Triangle/Trapezoid Wave Generation Using PWM With Hercules™ N2HET

Charles Tsai

ABSTRACT

This application report illustrates how to generate various forms of triangle and trapezoid waves using the versatile programmable high-end timer (N2HET). The examples 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 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/spna220.

Contents

List of Figures

List of Tables

1   Introduction

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 period and pulse measurements, output compare and PWMs.

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 power 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, it is possible to generate an arbitrary analog waveform.

Using the PWM method as a DAC is nothing new. Many microcontrollers on the market can accomplish this simple task. What is needed is a PWM capable hardware in the microcontrollers. This normally involves a simple time-based counter that will either count up or count down until the counter expires. The programmable width of the counter will determines the frequency of the PWM. While the counter is either counting up or down, the current counter value is compared against a programmable match value that defines the duty cycle. The output PWM signal could be setup to remain set while a match is not found. As soon as the counter value matches the programmable match value, the PWM will clear the PWM output. The pin remains clear until the counter expires and sets the pin again. Figure 1 illustrates the generation of an un-modulated PWM signal.
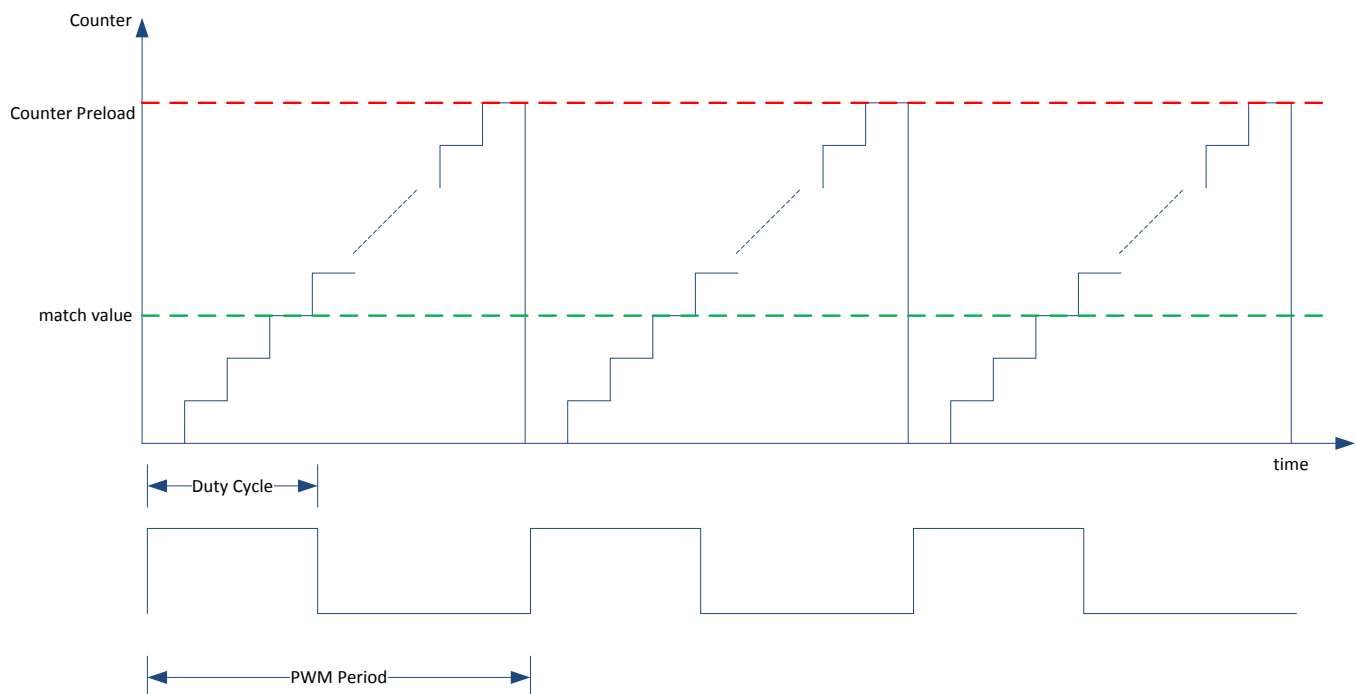

Figure 1. An Unmodulated PWM Signal

However, for a time-vary PWM output, the duty cycle needs to be changed. This normally involves generating an interrupt to the CPU upon a compare match or when the counter reaches its pre-loaded count. For example, to generate a ramp waveform, this could mean generating an interrupt to the CPU each time the counter expires at its programmable pre-load value. In the interrupt service routine, the CPU increments the duty cycle by a desire amount and updates the compare value register, see Figure 2. Generating interrupts to the CPU has some drawbacks; it can slow down the CPU from doing other tasks. The interrupt latency can also impact the maximum achievable output signal frequency.
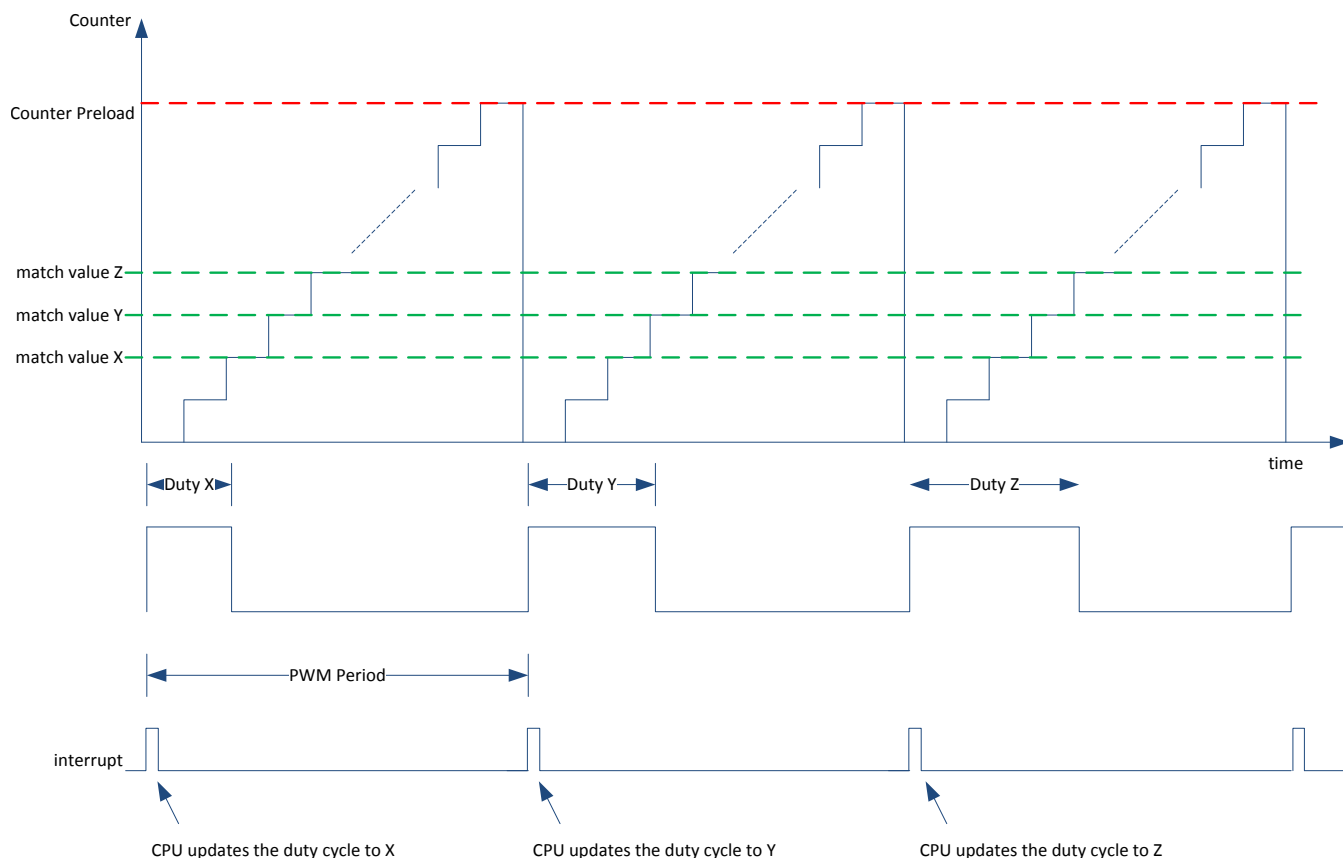
Figure 2. A Modulated PWM Signal

On the other hand, some of the mentioned drawbacks can be avoided using the N2HET since it is a coprocessor. You can write N2HET code to modulate the duty cycle of the PWM without involving the CPU at all. How and when to modulate the duty cycle is under the N2HET's control. This capability is the main focus of this application note.

In this application note, we will use the N2HET to generate time-varying PWM signals. The output signal will go through a simple analog low-pass filter to remove the high frequency components. The filter output renders various forms of triangle and trapezoid waves. The versatility of the N2HET allows you to control the slope of the ramp up and ramp down when forming a triangle. By controlling the amount of time the PWM stays at its maximum duty cycle and at 0% duty, an arbitrary trapezoid waveform can also be generated. Figure 3 shows the waveforms that are demonstrated in this application report.
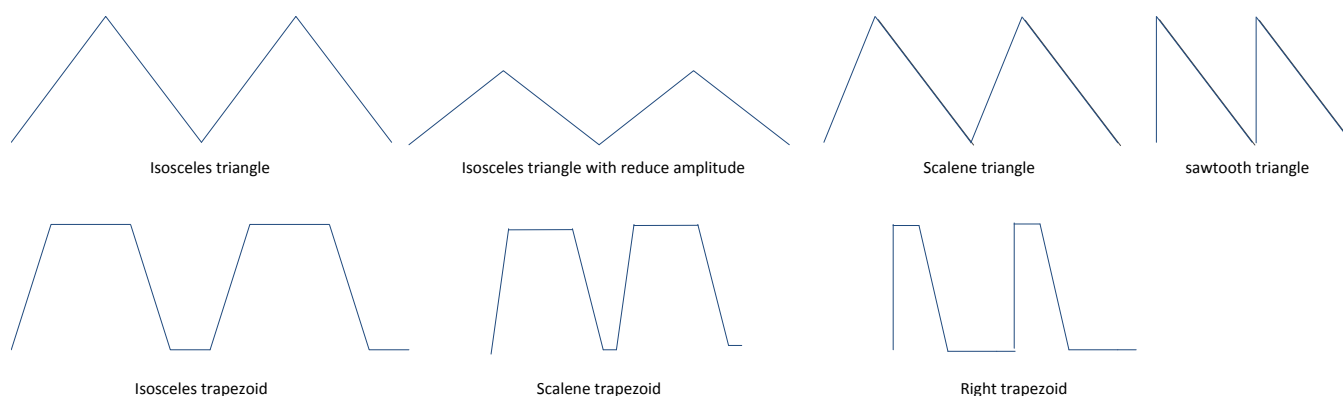
Figure 3. Triangle and Trapezoid Waves

2 Low-Pass Filter

The low-pass filter is used to remove the high frequency components that are produced on the PWM output. For simplicity reason, a first order passive low-pass filter using only the low-cost RC components are used, as shown in Figure 4. The focus of this application report is to show how to write the N2HET code to produce various waveforms. This document does not touch on the optimum filter to use. Extra information on the analysis of filters is provided in Section 7.

Figure 4. Low-Pass Filter

3   N2HET Implementation

3.1   N2HET1 Triangle/Trapezoid Wave Generation Flow Chart

N2HET1 Triangle/Trapezoid Wave Flowchart

Figure 5. N2HET1 Triangle/Trapezoid Wave Flow Chart

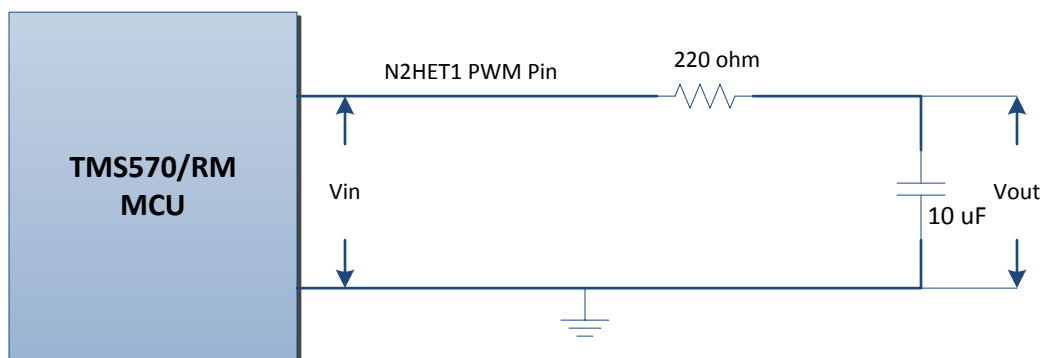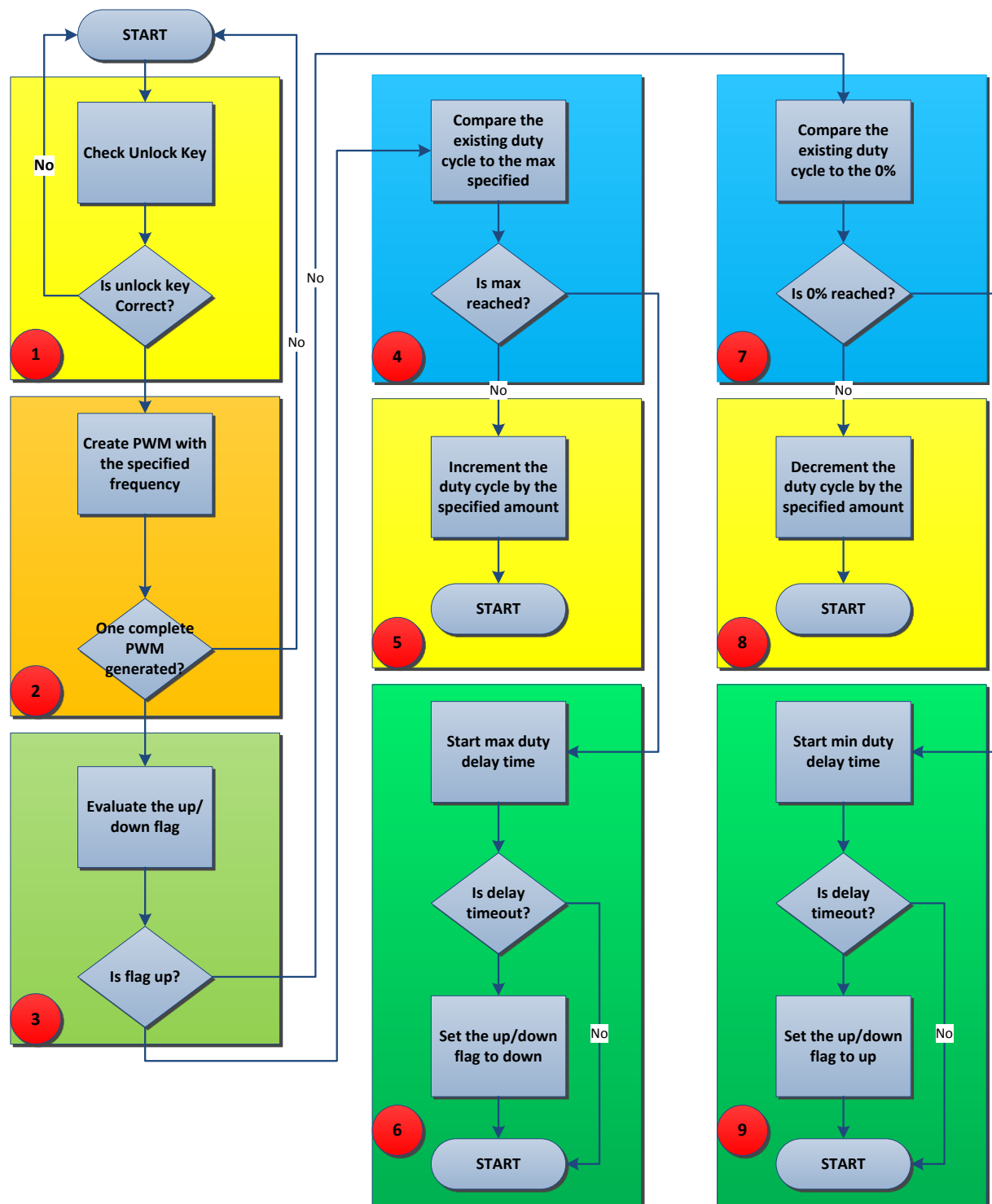1. The N2HET1 is put into a self-loop until a proper unlock key is written to the N2HET1. While the N2HET1 is still locked, the host CPU can setup various parameters for the N2HET1 program.

2. Generate the specified PWM starting with 0% duty cycle. The duty cycle will be self modified by the N2HET1 itself either in increasing order or decreasing order. Wait until one full PWM period is generated before changing to a new duty cycle.

3. Evaluate the up/down flag to determine whether the duty cycle should increase or decrease. The flag will be initialized to zero after reset, meaning to increment the duty cycle.

4. Compare the existing duty cycle to the programmable maximum duty cycle. The maximum duty cycle is a parameter changeable by the host CPU before the N2HET program starts. If the maximum is reached, go to step 6.

5. Increment to the next duty cycle percentage by the amount that is programmable by the host CPU before the N2HET program starts.

6. Start a programmable timer. The timer is used to hold the PWM at the maximum duty cycle for a programmable amount of time. This step is needed for creating trapezoid waveforms. To generate a triangle wave, this timer delay will be set to zero. The timer length is programmable by the host CPU before the N2HET program starts.

7. Compare the existing duty cycle to the 0% duty cycle. If the 0% is reached, go to step 9.

8. Decrement to the next duty cycle by the amount that is programmable by the host CPU. Note that the amount to decrement can be different than the amount to increment.

9. Start a programmable timer. The timer is used to hold the PWM at the 0% duty cycle for a programmable amount of time. Note that this minimum duty cycle timer length can be different from the maximum duty cycle timer length. This step is needed for creating trapezoid waveforms. To generate a triangle wave this timer delay will be set to zero. The timer length is programmable by the host CPU before the N2HET program starts.

3.2   *N2HET1 Triangle/Trapezoid 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 code is to be loaded into N2HET1's RAM. This code will generate a
; time-varying PWM signal to render either triangle waves or trapezoid waves.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; PWM frequency to be generated
PWM_PERIOD          .equ 2
; The pin number that will output the PWM signal
PWM_PIN_NUM         .equ 9
; The initial maximum duty cycle (LR compare value) to be generated.
INIT_COMPARE        .equ 3
; The initial maximum duty cycle (HR compare value) to be generated.
INIT_COMPARE_HR     .equ 0
; amount of increment in terms of LRP. Note the total amount to increment is
; equal to DUTY_INCREMENT + DUTY_INCREMENT_HR.
DUTY_INCREMEMT      .equ 0
; amount of increment in terms of HR.
DUTY_INCREMEMT_HR   .equ 1
; amount of decrement in terms of LRP
DUTY_DECREMEMT      .equ 0
; amount of decrement in terms of HR
DUTY_DECREMEMT_HR   .equ 1
; amount of timer delay to keep the PWM at 0% duty cycle
LOW_DELAY           .equ 0
; amount of timer delay to keep the PWM at maximum duty cycle
HIGH_DELAY          .equ 0
; key to unlock N2HET
UNLOCK_KEY          .equ 0xA
```

```
; The data field of the MOV32 instruction contains an initial value (0x5) that
; is not equal to the key to unlock the N2HET program. First the MOV32
; instruction moves the initial value to a temporary register T
L00    MOV32 { remote=DUMMY,type=IMTOREG,reg=T,data=0x5};


; Compare the register T value with the key to unlock N2HET. The key to unlock
; is 0xA. If the key is not matched then go back to L00. The CPU is supposed
; to write the proper key (0xA) to unlock the N2HET
L01    ECMP { next=L00,hr_lr=LOW,cond_addr=L02,pin=0,reg=T,data=UNLOCK_KEY};


; Creating a virtual counter using CNT which will determine the period of
; the PWM to be generated. The initial small max count allows for quick
; simulation which can later be changed by the host CPU.
L02    CNT { reg=A,irq=OFF,max=PWM_PERIOD};


; Use ECMP to determine the duty cycle of the PWM on the specified pin. The
; pin field and the duty cycle are changeable by the CPU.
L03    ECMP { hr_lr=HIGH,en_pin_action=ON,cond_addr=L04,pin=PWM_PIN_NUM,
            action=PULSELO,reg=A,irq=OFF,data=0,hr_data=0};


; Only when the CNT reaches the max count will the program go to the
; conditional address. We want to wait for one complete PWM waveform to be
; generated before changing the duty cycle. When CNT reaches the max
; value it will set the Z flag.
L04    BR { next=L00,cond_addr=L05,event=Z};


; the data field in this ADD acts as a up/down flag. To create either a triangle
; or a trapezoid wave we first need to create a ramp up waveform. The PWM will first
; increase the duty cycle until it reaches the specified maximum duty cycle before
; it starts to decrease the duty or stay at the maximum duty. The up/down flag is
; used to create two different paths in the flow to alternate before increasing duty
; cycle vs decreasing duty cycle.
L05    ADD { src1=ZERO,src2=ZERO,dest=NONE,data=0};


; Move the up/down flag to a temp register T.
L06    MOV32 { remote=L05,type=REMTOREG,reg=T};


; Compare this up/down flag to 0. 0 means to increase the duty cycle and 1
; means to decrease the duty cycle.
L07    ECMP { next=L16,cond_addr=L08,pin=0,reg=T,data=0};


; move the ECMP DF which contains the compare value for duty cycle creation
; to register R
L08    MOV32 { remote=L03,type=REMTOREG,reg=R};


; Subtract the current compare value from the max duty cycle stored in
; REM_DUTY. The result will be stored in register S.
L09    SUB { src1=REM,src2=R,dest=S,remote=REM_DUTY,data=0};


; If the subtraction result is more than 0 then it means it has not
; reached the max duty cycle we will increase the duty cycle. If it is
; zero or less than zero then we have reached the max duty cycle and we
; will change the up/down flag to down position.
L10    BR { next=L12,cond_addr=L11,event=GT};


; Add specified amount to the existing compare value (duty cycle) to specify the new
; duty cycle. The amount to increment is changeable by CPU before the N2HET program starts.
; After the addition, jump back to the beginning of the program
L11    ADD { next=L15,src1=R,src2=IMM,dest=S,rdest=REM,remote=L03,data=DUTY_INCREMEMT,
            hr_data=DUTY_INCREMEMT_HR};


; Insert a timer delay after the maximum duty cycle is reached. A timer delay here has the
; effect of creating the high side of a trapezoid waveform. If the timer is zero then it
; becomes a triangle wave.
L12    DJZ { next=L00,cond_addr=L13,reg=NONE,data=HIGH_DELAY};
```

```
; After the above DJZ expires on its counter we need to reload the DJZ counter to the
; specified amount of delay.
L13   MOV32 { next=L14,remote=L12,type=IMTOREG&REM,reg=NONE,data=HIGH_DELAY};


; Now change the up/down flag to down by moving a 1 to the up/down flag
L14   MOV32 { remote=L05,type=IMTOREG&REM,reg=NONE,data=1};


; Branch to the beginning
L15   BR { next=L00,cond_addr=L00,event=NOCOND};


; move the ECMP DF to register R which contains the current compare value
; (duty cycle)
L16   MOV32 { remote=L03,type=REMTOREG,reg=R};


; Subtract the current duty cycle by the specified amount. This amount of decrement is
; changeable by CPU before the N2HET program starts. When this instruction is executed
; the first time, the current duty cycle is at the maximum duty cycle. Here we are creating
; the ramp down part of the triangle/trapezoid waveforms.
L17   SUB { src1=R,src2=IMM,dest=S,rdest=NONE,data=DUTY_DECREMEMT,
            hr_data=DUTY_DECREMEMT_HR};


; As long as the subtraction result is greater than zero, we will keep
; decreasing the duty cyle or otherwise we will again change the up/down
; flag to up position. The destination register is S which contains the
; subtraction result.
L18   BR { next=L19,cond_addr=L20,event=N};


; Move the subtraction result to the ECMP DF as the new duty cycle
L19   MOV32 { next=L00,remote=L03,type=REGTOREM,reg=S};


; Insert a timer delay after the 0% duty cycle is reached. A timer delay here has the
; effect of creating the low side of a trapezoid waveform. If the timer is zero then it
; becomes a triangle wave.
L20   DJZ { next=L00,cond_addr=L21,reg=NONE,data=LOW_DELAY};


; Reset the increment delay to the specified amount.
L21   MOV32 {remote=L20,type=IMTOREG&REM,reg=NONE,data=LOW_DELAY};


; Move the value 0 to the up/down flag so in the next LRP the program
; flow will execute the path to increase duty cycle.
L22   MOV32 { remote=L05,type=IMTOREG&REM,reg=NONE,data=0};


; Branch to beginning
L23   BR { next=L00,cond_addr=L00,event=NOCOND};


; REM_DUTY data field stores the maximum duty cycle the PWM to be generated.
; The host CPU can change this value.
REM_DUTY  ECMP { next=REM_DUTY,cond_addr=REM_DUTY,pin=0,reg=A,data=INIT_COMPARE,
                 hr_data=INIT_COMPARE_HR};
DUMMY   BR { next=DUMMY,cond_addr=DUMMY,event=NOCOND,irq=OFF};
```

3.3   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 Trapezoid_Wave.het
```

The -hc32 argument produces C header file Trapezoid_Wave.h and source file Trapezoid_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.

4 Analog Output Waveform Frequency Calculation

The analog output frequency after the low-pass filter depends on the reference clock frequency (VCLK2) for the N2HET as well the duty cycle resolution. The resolution is the smallest increment in the analog output voltage that can be divided between 0 V and the power rail (Vccio). For a 10-bit resolution, there are a total of 1024 steps that can be divided between 0 V and Vccio = 3.3 V. Each step is equal to 3.3 V / 1024 ~= 3.2 mV. The higher the resolution, the less errors on the output voltage. However, the higher the resolution, the lower the output frequency.

The triangle wave frequency **F**o can be calculated using Equation 1.

$$F_O = \frac{F_{PWM}}{(2 \times DCR)} \tag{1}$$

where:

$\mathbf{F}_{PWM}$ = the carrier PWM frequency

DCR = duty cycle resolution or 2^N where N is the number of bits

$\mathbf{F}_{PWM}$ can also be expressed as shown in Equation 2.

$$F_{PWM} = \frac{1}{T_{PWM}} \tag{2}$$

where:

$$T_{PWM} = VCLK2 \times DCR \tag{3}$$

$VCLK2$ = the reference clock cycle to the N2HET module

Therefore,
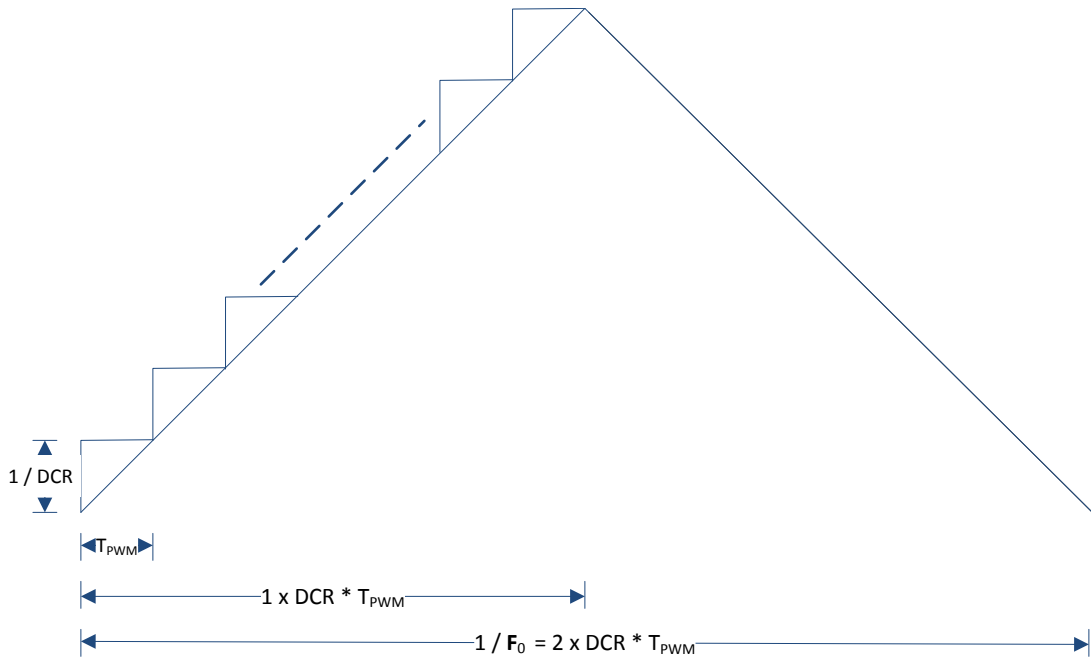
$$F_O = \frac{1}{(T_{PWM} \times 2 \times DCR)} \tag{4}$$

$$F_O = \frac{1}{(VCLK2 \times DCR \times 2 \times DCR)} = \frac{1}{(2 \times VCLK2 \times DCR^2)} \tag{5}$$

For example,

$VCLK2$ = 90 MHz or 11.11 ns and

DCR = 1024 steps of resolution

$$F_O = \frac{1}{(2 \times 11.11\,ns \times 1024^2)} = 42.9 Hz \tag{6}$$

1 / DCR

T$_{PWM}$

1 x DCR * T$_{PWM}$

1 / **F**$_0$ = 2 x DCR * T$_{PWM}$

Calculating the output frequency for a trapezoid wave is similar. The triangle wave frequency **F**$_o$ can be calculated using Equation 7.

$$F_O = \frac{1}{\left(2 \times VCLK2 \times DCR^2 + HTIME + LTIME\right)}$$

(7)

where,

HTIME = time duration on the high phase of the trapezoid wave

LTIME = time duration of the low phase of the trapezoid wave

HTIME

1 / DCR

T$_{PWM}$

LTIME

1 x DCR * T$_{PWM}$

1 x DCR * T$_{PWM}$

1 / **F**$_0$ = 2 x DCR * T$_{PWM}$ + HTIME + LTIME
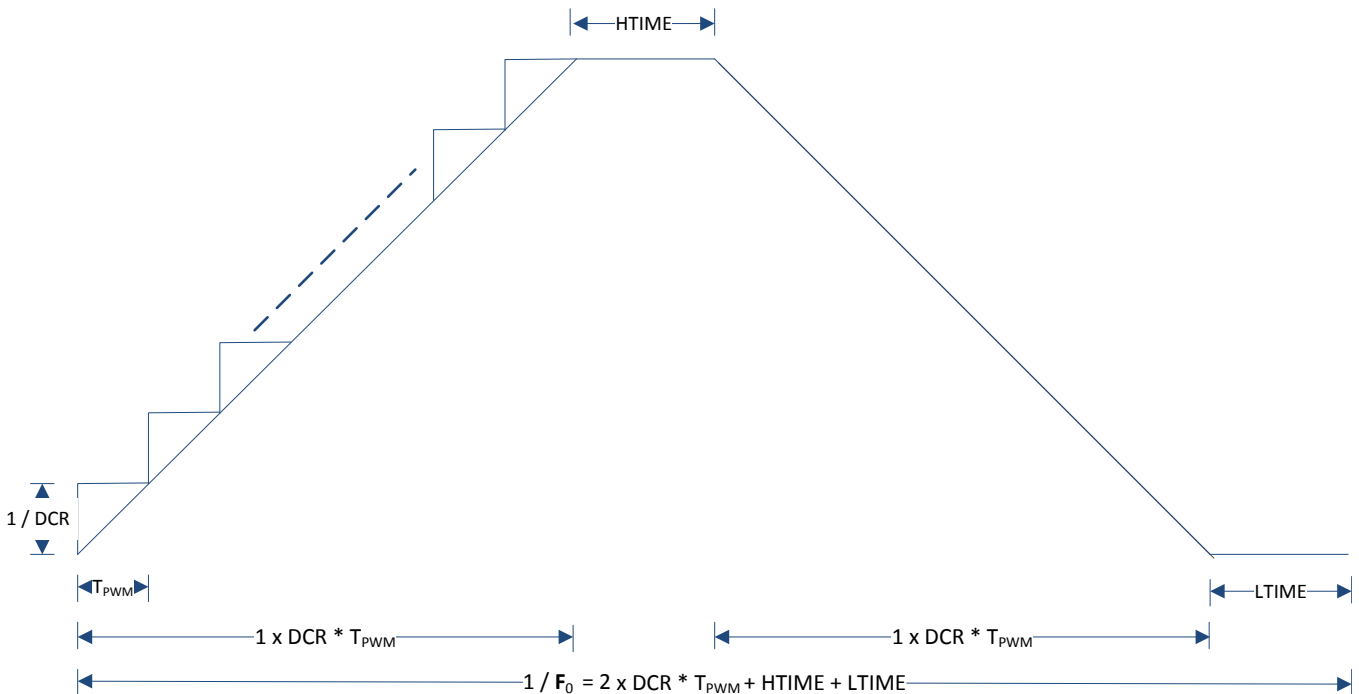
5 CPU Side Setup

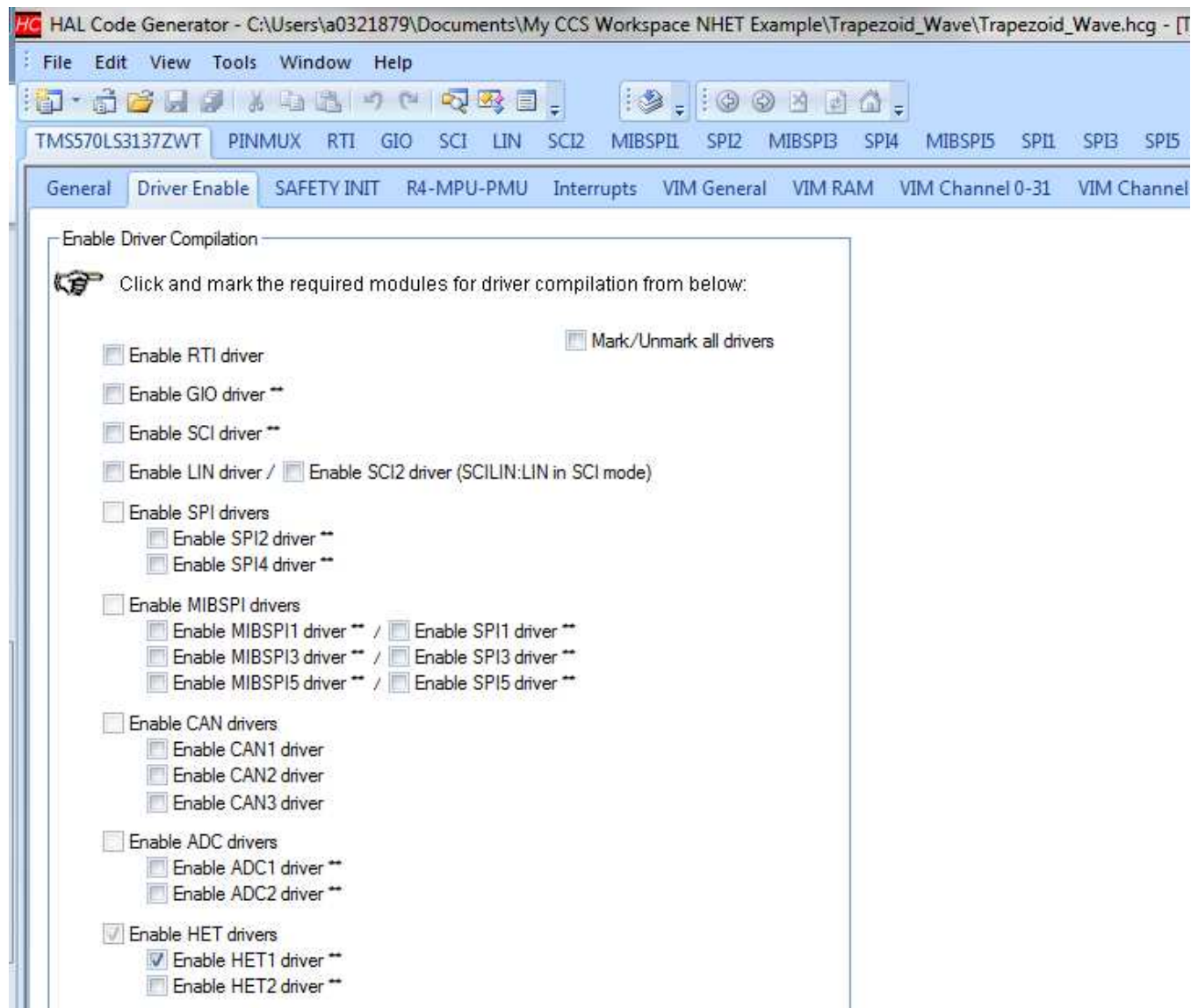## 5.1 HalcoGen Setup

This example utilizes the HalCogen tool to configure the device. The target device selected in the HalCoGen is the TMS570LS3137. It can easily be ported to other devices by following the below steps.

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

3. Configure the N2HET1. Make sure to enable the checkbox for "Enable Advanced Config Mode/Disable BlackBox Driver" and provide the header file (Trapezoid_Wave.h) and source file (Trapezoid_Wave.c) generated in Section 3.3. This step bypasses the default blackbox N2HET code provided by HalCoGen so that the custom N2HET code in Section 3.2 can be loaded to the N2HET module.
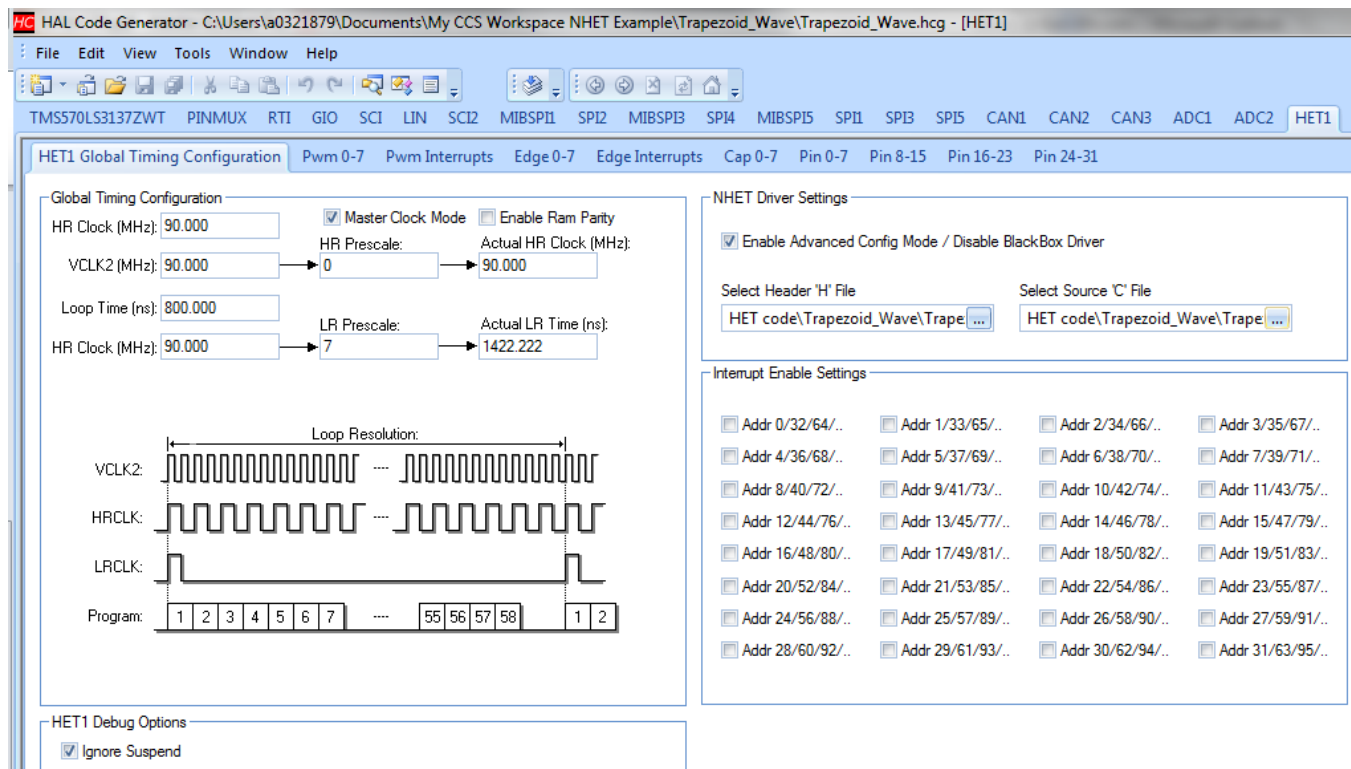
4. Select File → Generate Code to generate the code.

5.2   *CPU Main() Code Setup*

On the host CPU side, its main task is to first initialize the device and configure both the N2HET1 module. Various macros are utilized to configure the N2HET1 module. Below are the list of changeable macros.

```
/**************************************************************************
 * Below 8 macros are changeable by user to generate various different    *
 * waveforms.                                                             *
 **************************************************************************/

/* PWM_PERIOD defines the period of the carrier PWM frequency. PWM_PERIOD is
 * expressed in terms of number of LRP (Loop Resolution Period). The frequency
 * of the carrier PWM will remain constant. However, the N2HET will
 * modulate the duty cycle to generate a time-varying PWM signal. This
 * time-varying PWM signal is then passed through a low pass filter to remove
 * unwanted high frequency components, an analog waveform is thus rendered.
 *
 * The LRP period depends on the reference VCLK2 frequency to the N2HET
 * module as well as the programmable LR Prescaler factor (lr). It can be
 * expressed as:
 * 1 LRP = VCLK2 * lr
 * If VCLK2 = 11.11ns and lr = 128 then
 * 1 LRP = 11.11 ns * 128 = 1.42us
 * Note that for this example, the high resolution prescale factor (hr)
 * is always configured to 1.
 *
 * The width of the PWM_PERIOD also defines the resolution of the output
 * analog signal to be rendered. With PWM_PERIOD = 1, there are a total
 * of 128 VCLK2 cycles. This means that the N2HET will module its duty
 * cycle between these 128 steps. Thefore, with PWM_PERIOD = 1, the
 * output analog signal will have a log2(128) = 8-bit resolution. If
 * user desires higher resolution, the PWM_PERIOD can be changed to a
 * higher value such as 32. With PWM_PERIOD = 32, the duty cycle can be
 * modulated among a total of 32 * 128 = 4096 steps and hence render a
```

```
 * log2(4096) = 12-bit of resolution. The higher the resolution the
 * less noise on the final output. However, the higher the resolution,
 * the lower the final output frequency of the signal to be generated.
 */
#define PWM_PERIOD     32

/* PWM_DUTY defines the maximum duty cycle the N2HET is allowed to
 * modulate to. The PWM_DUTY is expressed in terms of percent (%).
 * With PWM_DUTY = 100%, the maximum amplitude on the output signal
 * will be equal to Vccio. Setting PWM_DUTY = 50% will create a
 * maximum amplitude of Vccio / 2. Note that the maximum amplitude
 * will also depend on the type of filter used and its respective
 * RC values. */
#define PWM_DUTY       100.0

/* allowable LR Prescaler factors are 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
 */
#define LRPFC 7

/* The NH2ET1 program will automatically increase the PWM
 * modulation from 0% duty cycle to maximum duty cycle
 * specified in PWM_DUTY. When PWM_DUTY is reached it starts
 * to decrease the duty cycle from PWM_DUTY to 0%.
 * DUTY_INCREMENT specifies the delta amount of duty cycle to
 * change from one duty cycle to the next duty cycle while
 * the duty cycle is increasing. Increasing DUTY_INCREMENT
 * has the effect of decreasing the duty cycle resolution for
 * a given PWM_PERIOD. This is expressed in terms of (%).
 * For example specifying DUTY_INCREMENT equal to 5 will mean
 * the duty cycle will start at 0% and the next duty cycle
 * will be 5% at a 5% increment. If 0 is specified, then the
 * N2HET1 will increment the duty cycle at 1 VCLK2 clock resolution */
#define DUTY_INCREMENT   0.0

/* DUTY_DECREMEMT specifies the delta amount of duty cycle to
 * change from one duty cycle to the next duty cycle while
 * the duty cycle is decreasing. This is expressed in terms
 * of (%). */
#define DUTY_DECREMENT   0.0

/* The MAX_DUTY_TIMER is the amount of wait time to stay at
 * the maximum duty cycle. Change to a non-zero value will
 * create trapozoid waveform. A zero value will create a
 * trianagle wave
 *
 * The MAX_DUTY_TIMER is expressed in terms of number of
 * LRP. */
#define MAX_DUTY_TIMER  0

/* The MIN_DUTY_TIMER is the amount of wait time
 * to stay at the minimum duty cycle. Change to
 * a non-zero value will create trapezoid waveform.
 * A zero value will create a trainable wave */
#define MIN_DUTY_TIMER  0

/* Pin number in N2HET1 to generate the PWM. */
#define NHET1_PIN_PWM PIN_HET_9

/***************************************************************************/
```

```
/* Below macros are for internal calculation. Do not change.              */
/***************************************************************************/

/* The PWM Period to be loaded to N2HET1 CNT instruction */
#define CNT_MAX_PERIOD (PWM_PERIOD - 1)

/* The max PWM Duty to be loaded to N2HET1 ECMP Instruction */
#define ECMP_MAX_DUTY (PWM_PERIOD * PWM_DUTY / 100.0)

/* DELTA_INCREMENT is the amount to be loaded to N2HET1 to increment the
 * duty cycle */
#define DELTA_INCREMENT (CNT_MAX_PERIOD * DUTY_INCREMENT / 100)

/* DELTA_DECREMENT is amount to be loaded to N2HET1 to decrement the
 * duty cycle */
#define DELTA_DECREMENT (CNT_MAX_PERIOD * DUTY_DECREMENT / 100)


/* Unlock key to for N2HET2 */
#define UNLOCK_KEY 0xAU
```

Table 1. Triangle Wave Frequency for Different LRPFC and PWM_PERIOD With VCLK2 = 90MHz

LRPFC	PWM_PERIOD							
	1	2	4	8	16	32	64	128
5	43.9KHz	11KHz	2.7KHz	686.7Hz	171.7Hz	42.9Hz	10.7Hz	2.7Hz
6	11KHz	2.7KHz	686.7Hz	171.7Hz	42.9Hz	10.7Hz	2.7Hz	0.67Hz
7	2.7KHz	686.7Hz	171.7Hz	42.9Hz	10.7Hz	2.7Hz	0.67Hz	0.17Hz

Table 2. Triangle Wave Resolution for Different LRPFC and PWM_PERIOD

LRPFC	PWM_PERIOD							
	1	2	4	8	16	32	64	128
5	5-bit	6-bit	7-bit	8-bit	9-bit	10-bit	11-bit	12-bit
6	6-bit	7-bit	8-bit	9-bit	10-bit	11-bit	12-bit	13-bit
7	7-bit	8-bit	9-bit	10-bit	11-bit	12-bit	13-bit	14bit

NOTE: Table 1 and Table 2 do not take into account the type of analog low-pass filters used.

The triangle and trapezoid waveform generation mainly handles the N2HET itself without CPU intervention. The host CPU's job is to first initialize the device and configure the N2HET1. The rest of the time the host CPU stays in a loop.

```
void main(void)
{
/* USER CODE BEGIN (3) */
    /* This example uses the N2HET1 to generate either a triangle wave
     * or a trapezoid wave by modulating the PWM duty cycle.
     * N2HET1 program: Trapezoid_Wave.het
     */

    /* initialize N2HET1 based on HalCoGen settings */
    hetInit();

    /* Configure additional settings of N2HET1 based on the macros settings */
    configNHET1();

    while(1);

/* USER CODE END */
```

```
}
/* this function is to configure additional settings for the N2HET1 */
void configNHET1()
{

    /* calculate_ecmp_compare() will calculate the compare value as well as
     * the high resolution delay values to be loaded into the N2HET1 ECMP1
     * instruction for PWM Duty generation based on the input ECMP_MAX_DUTY */
    calculate_ecmp_compare();

    /* Set NHET1_PIN_PWM to output */
    hetREG1->DIR = 1 << NHET1_PIN_PWM;

    /* Change the LRPFC according to user input */
    hetREG1->PFR = (LRPFC << 8);

    /* Initialize the PWM period and duty cycle based on the defined parameters */
    hetRAM1->Instruction[pHET_L02_0].Control = (uint32)CNT_MAX_PERIOD |
                                               (hetRAM1-
>Instruction[pHET_L02_0].Control & 0xFD0000);
    hetRAM1->Instruction[pHET_REM_DUTY_0].Data = ecmp_compare_value;

    /* Configure the N2HET1 pin to output the PWM */
    hetRAM1->Instruction[pHET_L03_0].Control = (hetRAM1-
>Instruction[pHET_L03_0].Control & 0xFFFFE0FF) |
                                               (NHET1_PIN_PWM << 8);

    /* Configure the amount of delay to stay at the maximum duty cycle */
    hetRAM1->Instruction[12].Data = ((uint32)MAX_DUTY_TIMER << 7);
    hetRAM1->Instruction[13].Data = ((uint32)MAX_DUTY_TIMER << 7);

    /* Configure the amount of delay to stay at the minumum duty cycle */
    hetRAM1->Instruction[20].Data = ((uint32)MIN_DUTY_TIMER << 7);
    hetRAM1->Instruction[21].Data = ((uint32)MIN_DUTY_TIMER << 7);

    /* Duty cycle increment amount */
    if ( (uint32)DELTA_INCREMENT == 0){
        hetRAM1->Instruction[pHET_L11_0].Data = 1 << (7 - LRPFC);
    } else {
        hetRAM1->Instruction[pHET_L11_0].Data = (uint32)DELTA_INCREMENT << (7 - LRPFC);
    }

    /* Duty cycle decrement amount */
    if ((uint32)DELTA_DECREMENT == 0) {

        hetRAM1->Instruction[pHET_L17_0].Data = 1 << (7 - LRPFC);
    } else {
        hetRAM1->Instruction[pHET_L17_0].Data = (uint32)DELTA_DECREMENT << (7 - LRPFC);
    }

    /* Unlock the N2HET program. Initially after reset the N2HET program is locked */
    hetRAM1->Instruction[pHET_L00_0].Data = UNLOCK_KEY << 7;

}
```

The complete main() can be found in the project folder under Trapezoid_Wave/source/sys_main.c.

5.3 Project Directory Structure

This example project is named Async_NHET1_PWM_NHET2_Monitoring; Figure 6 shows the project directory structure. The two N2HET programs are contained under the HET code folder.
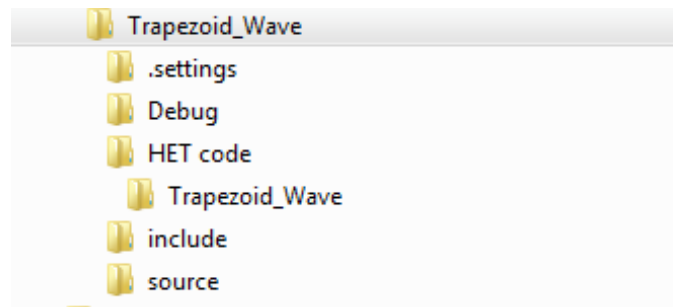

**Figure 6. Trapezoid_Wave Project Directory Structure**

6   Examples

By changing the parameters listed in Section 5.2, various forms of triangle and trapezoid waveforms can be rendered.

6.1   An Isosceles Triangle 1

In this example, a basic isosceles triangle (triangle with two equal sides) is created. With PWM_DUTY set to 100%, the output amplitude is equal to the rail voltage.

```
#define PWM_PERIOD        32
#define PWM_DUTY          100.0
#define LRPFC             7
#define DUTY_INCREMENT    0.0
#define DUTY_DECREMENT    0.0
#define MAX_DUTY_TIMER    0
#define MIN_DUTY_TIMER    0
```
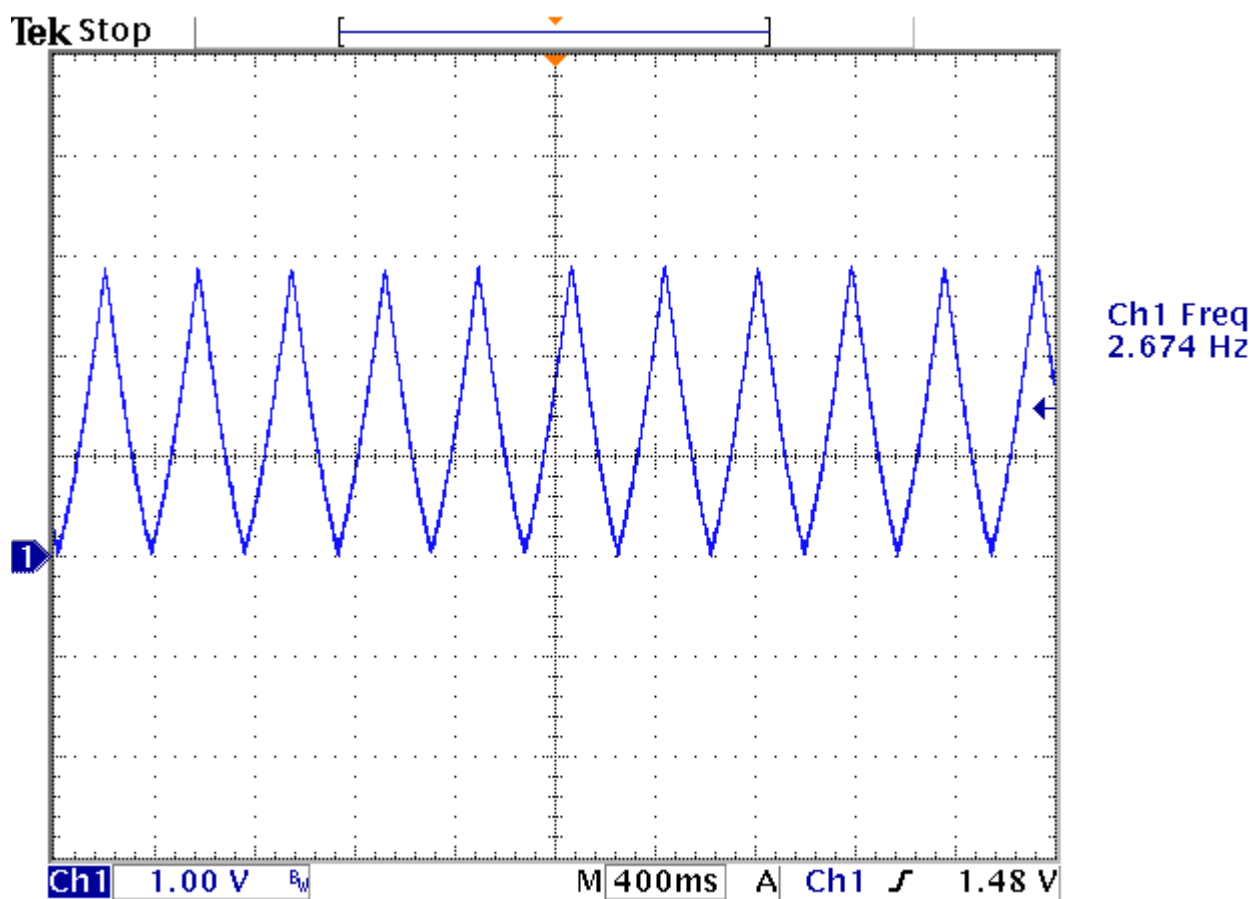

Figure 7. An Isosceles Triangle 1

6.2   An Isosceles Triangle 2

In this example, a basic isosceles triangle (triangle with two equal sides) is created. With PWM_DUTY set to 50%, the output amplitude is equal to half of the rail voltage (Vccio / 2). Reducing the maximum duty cycle to 50% also has the effect of reducing the duty cycle resolution by half. Reducing the resolution by half increases the output frequency by 2X.

```
#define PWM_PERIOD       32
#define PWM_DUTY         50.0
#define LRPFC            7
#define DUTY_INCREMENT   0.0
#define DUTY_DECREMENT   0.0
#define MAX_DUTY_TIMER   0
#define MIN_DUTY_TIMER   0
```
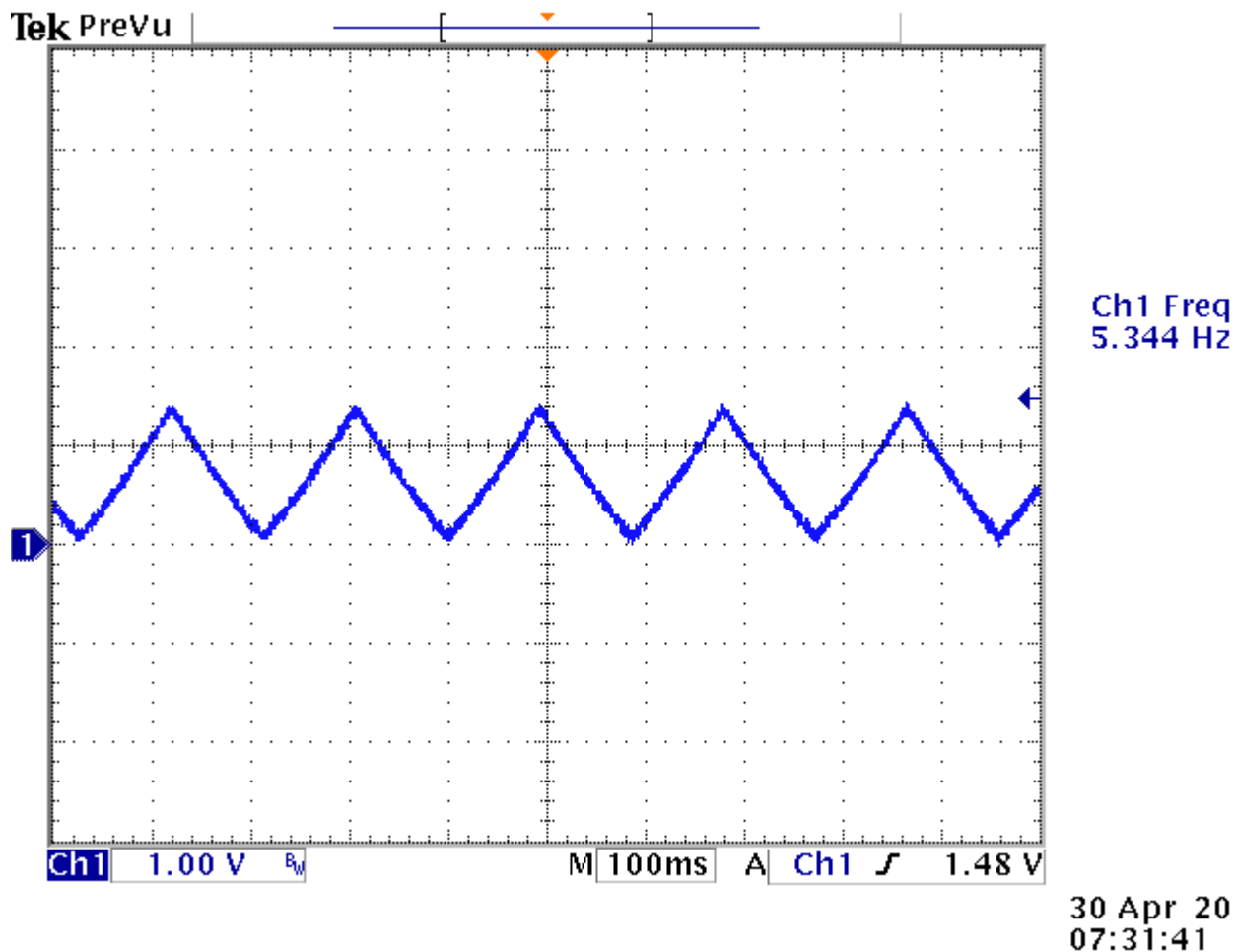

Figure 8. An Isosceles Triangle 2

6.3 A Scalene Triangle

In this example, a scalene triangle (triangle with no equal sides) is created.

```
#define PWM_PERIOD        32
#define PWM_DUTY          100.0
#define LRPFC             7
#define DUTY_INCREMENT    10.0
#define DUTY_DECREMENT    0.0
#define MAX_DUTY_TIMER    0
#define MIN_DUTY_TIMER    0
```
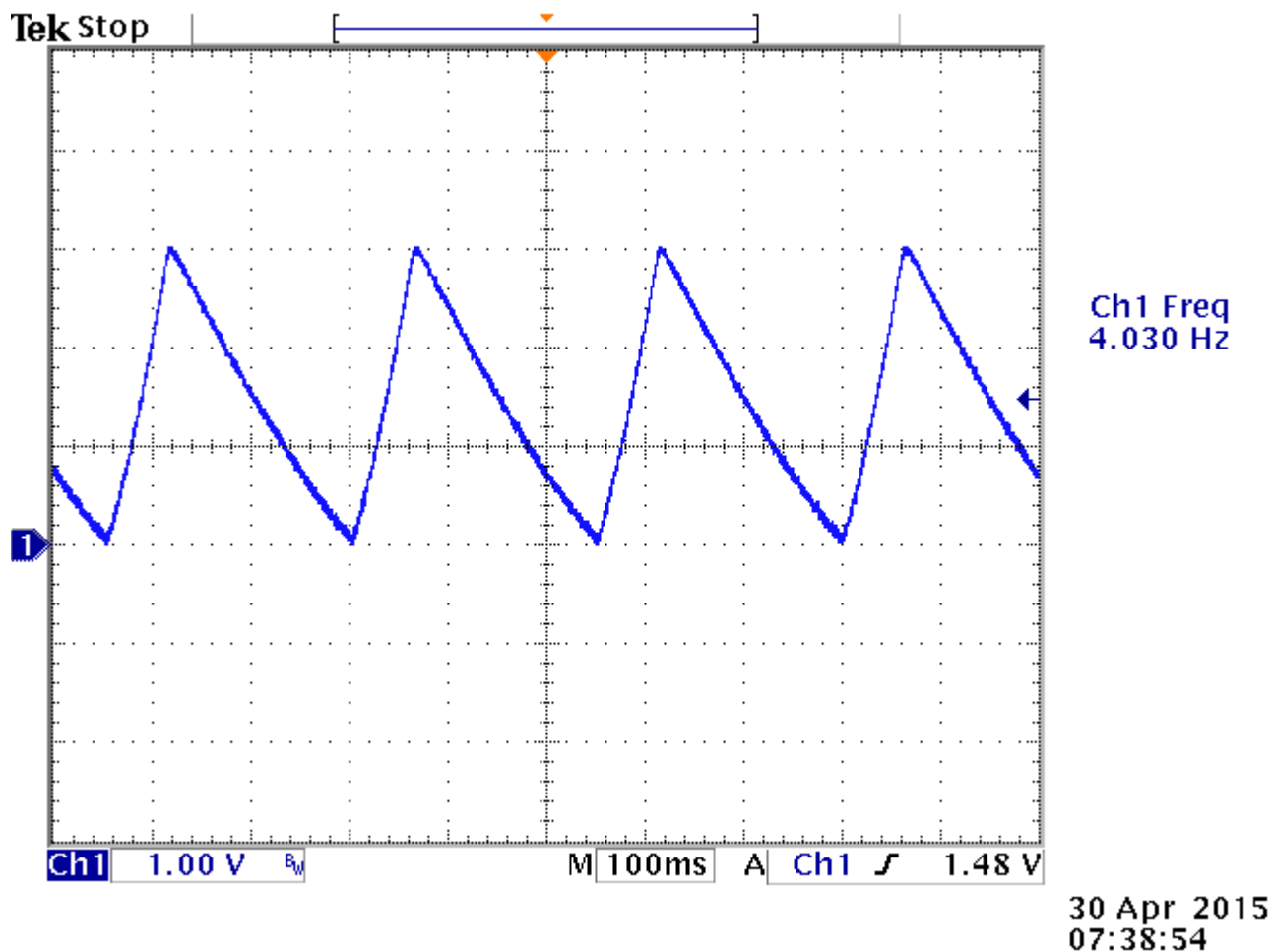

Figure 9. A Scalene Triangle

6.4   A Scalene Triangle

When DUTY_INCREMENT is increased to 100%, it is approaching a right triangle that is also a sawtooth wave.

```
#define PWM_PERIOD        32
#define PWM_DUTY          100.0
#define LRPFC             7
#define DUTY_INCREMENT    100.0
#define DUTY_DECREMENT    0.0
#define MAX_DUTY_TIMER    0
#define MIN_DUTY_TIMER    0
```
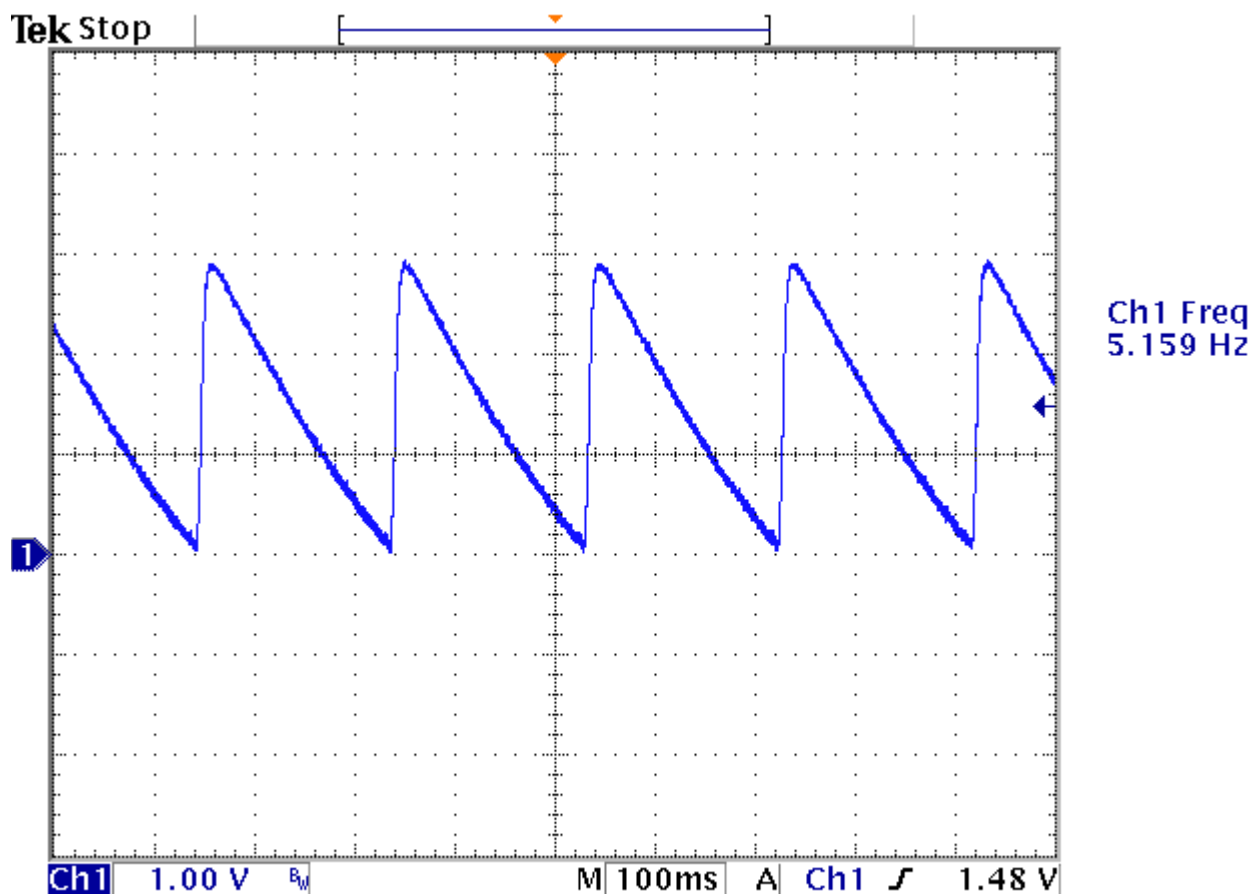

Figure 10. A Sawtooth Triangle

6.5   An Isosceles Trapezoid

In this example, a basic isosceles trapezoid (trapezoid with two equal sides) is created. With PWM_DUTY set to 100%, the output amplitude is equal to the rail voltage. Reducing the LRPFC to 5 is equivalent to generating 10-bit resolution on the ramp up and ramp down.

```
#define PWM_PERIOD        32
#define PWM_DUTY          100.0
#define LRPFC             5
#define DUTY_INCREMENT    0.0
#define DUTY_DECREMENT    0.0
#define MAX_DUTY_TIMER    2048
#define MIN_DUTY_TIMER    2048
```


Figure 11. An Isosceles Trapezoid

6.6 An Right Trapezoid

```
#define PWM_PERIOD        32
#define PWM_DUTY          100.0
#define LRPFC             5
#define DUTY_INCREMENT    100.0
#define DUTY_DECREMENT    0.0
#define MAX_DUTY_TIMER    4096
#define MIN_DUTY_TIMER    1024
```


Figure 12. A Right Trapezoid

6.7   An Scalene Trapezoid

```
#define PWM_PERIOD       32
#define PWM_DUTY         100.0
#define LRPFC            5
#define DUTY_INCREMENT   10.0
#define DUTY_DECREMENT   0.0
#define MAX_DUTY_TIMER   1024
#define MIN_DUTY_TIMER   2048
```
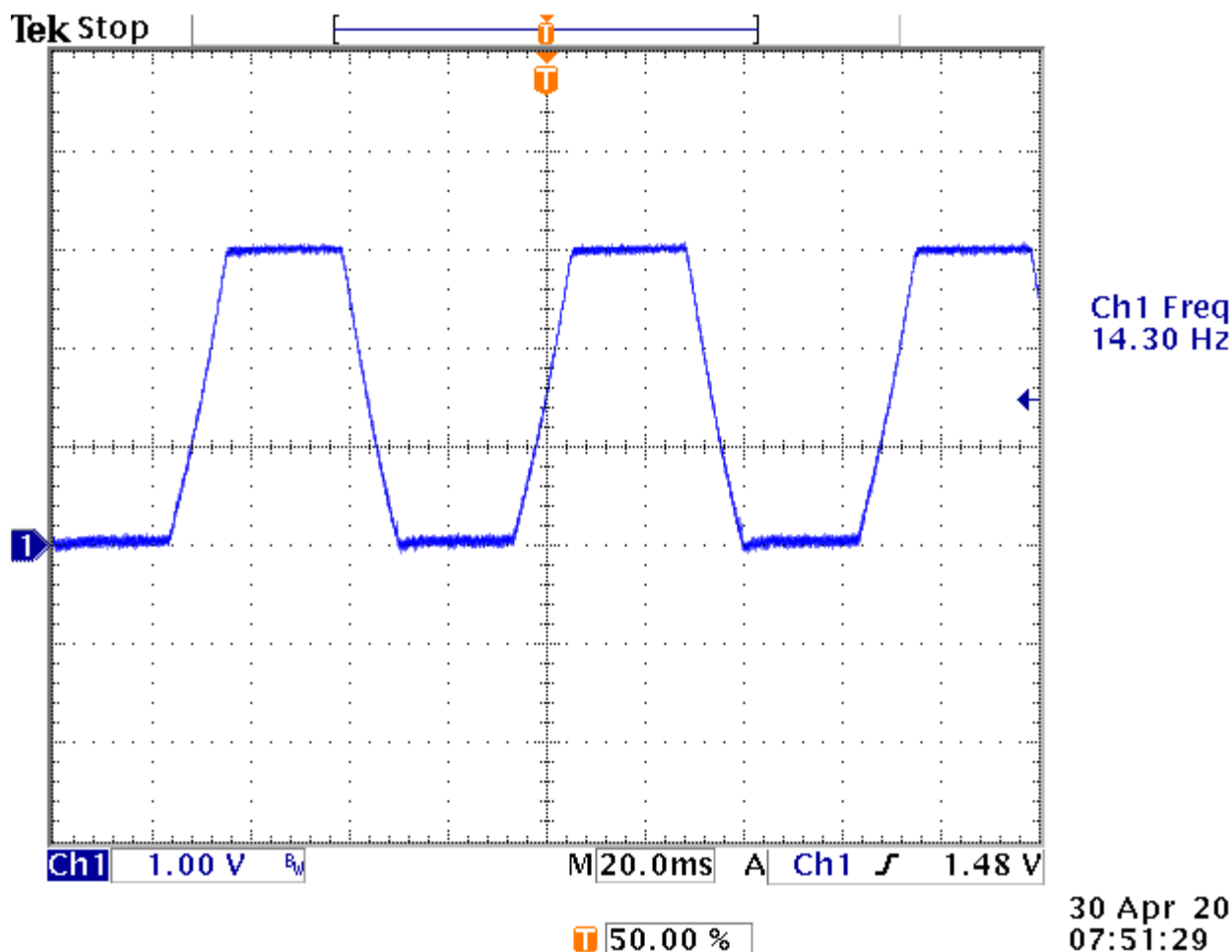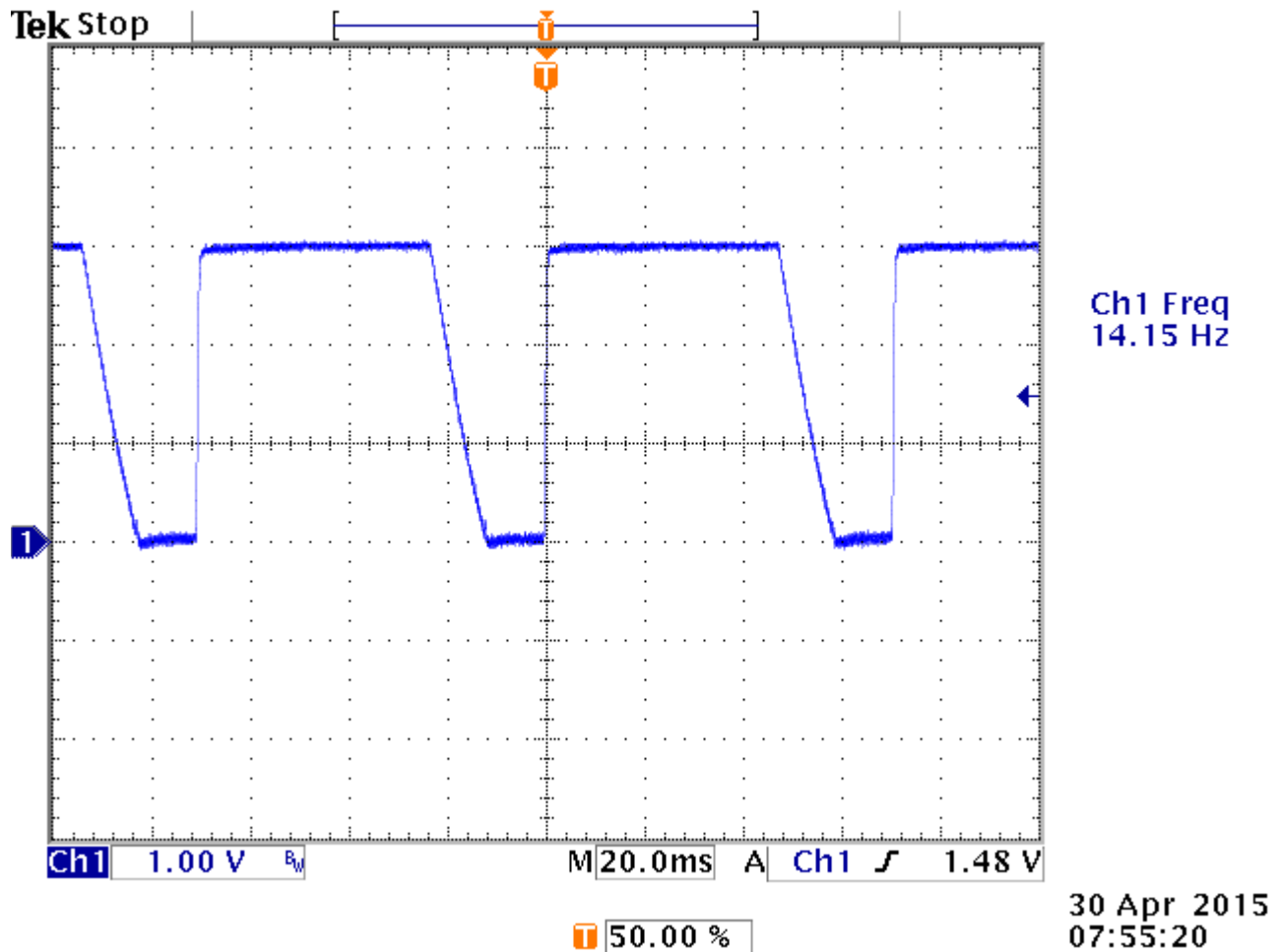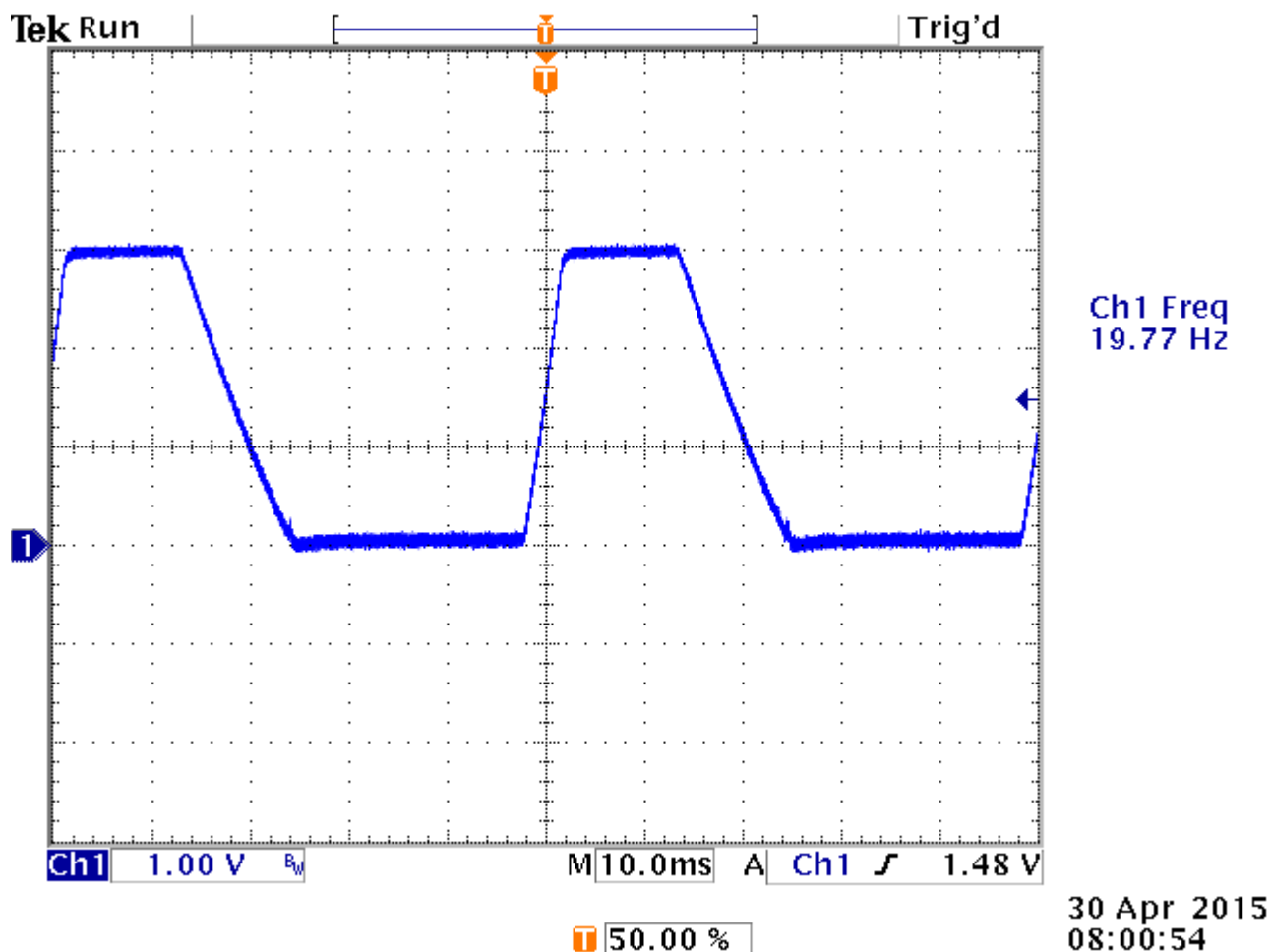

Figure 13. A Scalene Trapezoid

7   References

- *HET Integrated Development Environment User's Guide* (SPNU483)
- *NHET Getting Started* (SPRABA0B)
- *Enhanced High-End Timer (NHET) Assembler User's Guide* (SPNU490)
- *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		