4 N2HET Implementation

4.1 N2HET1 Sine Wave Generation Flowchart

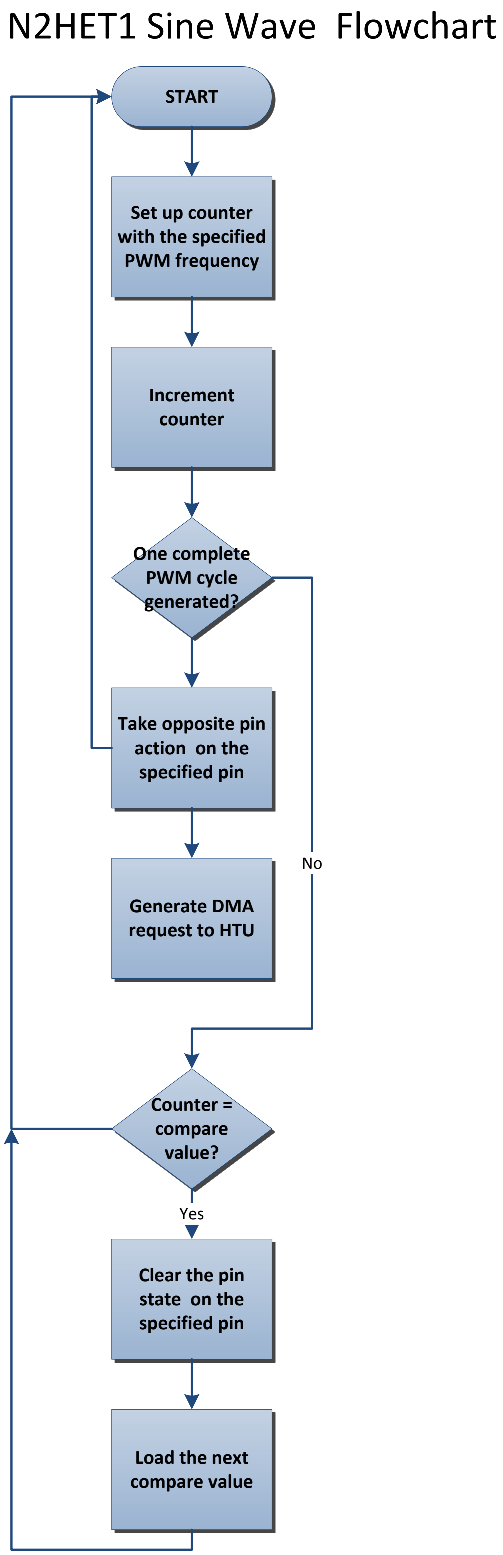

N2HET1 Sine Wave Flowchart
START
Set up counter with the specified PWM frequency
Increment counter
One complete PWM cycle generated?
Take opposite pin action on the specified pin
Generate DMA request to HTU
No
Counter = compare value?
Yes
Clear the pin state on the specified pin
Load the next compare value

Figure 4. N2HET1 Sine Wave Flowchart

1. Set up the time-base counter with the specified PWM period.
2. Increment the counter
3. Check if the counter has reached the specified pre-load value which defines the PWM period. If yes, go to step 4. If no, go to step 6.
4. Set the specified pin to the opposite state.
5. Generate a DMA request to the HTU. Wait for one complete PWM period before generating the request to HTU to retrieve the next duty cycle compare value. This way you will have a smooth sine wave.
6. Compare the existing time-base counter with the current compare value. If there is a match, go to 7. If no match, go to step 1.
7. Clear the pin state on the specified pin.
8. Load the new compare value.
9. Go back to step 1.

4.2 N2HET1 Sine Wave Program

The example N2HET1 program code is illustrated below. Directives using .equ are parameters used to configure the program; you can change these parameters. By default, these parameters have initial values that are small for quick simulation using HET IDE. The host CPU will overwrite these parameters in the host side application code.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This example illustrates how to generate a PWM asymmetrically. The duty cycle
; is to be updated by the host application by writing the new duty cycle (compare value) to
; the MOV32 data field. When there is a compare match, the MOV32 instruction will move
; the new compare value to the ECMP instruction. The update of the ECMP is considered
; synchronous as the update is only done with respect to the LRP (loop resolution period)
; time base and only when there is a compare match.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


PWM_PERIOD            .equ 1
PWM_PIN_NUM           .equ 9
INIT_COMPARE          .equ 1
INIT_HR_DELAY         .equ 0

; Use CNT to generate a virtual counter. The counter period is changable by the host
; application. When counter reaches the PWM_PERIOD value, a DMA request is generated. The
; HTU is setup to take this request and return a new duty cycle. The new duty cycle will
; be written to the MOV32 data field as the new compare value.
L00   CNT { reqnum=0,request=GENREQ,reg=A,irq=OFF,max=PWM_PERIOD};

; ECMP is setup in high resolution mode to create a compare value against the current
; counter value in CNT instruction. Whenever, there is a match, it will set the specified
; pin low until the end of the PWM period. The pin starts as high at the beginning of the
; PWM cycle. The specified pin is changable by the host application.
L01   ECMP { next=L03,hr_lr=HIGH,en_pin_action=ON,cond_addr=L02,pin=PWM_PIN_NUM,
             action=PULSELO,reg=A,irq=OFF,data=INIT_COMPARE,hr_data=INIT_HR_DELAY};

; MOV32 instruction is only executed when ECMP has a compare match. When this instruction
; is executed, it will copy its data field value which contains the latest duty cycle (
; the compare value) back to ECMP's data field. The MOV32's data field is updated by the HTU
; with the new duty cycle value when responding to the request by the CNT instruction.
L02   MOV32 { remote=L01,type=IMTORE&GREM,reg=NONE,data=INIT_COMPARE,hr_data=INIT_HR_DELAY};

; Branch back to the beginning.
L03   BR { next= L00, cond_addr=L00, event= NOCOND }
```

4.3 N2HET Assembler

The N2HET code needs to be translated into the opcode that the N2HET can execute. This is done with the N2HET assembler *hetp*. The assembler can be executed on the command line. Here is an example of the command line to use for assembling the code for N2HET1 instance:

```
hetp -n0 -hc32 Sine_Wave.het
```

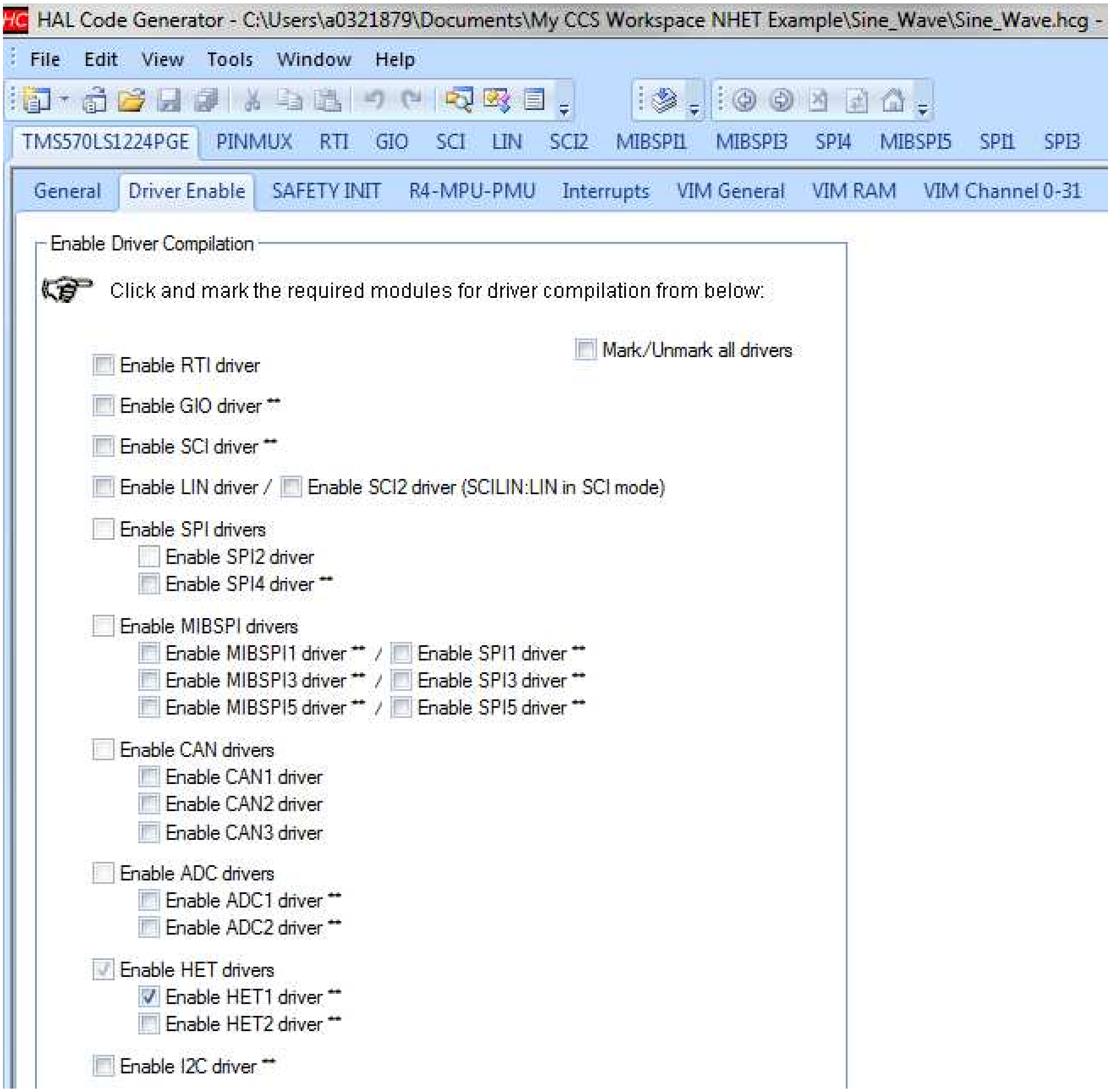
The -hc32 argument produces C header file Sine_Wave.h and source file Sine_Wave.c for the Texas Instruments TI's C compiler. Specifying the -n0 argument will allow the assembler to produce unique header and source files for the N2HET1 instance.

5 Host Side Setup

5.1 HalcoGen Setup

This example utilizes the HalCogen tool to configure the device. The target device selected in the HalCoGen is the TMS570LS1224 running at 160 MHz. It can easily be ported to other devices by following the steps below.

1. Create a new project: File → New → Project. Name the project Sine_Wave.
2. Enable the N2HET1 driver and disable the rest.


HAL Code Generator - C:\Users\a0321879\Documents\My CCS Workspace NHET Example\Sine_Wave\Sine_Wave.hcg -
File Edit View Tools Window Help
TMS570LS1224PGE PINMUX RTI GIO SCI LIN SCI2 MIBSPI1 MIBSPI3 SPI4 MIBSPI5 SPI1 SPI3
General Driver Enable SAFETY INIT R4-MPU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31

Enable Driver Compilation

Click and mark the required modules for driver compilation from below:

□ Mark/Unmark all drivers

□ Enable RTI driver

□ Enable GIO driver **

□ Enable SCI driver **

□ Enable LIN driver / □ Enable SCI2 driver (SCILIN:LIN in SCI mode)

□ Enable SPI drivers
- □ Enable SPI2 driver
- □ Enable SPI4 driver **

□ Enable MIBSPI drivers
- □ Enable MIBSPI1 driver ** / □ Enable SPI1 driver **
- □ Enable MIBSPI3 driver ** / □ Enable SPI3 driver **
- □ Enable MIBSPI5 driver ** / □ Enable SPI5 driver **

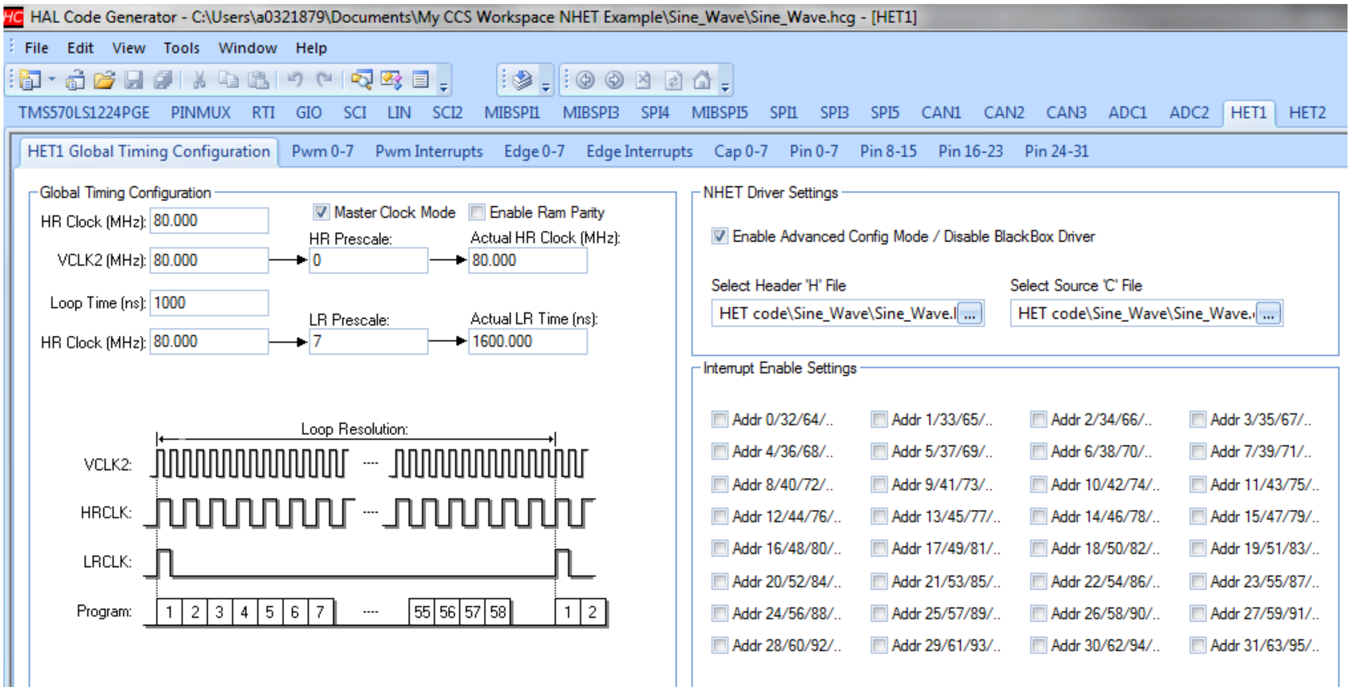
□ Enable CAN drivers
- □ Enable CAN1 driver
- □ Enable CAN2 driver
- □ Enable CAN3 driver

□ Enable ADC drivers
- □ Enable ADC1 driver **
- □ Enable ADC2 driver **

☑ Enable HET drivers
- ☑ Enable HET1 driver **
- □ Enable HET2 driver **

□ Enable I2C driver **

3. Configure the N2HET1. Make sure to enable the checkbox for "Enable Advanced Config Mode/Disable BlackBox Driver" and provide the header file (Sine_Wave.h) and source file (Sine_Wave.c) generated in Section 4.3. This step bypasses the default blackbox N2HET code provided by HalCoGen so that the custom N2HET code in Section 4.2 can be loaded to the N2HET module.


HAL Code Generator - C:\Users\a0321879\Documents\My CCS Workspace NHET Example\Sine_Wave\Sine_Wave.hcg - [HET1]
File Edit View Tools Window Help
TMS570LS1224PGE PINMUX RTI GIO SCI LIN SCI2 MIBSPI1 MIBSPI3 SPI4 MIBSPI5 SPI1 SPI3 SPI5 CAN1 CAN2 CAN3 ADC1 ADC2 HET1 HET2
HET1 Global Timing Configuration Pwm 0-7 Pwm Interrupts Edge 0-7 Edge Interrupts Cap 0-7 Pin 0-7 Pin 8-15 Pin 16-23 Pin 24-31
Loop Resolution:
VCLK2:
HRCLK:
LRCLK:
Program:
1 2 3 4 5 6 7 .... 55 56 57 58 1 2

Global Timing Configuration

HR Clock (MHz): 80.000

VCLK2 (MHz): 80.000

Loop Time (ns): 1000

HR Clock (MHz): 80.000

☑ Master Clock Mode ☐ Enable Ram Parity

HR Prescale: 0

Actual HR Clock (MHz): 80.000

LR Prescale: 7

Actual LR Time (ns): 1600.000

NHET Driver Settings

☑ Enable Advanced Config Mode / Disable BlackBox Driver

Select Header 'H' File: HET code\Sine_Wave\Sine_Wave.l

Select Source 'C' File: HET code\Sine_Wave\Sine_Wave.(

Interrupt Enable Settings

☐ Addr 0/32/64/..	☐ Addr 1/33/65/..	☐ Addr 2/34/66/..	☐ Addr 3/35/67/..
☐ Addr 4/36/68/..	☐ Addr 5/37/69/..	☐ Addr 6/38/70/..	☐ Addr 7/39/71/..
☐ Addr 8/40/72/..	☐ Addr 9/41/73/..	☐ Addr 10/42/74/..	☐ Addr 11/43/75/..
☐ Addr 12/44/76/..	☐ Addr 13/45/77/..	☐ Addr 14/46/78/..	☐ Addr 15/47/79/..
☐ Addr 16/48/80/..	☐ Addr 17/49/81/..	☐ Addr 18/50/82/..	☐ Addr 19/51/83/..
☐ Addr 20/52/84/..	☐ Addr 21/53/85/..	☐ Addr 22/54/86/..	☐ Addr 23/55/87/..
☐ Addr 24/56/88/..	☐ Addr 25/57/89/..	☐ Addr 26/58/90/..	☐ Addr 27/59/91/..
☐ Addr 28/60/92/..	☐ Addr 29/61/93/..	☐ Addr 30/62/94/..	☐ Addr 31/63/95/..

4. Select File → Generate Code to generate the code.

5.2 Configurable Macros

Three macros are utilized to configure the N2HET1 operation. Below are the list of changeable macros.

```
/* Below 3 macros are changeable by user */

/* This example uses the below equations to calculate the SINE wave frequency (F):
 *
 * (1)  F = 1 / (PWM_PERIOD * samples)
 * (2)  PWM_PERIOD is the period of the PWM Base Frequency and
 *      PWM_PERIOD = (SINE_FREQ_DIVIDER  * LRP) where SINE_FREQ_DIVIDER >= 1
 * (3)  LRP = hr * lr. hr is fixed to 1 in this program. The supported lr
 *      in this program are 16, 32, 64, and 128. lr smaller than 16 will not
 *      give enough time slots for the N2HET to execute and also too short for
 *      the HTU to complete the transfer.
 * (4)  samples = the number of samples to digitize the sine wave
 *
 * The fastest SINE frequency this example can support for SINE_FREQ_DIVIDER = 1 and
 * samples = 128 would be:
 * F = 1 / ((SINE_FREQ_DIVIDER  * LRP) * Samples) or
 * F = 1 / ((1  * LRP) * 128)
 * Suppose HCLK=160MHz, VCLK2=80MHz, 1 LRP = 128 VCLK2 :
 * 1 LRP = 128 * 12.5ns = 1.6uS and F = 1 / (1.6uS * 128) ~= 4.88KHz
 *
 * With 1 LRP = 16 VCLK2 it is possible to generate a sine wave frequency at:
 * F = 1 / (16 * 12.5ns * 128) ~=39KHz.
 *
 * SINE_FREQ_DIVIDER is used to divide down the SINE frequency.
 * Setting SINE_FREQ_DIVIDER to 2 will generate 2.44KHz sine wave.
 * SINE_FREQ_DIVIDER can be any integer value larger or equal to 1
 */
#define SINE_FREQ_DIVIDER 16

/* allowable LR Prescaler factors are 16, 32, 64 and 128. Anything less
 * than 32 will not have enough time slots for the N2HET program to
 * run.
 * LRPFC can be either 5, 6 or 7.
 * 7 -> one lr = 128 VCLK2
 * 6 -> one lr = 64 VCLK2
 * 5 -> one lr = 32 VCLK2
 * 4 -> one lr = 16 VCLK2
 */
#define LRPFC 7

 /* The pin number to output the sine wave signal */
#define NHET1_PIN_PWM PIN_HET_9

/****************************************************************************/
/* The PWM Period to be loaded to NHET1 CNT instruction. The minimum
 * PWM base frequency of the PWM is 1 * LRP */
#define CNT_MAX_PERIOD SINE_FREQ_DIVIDER

/* Number of sample points to digitize the SINE wave */
#define SAMPLE_SIZE 128

#if SAMPLE_SIZE == 128
/* sine_table_percent array contains the duty cycle of each sample point on the
 * sine wave. The percent values will then be converted to actual compare values
 * during run-time. The reason that the actual compare values are not hardcoded
 * is because this program is designed to support programmable sine frequency.
 * When the sine frequency is changed using SINE_FREQ_DIVIDER then the CNT_MAX_PERIOD
 * will need to be adjusted. The compare values need to be adjusted as well. Keeping
 * only the duty cycle percent values in the lookup table allows us to simply
 * calculate the final compare values by multiplying the percent with the CNT_MAX_PERIOD.
 *
 */
float sine_table_percent[SAMPLE_SIZE] = {0.5,0.52452141,0.548983805,0.573328313,0.597496346,
        0.62142974,0.645070896,0.668362917,0.691249749,0.71367631,0.735588627,0.756933966,
        0.777660956,0.797719715,0.817061967,0.835641164,0.853412591,0.870333478,0.886363105,
        0.901462892,0.9155965,0.928729914,0.940831527,0.951872215,0.961825406,0.970667147,
        0.978376158,0.984933888,0.990324553,0.994535181,0.997555637,0.999378653,0.999999841,
        0.999417707,0.997633651,0.994651967,0.990479831,0.985127283,0.978607206,0.970935291,
        0.962130001,0.952212527,0.941206738,0.929139121,0.916038718,0.901937056,0.886868075,
        0.870868039,0.853975454,0.836230976,0.81767731,0.798359106,0.778322858,0.757616784,
        0.736290719,0.714395985,0.691985276,0.669112527,0.645832783,0.622202072,0.598277264,
        0.574115936,0.549776238,0.525316747,0.500796326,0.47627399,0.451808753,0.427459496,
        0.403284818,0.379342899,0.355691359,0.332387119,0.309486264,0.287043908,0.265114062,
        0.243749504,0.223001649,0.202920432,0.18355418,0.164949501,0.14715117,0.130202021,
        0.114142845,0.099012291,0.084846772,0.07168038,0.059544802,0.048469244,0.03848036,
        0.029602191,0.021856103,0.015260738,0.009831969,0.00558286,0.002523639,0.000661668,
        0.000001426,0.000544506,0.002289597,0.005232501,0.009366135,0.014680552,0.02116296,
        0.02879776,0.037566577,0.047448308,0.05841917,0.070452761,0.08352012,0.097589799,
        0.112627937,0.128598342,0.14546258,0.163180064,0.181708154,0.20100226,0.221015947,
        0.241701051,0.263007789,0.284884883,0.307279683,0.330138292,0.353405699,0.377025906,
        0.400942069,0.425096628,0.449431454,0.47388798
};
#else
uint32 sine_table[SAMPLE_SIZE] = {0.5,0.548983805,0.597496346,0.645070896,0.691249749,0.735588627,

0.777660956,0.817061967,0.853412591,0.886363105,0.9155965,0.940831527,0.961825406,0.978376158,
        0.990324553,0.997555637,0.999999841,0.997633651,0.990479831,0.978607206,0.962130001,
        0.941206738,0.916038718,0.886868075,0.853975454,0.81767731,0.778322858,0.736290719,
        0.691985276,0.645832783,0.598277264,0.549776238,0.500796326,0.451808753,0.403284818,
        0.355691359,0.309486264,0.265114062,0.223001649,0.18355418,0.14715117,0.114142845,
        0.084846772,0.059544802,0.03848036,0.021856103,0.009831969,0.002523639,0.000001426,
        0.002289597,0.009366135,0.02116296,0.037566577,0.05841917,0.08352012,0.112627937,
        0.14546258,0.181708154,0.221015947,0.263007789,0.307279683,0.353405699,0.400942069,
        0.449431454
 };
#endif
uint32 sine_table[SAMPLE_SIZE] = {0};
/* USER CODE END */
```

5.3 CPU Main()

The host CPU's job is to first initialize the device and configure the N2HET1 and HTU. The rest of the time the host CPU stays in a loop.

```
void main(void)
{
/* USER CODE BEGIN (3) */

    /* initialize HTU1 based on HalCoGen settings */
    htuInit();

    /* initialize NHET1 based on HalCoGen settings */
    hetInit();

    /* Configure additional settings of NHET1 based on the macros settings */
    configNHET1();

    while(1);

/* USER CODE END */
}
```

5.4 N2HET Setup

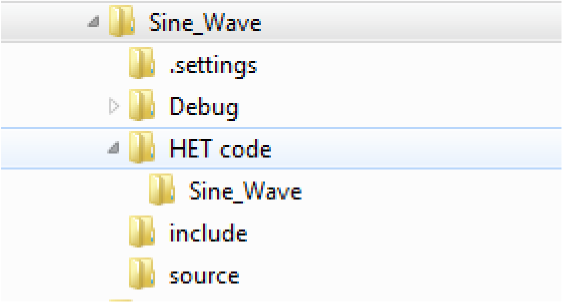
```
/* configureNHET1 configures the selected NHET1 pin to output PWM. It
 * also loads the specified PWM period into the NHET1 RAM */
void configNHET1()
{
 /* configure the LRP prescaler, the hr is always 1 and lr can be
     * either 16, 32, 64 or 128 */
    hetREG1->PFR = LRPFC << 8;

    /* calculate_ecmp_compare() will calculate the actual compare values
     * as well as the high resolution delay values for each sample point
  * of the same wave to be loaded into the NHET1 ECMP pin instructions  */
    calculate_ecmp_compare();

    /* Enable DMA request on channel 0 of HTU. In the CNT instruction, the
  * DMA request is asserted to channel 0 of the HTU module */
    hetREG1->REQENS = 1 ;

    /* Set the selected pin to output. */
    hetREG1->DIR = (1 << NHET1_PIN_PWM) ;

    /* Load the PWM period based on the defined macro to the CNT instruction */
    hetRAM1->Instruction[pHET_L00_0].Control = (uint32)(CNT_MAX_PERIOD - 1) |
    (hetRAM1->Instruction[pHET_L00_0].Control & 0xFFFD0000);

    /* Configure the pin number to output the PWM */
    hetRAM1->Instruction[pHET_L01_0].Control =
            (hetRAM1->Instruction[pHET_L01_0].Control & 0xFFFFE0FF) |
            (NHET1_PIN_PWM << 8);
}

/* This function calculates the compare value and the high resolution delay
 * to be loaded into the NHET1 ECMP instruction for generating the PWM
 * DUTY cycle
 */
void calculate_ecmp_compare()
{

    int i=0;
    /* The number of sample points is fixed using SAMPLE_SIZE. The SINE
     * wave frequency is F = 1 / ((SINE_FREQ_DIVIDER * LRP) * Samples).
     * The way to generate a divided sine wave frequency is to increase
     * the PWM base frequency inversely. As we change the PWM base frequency
     * we also need to adjust the compare values to generate the PWM duty
     * cycle at each sine wave sample point.
  */

  for (i=0;i<=SAMPLE_SIZE;i++){
      /* CNT_MAX_PERIOD * 128 gives the total number of high resolution
       * clocks in one PWM_PERIOD. The multiplication is done by performing
       * a left shift of CNT_MAX_PERIOD by 7 bits.
       */
      sine_table[i] = (sine_table_percent[i] * (CNT_MAX_PERIOD << 7));
  }
}
```

5.5 HTU Setup

```
/* htuInit setups the HTU to transfer new compare value stored in
 * the sine_table to the NHET ECMP instruction at each completion of
 * one PWM period. The transfer is done in a circular fashion. */
void htuInit(void){

    /* DCP0 CPx element count = 1, frame count = SAMPLE_SIZE */
    htuRAM1 ->DCP[0].ITCOUNT = 0x00010000 + SAMPLE_SIZE;

    /* DCP0 CPx DIR = main memory to NHET                      */
    /* SIZE = 32-bit                                        */
    /* ADDMH = 16 bytes                                        */
    /* ADDFM = post-increment main memory                      */
    /* TMBA = circular buffer A                                */
    /* TMBB = one shot buffer B (buffer B not used)         */
    /* IHADDR = 0x28 MOV32 data field                          */
    htuRAM1 ->DCP[0].IHADDRCT = (htuRAM1 ->DCP[0].IHADDRCT & 0x0) |
                                 0x1 << 23 |  // DIR
                                 0x0 << 22 | // SIZE
                                 0x0 << 21 | // ADDMH
                                 0x0 << 20 | // ADDFM
                                 0x1 << 18 | // TMBA
                                 0x0 << 16 | // TMBB
                                 0x28 <<  0;  // IHADDR
    /* DCP0 CPA start address of source buffer */
    htuRAM1 ->DCP[0].IFADDRA = (unsigned int)sine_table;

    /* enable DCP0 CPA */
    htuREG1 ->CPENA = 0x00000001;
    /* enable HTU */
    htuREG1 ->GC = 0x00010000;
}
```

5.6 Project Directory Structure

This example project is named Sine_Wave; Figure 5 shows the project directory structure. The N2HET program is contained under the HET code folder.



```
Sine_Wave
    .settings
    Debug
    HET code
        Sine_Wave
    include
    source
```

Figure 5. Sine_Wave Project Directory Structure

6 Examples

Suppose we want to generate a 300Hz sine wave assuming the VCLK2 is 80 MHz and lr equals to 128. Calculate the SINE_FREQ_DIVIDER to be 16.27 using Equation 10. However, the SINE_FREQ_DIVIDER needs to be a whole number. Therefore, you need to round 16.27 down to 16. Since SINE_FREQ_DIVIDER equals to 16, the closest sine wave frequency that can be obtained is 305.18Hz using Equation 5. Figure 6 shows a clean sine wave being generated at the specified frequency. FFT analysis of the sine wave affirms the fundamental frequency at about 300Hz as shown in the lower zoom in window of Figure 7 while the rest of harmonics quickly die off at -60dB shown in the upper zoom out window. Further improvement to attenuate the higher harmonic components can be achieved by using higher order filters.

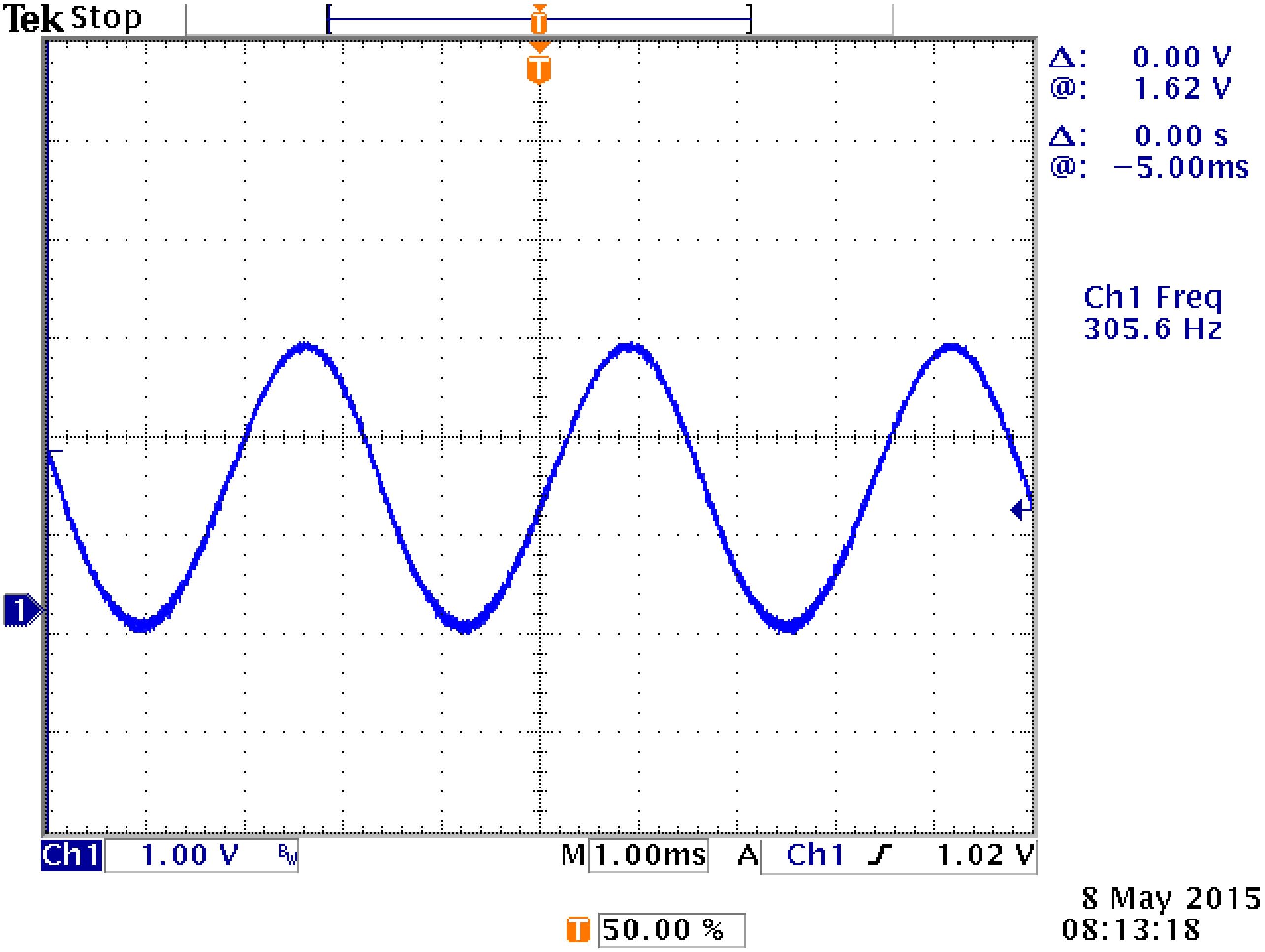


Figure 6. A 305Hz Sine Wave

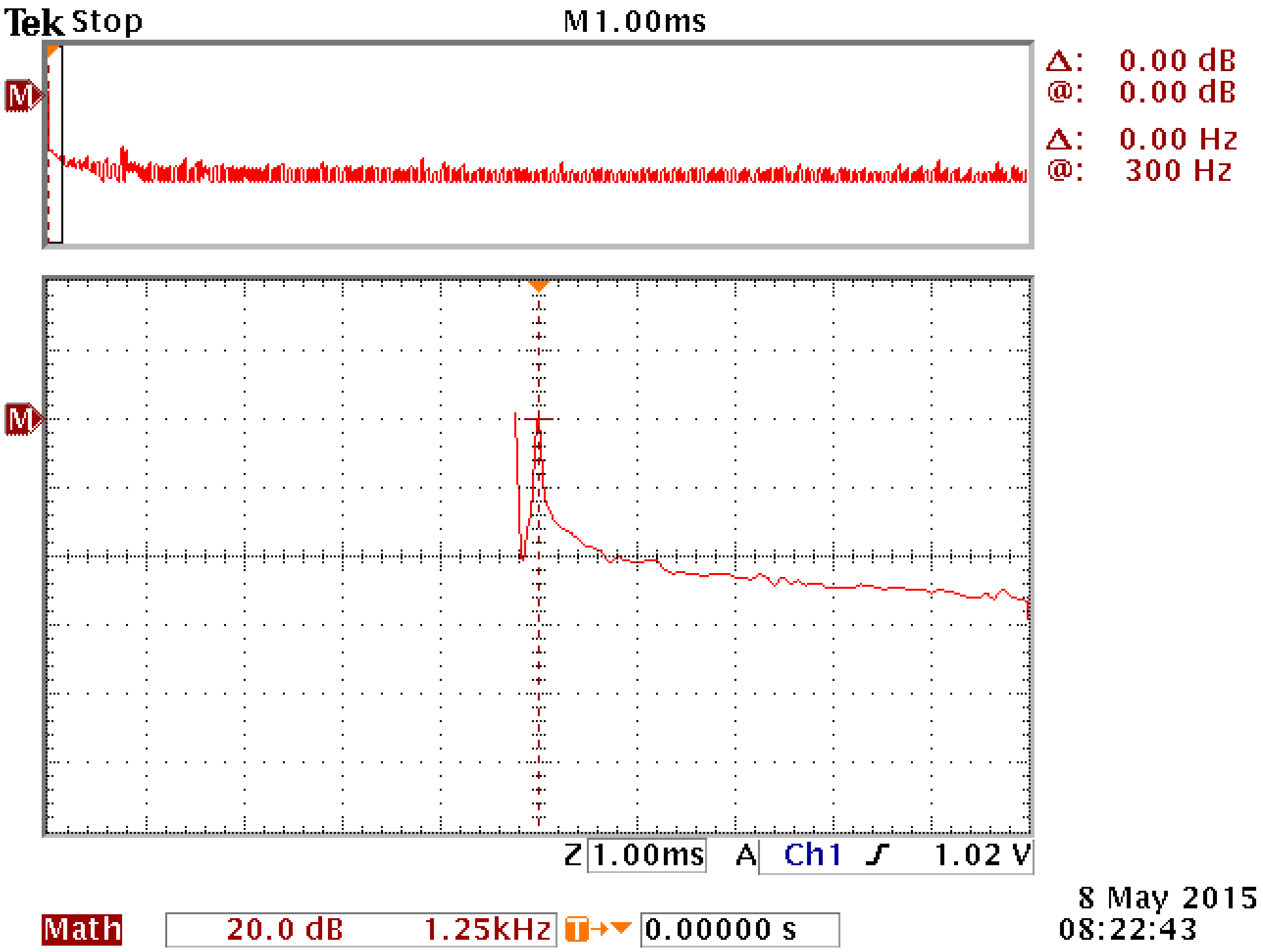


Figure 7. FFT Analysis of the Sine Wave

7 References

- *HET Integrated Development Environment User's Guide* (SPNU483)
- *NHET Getting Started* (SPRABA0B)
- *Enhanced High-End Timer (NHET) Assembler User's Guide* (SPNU490)
- *Triangle/Trapezoid Wave Generation Using PWM With Hercules N2HET* (SPNA220)
- *Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Controller* (SPRAA88)
- *PWM DAC Using MSP430 High-Resolution Timer* (SLAA497)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

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

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 relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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 significant portions 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

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