Application Report
SPNA178–April 2015

Monitoring PWM Using N2HET

Charles Tsai

ABSTRACT

This application report illustrates two examples to monitor an incoming PWM signal using the versatile programmable high-end timer (N2HET). Both examples can be run in either the Hercules™ HDK or the LaunchPad board. 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 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/spna178.

Contents

List of Figures

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.

2 Example 1 Description

The goal of this example is to monitor an incoming PWM pulse. N2HET1 will generate a PWM test signal that starts at 0% duty cycle and slowly changes the duty cycle. N2HET2 will monitor the generated signal and indicate an error, duty cycle outside of the specified range, by triggering an interrupt and setting a pin.

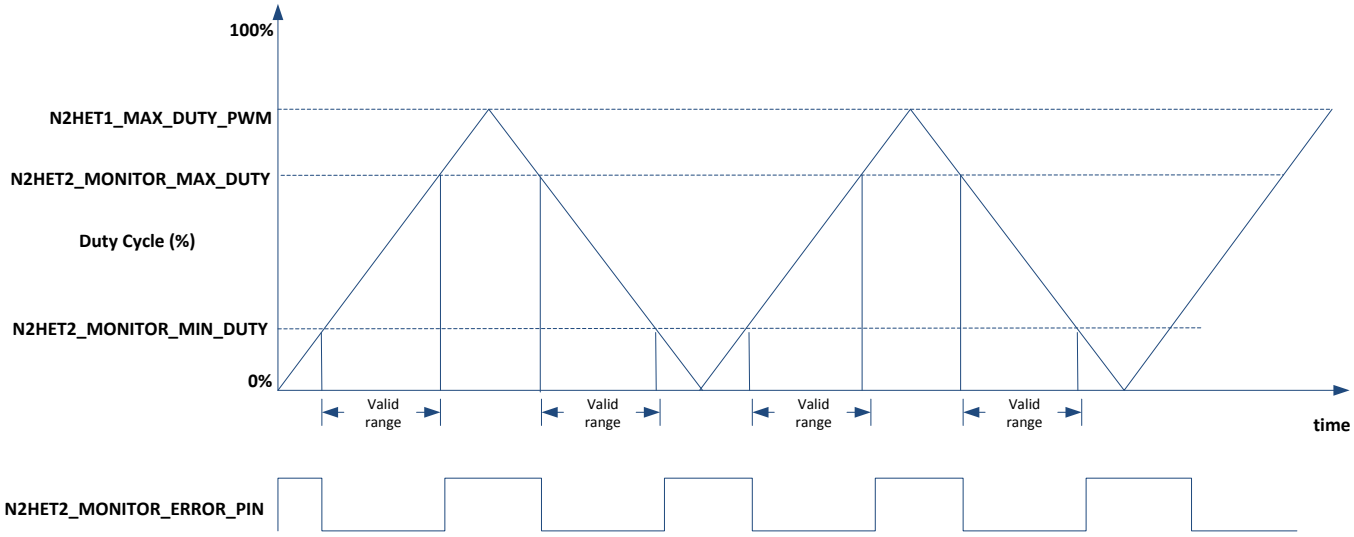


Figure 1. N2HET1 Duty cycle vs. Time

As illustrated in Figure 1, you can specify how the PWM is to be generated in terms of the maximum duty cycle using N2HET1. The N2HET1 is capable of sweeping from a minimum duty cycle of 0% until it reaches the maximum duty cycle as specified in N2HET1_MAX_DUTY_PWM. The amount of increments and decrements in the duty cycle and also the increment and decrement rate can be specified by the host CPU. The N2HET1_MAX_DUTY_PWM is a changeable macro by the host CPU. The N2HET2_MONITOR_MAX_DUTY and N2HET2_MONITOR_MIN_DUTY defines the range that the N2HET2 considers as a valid range. If the PWM generated by the N2HET1 is outside of this valid range, then an interrupt is generated to the host CPU. The N2HET_MONITOR_ERROR_PIN is an user-programmable N2HET2 functional pin that can be selected to output the monitor status. When the PWM is outside the valid range, the N2HET_MONITOR_ERROR_PIN is set.

Figure 2 shows a scope shot of the N2HET2_MONITOR_ERROR_PIN on top and the triangle wave generated by varying the duty cycle of N2HET1 at the bottom. The scope shot was taken with the minimum duty cycle set to 20% and the maximum duty cycle set to 80% for the valid range. The PWM generated by N2HET1 is capable of varying its duty cycle from 0% to 100% with the PWM frequency of 20KHz.

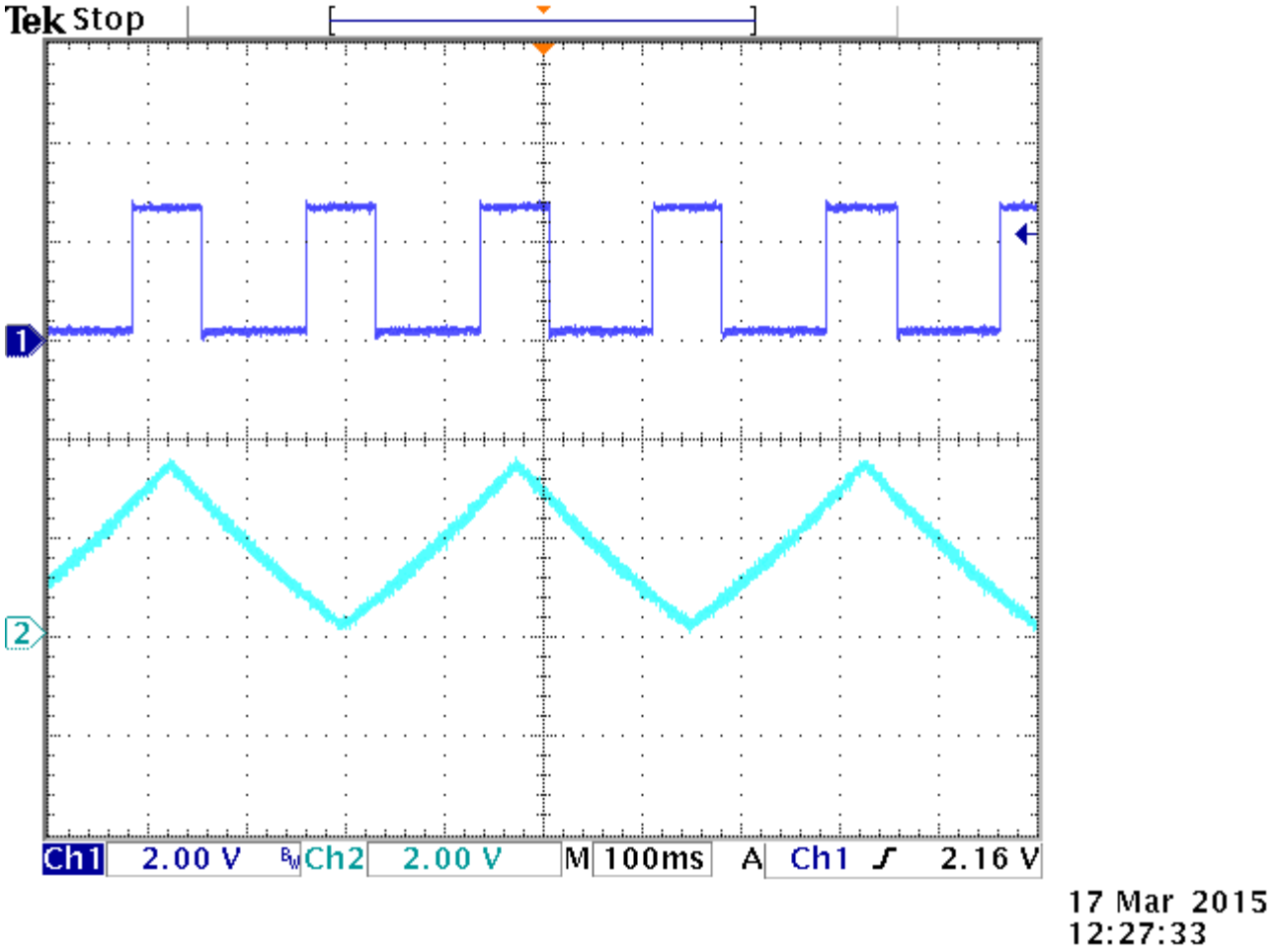


Figure 2. N2HET_MONITOR_ERROR_PIN and Triangle Wave PWM

To render the triangle wave on the scope, the generated PWM will run through a low pass filter as shown in Figure 3. Note that the low pass filter is mainly to display the varying duty cycle PWM on the scope for easier visualization. The example does not need this low pass filter to work.

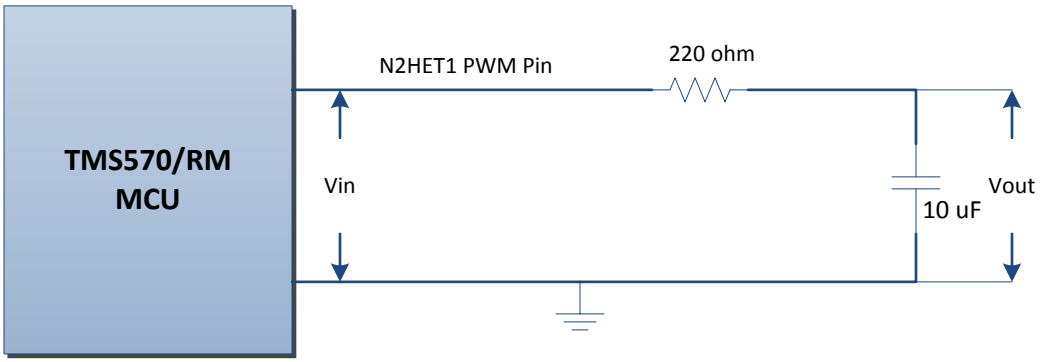


Figure 3. Low Pass Filter

2.1 Example 1 N2HET1 Triangle Wave Generation Flow Chart

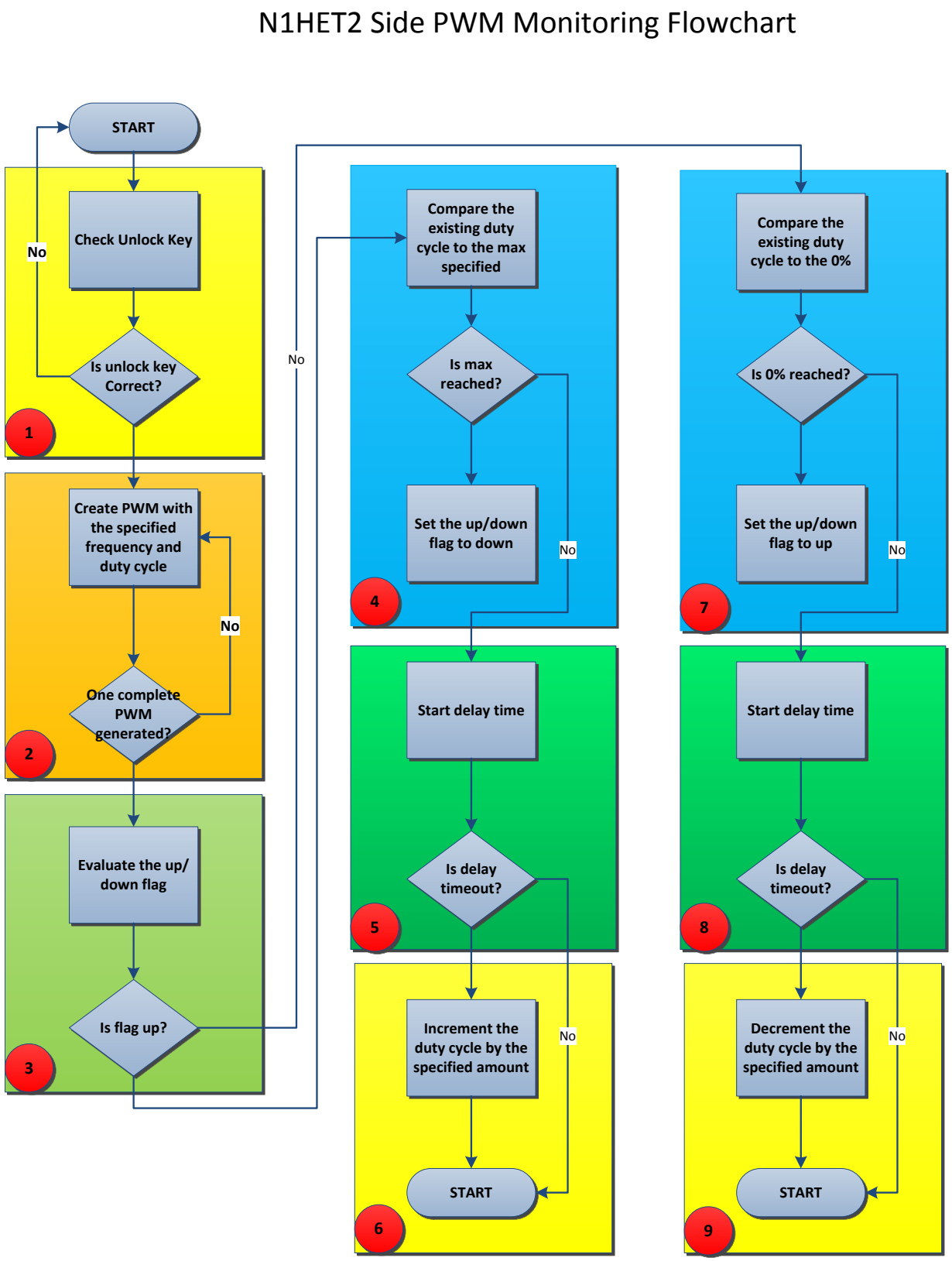


Figure 4. N2HET1 Triangle Wave Flow Chart

1. The N2HET1 is put into a self-loop until a proper unlock key is written to the N2HET1. While the N2HET1 is 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 with be self modified by the N2HET1 itself either in increasing order or decreasing order. Do not proceed to the next step until one full PWM period is generated.
3. Evaluate the up/down flag to determine whether or not the duty cycle should increase or decrease. The flag will be initialized to 0 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. If the maximum is reached then set the up/down flag to 1.
5. Start a programmable increment delay. The delay is used to wait before incrementing to the next duty cycle. The amount of increment delay is programmable by the host CPU.
6. After the delay expires, increment to the next duty cycle by the amount that is programmable by you.
7. Compare the existing duty cycle to the 0% duty cycle. If the 0% is reached then set the up/down flag to 0.
8. Start a programmable decrement delay. The delay is used to wait before decrementing to the next duty cycle. Note that the decrement delay can be different from increment delay.
9. After the delay expires, decrement to the next duty cycle by the amount that is programmable by you. Note that the amount to decrement can be different from the amount to increment.

2.2 Example 1 N2HET1 Triangle Wave Program

The example N2HET1 program code is illustrated below. Directives using .equ are used to define some of the parameters to control the program. These parameters can be changed by you at assembly time. By default, these parameters have initial values that are small for quick simulation using HET IDE. The host CPU can change these parameters in the host side application.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This example code is to be loaded to N2HET1 to generate a triangle waveform using varying
; duty cycle ramping from 0% duty to a programmable maximum duty cycle and then ramp down
; to 0% again.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; PWM frequency to be generated
PWM_PERIOD             .equ 3

; The pin number that will output the PWM signal
PWM_PIN_NUM            .equ 9

; The initial maximum duty cycle (compare value) to be generated.
INIT_COMPARE           .equ 2

; delay amount to increment from one duty cycle to the next duty cycle.
INCREMEMT_DELAY        .equ 1

; delay amount to decrement from one duty cycle to the next duty cycle.
DECREMEMT_DELAY        .equ 1

; amount of increment
DUTY_INCREMEMT         .equ 0

; amount of decrement
DUTY_DECREMEMT         .equ 0

; amount of increment
DUTY_INCREMEMT_HR      .equ 1

; amount of decrement
DUTY_DECREMEMT_HR      .equ 1

; 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 temporaray 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 determines 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=ON,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. We want to create a
; triangle waveform. The PWM will first increase the duty cycle until it
; reaches the specified maximum duty cycle before it starts to decrease the
; duty. The up/down flag is used to create two different paths in the flow
; to alternate between 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,smode=LSR,scount=0,data=0};

; If the substraction 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 then we have reached the max duty cycle and we will change the
; up/down flag to down position.
L10   BR { next=L14,cond_addr=L11,event=NC};

; Insert delay before changing to the next duty cycle
L11   DJZ { next=L00,cond_addr=L12,reg=NONE,data=INCREMEMT_DELAY};

; Add specified amount to the existing compare value (duty cycle). This
; value is also changeable by CPU
L12   ADD { src1=R,src2=IMM,dest=S,rdest=REM,remote=L03,data=DUTY_INCREMEMT,
            hr_data=DUTY_INCREMEMT_HR};

; Reset the increment delay to the specified amount.
L13   MOV32 { next=L15,remote=L11,type=IMTOREG&REM,reg=NONE,data=INCREMEMT_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 value is
; also changeable by CPU.
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 A which contains the
; subtraction result.
L18   BR { next=L19,cond_addr=L22,event=ZN};

; Insert the delay before decreasing to the next duty cycle.
L19   DJZ { next=L00,cond_addr=L20,reg=NONE,data=DECREMEMT_DELAY};

; Move the subtraction result to the ECMP DF as the new duty cycle
L20   MOV32 { next=L21,remote=L03,type=REGTOREM,reg=S};

; Reset the decrement delay to the specified amount
L21   MOV32 { next=L00,remote=L19,type=IMTOREG&REM,reg=NONE,data=DECREMEMT_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=0};
DUMMY   BR { next=DUMMY,cond_addr=DUMMY,event=NOCOND,irq=OFF};
```

2.3 Example 1 N2HET2 Monitoring Flow Chart

Figure 5 illustrates a flowchart that tries to mimic the corresponding N2HET2 code sequence for monitoring the PWM. The flow chart is color coded to distinguish five major steps.

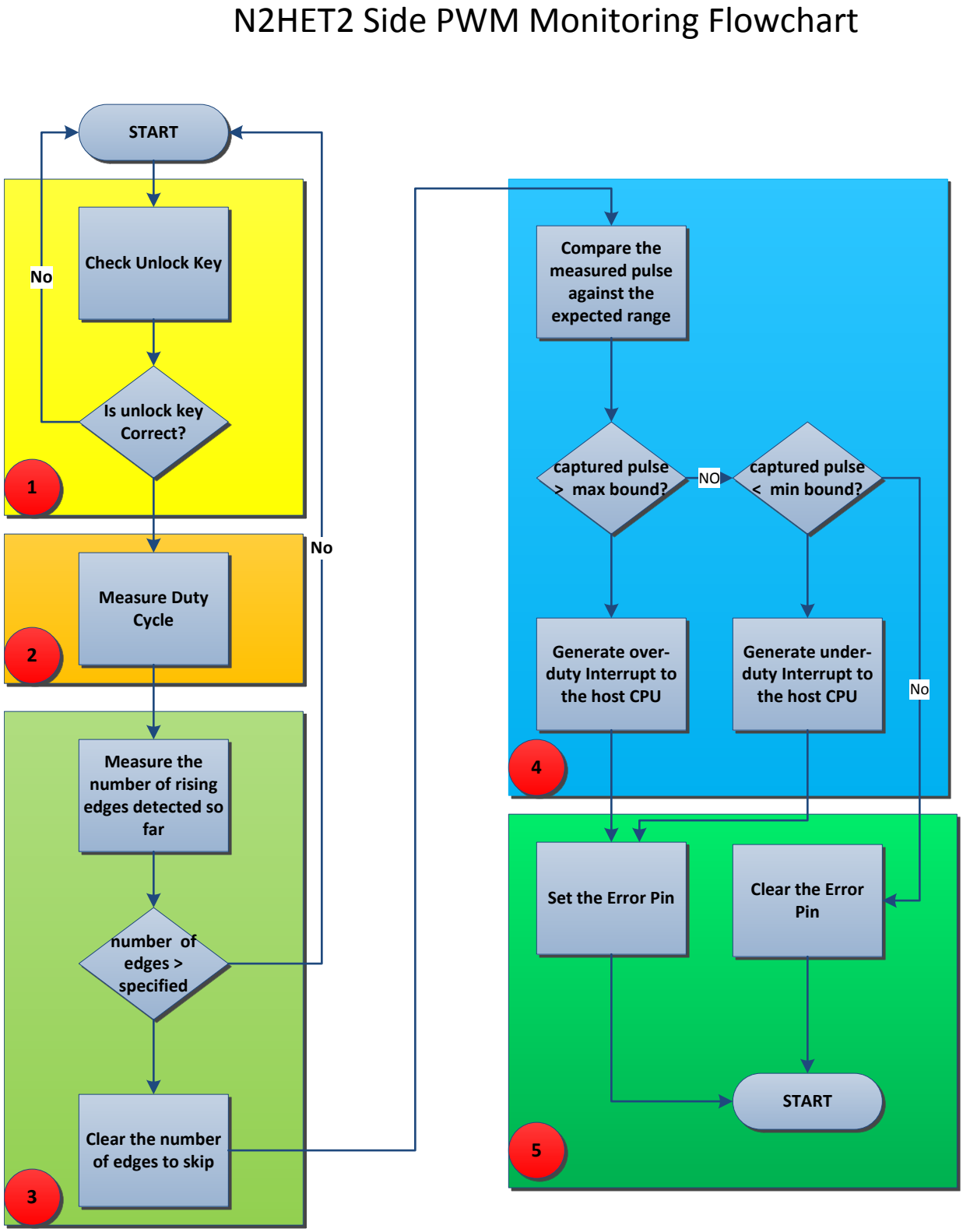


Figure 5. N2HET2 Monitoring Flow Chart

In step 1, the N2HET2 is put into a self-loop until a proper unlock key is written to the N2HET2. While the N2HET2 is locked, the host CPU can setup various parameters for the N2HET2 program. Even though the same intent can be achieved by globally turning off the N2HET2 via the HETGCR.TO bit there are other benefits to this unlock sequence. For example, you might write his N2HET code such that when certain event happens, it will interrupt the CPU and at the same time lock the N2HET by writing a lock key without CPU intervention. At this time the N2HET is frozen in a state for host CPU's examination until which time the CPU unlocks N2HET again. This step is not mandatory in each N2HET program especially if the code size is a limitation.

In step 2, the heart of the monitoring is the duty cycle measurement of the incoming PWM. Pusle count (PCNT) instruction is the ideal instruction to accomplish this task. PCNT can be configured to measure either a RISE2RISE or FALL2FALL period or to measure either a RISE2FALL (high phase) pulse or FALL2RISE (low phase) pulse. The measurement can be done in either loop resolution or high resolution. In high resolution, the accuracy of the measurement is at the HR granularity. One HR clock can be equal to the input functional clock (VCLK2) to the N2HET module. A 100 MHz VCLK2 will allow a measurement accuracy down to 10 ns.

The purpose of the step 3 is to skip the initial edges. Immediately after the N2HET2 is turned on and unlocked, the N2HET2 starts execution. The N2HET2 execution is asynchronous to the edges of the incoming PWM. If the first edge comes right after the N2HET is unlocked, the N2HET can measure incorrect period or pulse. Therefore, the first captured period or pulse is normally inaccurate. Normally, the host CPU needs to discard the first captured pulse or period. Here in step 3, it alleviates the CPU from determining when and how many edges to skip for an inaccurate capture with a built-in code sequence to skip an user-programmable number of edges. Note that immediately after the number of edges are skipped, it changes the number of edges to skip to zero. The reason is that the virtual counter used to count the number of edges will eventually overflow and restart from zero. Only skip edges as needed for the first time after the N2HET is unlocked, not after every time the counter overflows.

The step 4 performs various arithmetic operations to determine if the incoming PWM duty cycle is within the predefined valid range. If the measured duty cycle is outside the valid range then an interrupt is generated.

The step 5 is optional. In this example, an user-programmable N2HET2 pin is also asserted to indicate the monitor status. This allows you to view the N2HET1 PWM and the N2HET2 monitor status pin side-by-side on the scope.

2.4 Example 1 N2HET2 Monitoring Program

The example N2HET2 program code is illustrated below. Directives using .equ are used to define some of the parameters to control the program. These parameters can be changed by you at assembly time. By default these parameters have initial values that are small for quick simulation using HET IDE. The host CPU can change these parameters in the host side application.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This example code is to be loaded to N2HET2 to monitor the PWM waveform generated
; by another source. The N2HET2 expects a range of valid duty cycles (specified in MAX_DUTY
; and MIN_DUTY) to be monitored. If the PWM waveform is out of range then the N2HET2
; will generate an interrupt to the host CPU as well as set the specified pin high.
; N2HET2 will reset the pin if the monitored duty cycles are within the range between
; MIN_DUTY and MAX_DUTY.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; this is the pin number to monitor the PWM duty cycle
PULSE_MONITOR_PIN_NUM  .equ 4
; this is the maximum duty cycle of a valid range
MAX_DUTY               .equ 6
; this is the minimum duty cycle of a valid range
MIN_DUTY               .equ 3
; this is the number of rising edges on the PWM input to skip
SKIP_EDGES             .equ 3
; this is the pin number to assert/de-assert error
ERROR_PIN_NUM          .equ 0
; this is the unlock key for code execution
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 NHET program. First the MOV32 instruction moves the
; initial value to a temporaray register T
L00   MOV32 { remote=DUMMY,type=IMTOREG,reg=T,data=0x5,hr_data=0};

; Compare the register T value with the key to unlock NHET. 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 NHET
L01   ECMP { next=L00,hr_lr=LOW,cond_addr=L02,pin=0,reg=T,data=UNLOCK_KEY,hr_data=0};

; Use PCNT to measure the high phase of the PWM using type=RISE2FALL
; Note that the pin selected to measure pulse is pin=PULSE_MONITOR_PIN_NUM which
; is changeable by the host CPU.
L02   PCNT { hr_lr=HIGH,type=RISE2FALL,pin=PULSE_MONITOR_PIN_NUM};

; Use ECNT to first detect the number of rising edges. The purpose here is to throw
; away the measurements of the first few edges becasue the period/pulse measurement
; from PCNT will not be accurate for the first edge.
L03   ECNT { cond_addr=L05,pin=PULSE_MONITOR_PIN_NUM,event=RISE,reg=R,data=0};

; Compare with SKIP_EDGES. Only when the number of edges is greater or equal to SKIP_EDGES
; will we take the pulse measurement from PCNTs
L04   MCMP {
next=L00,hr_lr=LOW,cond_addr=L05,pin=0,order=REG_GE_DATA,reg=R,data=SKIP_EDGES,hr_data=0};

; Reset the compare value to 0 so that when ECNT counter overflows it will not need
; to skip edges again.
L05   MOV32 { remote=L04,type=IMTOREG&REM,reg=NONE,data=0x0,hr_data=0};

; Move the pulse captured by PCNT RISE2FALL to a temp register T
L06   MOV32 { remote=L02,type=REMTOREG,reg=T};

; Subtract the captured value in T by the max duty stored in REM_MAX_DUTY
; Note that the expected max duty stored in the REM_MAX_DUTY
; is actually the expected pulse plus one. For example, if the expected pulse is
; 10 then REM_PULSE will store 11. The PCNT can measure the pulse with accuracy
; of +/-1 from the expected meaning it may measure within the bound of 9 to 11.
; If we set expected value to 11 then it eases our comparision so that the measured
; value should never be greater than REM_MAX_DUTY. The largest difference between
; the measured value against the expected value will be between 9 and 11. To
; tolerate this difference of 2 we first do the subtraction and then perform a right
; shift of the result by two bits.
L07   SUB { src1=REM,src2=T,dest=IMM,rdest=NONE,remote=REM_MAX_DUTY,smode=LSR,scount=2,data=0};

; If the captured pulse is larger than max duty then enable interrupt to the CPU
; and set the error pin
L08   BR { next=L09,cond_addr=L11,event=LT,irq=ON};

; Subtract the min duty cycle boundary from the current captured pulse width. If
; the the min duty cycle is greater than the current pulse then it means the
; current pulse is smaller than the min duty which is out of bound.
L09   SUB { src1=REM,src2=T,dest=IMM,rdest=NONE,remote=REM_MIN_DUTY,smode=LSR,scount=2,data=0};

; Generate an interrupt and set the pin high if the current captured pulse width is
; out of range.
L10   BR { next=L13,cond_addr=L11,event=GT,irq=ON};

; Move 0 to register A. The register A will be used by the subsequent ECMP instruction
; to compare with. The purpose is for the ECMP to have a compare match all the time.
L11   MOV32 { remote=DUMMY,type=IMTOREG,reg=A,data=0};

; Depending on the result of the substraction. If the subtraction result indicates that
; PWM to be monitored exceeds the expected amount, we will assert the ERROR_PIN_NUM.
L12   ECMP {
next=L00,hr_lr=LOW,en_pin_action=ON,cond_addr=L00,pin=ERROR_PIN_NUM,action=SET,reg=A,data=0};
```

```
; If the subtraction result indicates that the PWM to be monitored does not exceed
; the epxected amount, we will clear the ERROR_PIN_NUM using the below ECMP instruction.
L13   MOV32 { remote=DUMMY,type=IMTOREG,reg=A,data=0};

; Clear the ERROR_PIN_NUM pin.
L14   ECMP {
next=L00,hr_lr=LOW,en_pin_action=ON,cond_addr=L00,pin=ERROR_PIN_NUM,action=CLEAR,reg=A,data=0};

; The max duty is stored in the data field in the below dummy ECMP instruction. This
; instruction is never executed.
REM_MAX_DUTY   ECMP {
next=REM_MAX_DUTY,cond_addr=REM_MAX_DUTY,pin=0,reg=A,data=MAX_DUTY,hr_data=0};

; The min duty is stored in the data field in the below dummy ECMP instruction. This
; instruction is never executed.
REM_MIN_DUTY   ECMP {
next=REM_MIN_DUTY,cond_addr=REM_MIN_DUTY,pin=0,reg=A,data=MIN_DUTY,hr_data=0};

; This is a dummy instruction. It is more a HET IDE issue where in L00 if a
; remote address is not given it is throwing an error even though that in
; L00 there is no need to move data from/to the remote field.
DUMMY   BR { next=DUMMY,cond_addr=DUMMY,event=NOCOND,irq=OFF};
```

2.4.1 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 N2HET2 instance:

```
hetp -n1 -hc32 NHET2_PWM_Range_Monitor.het
```

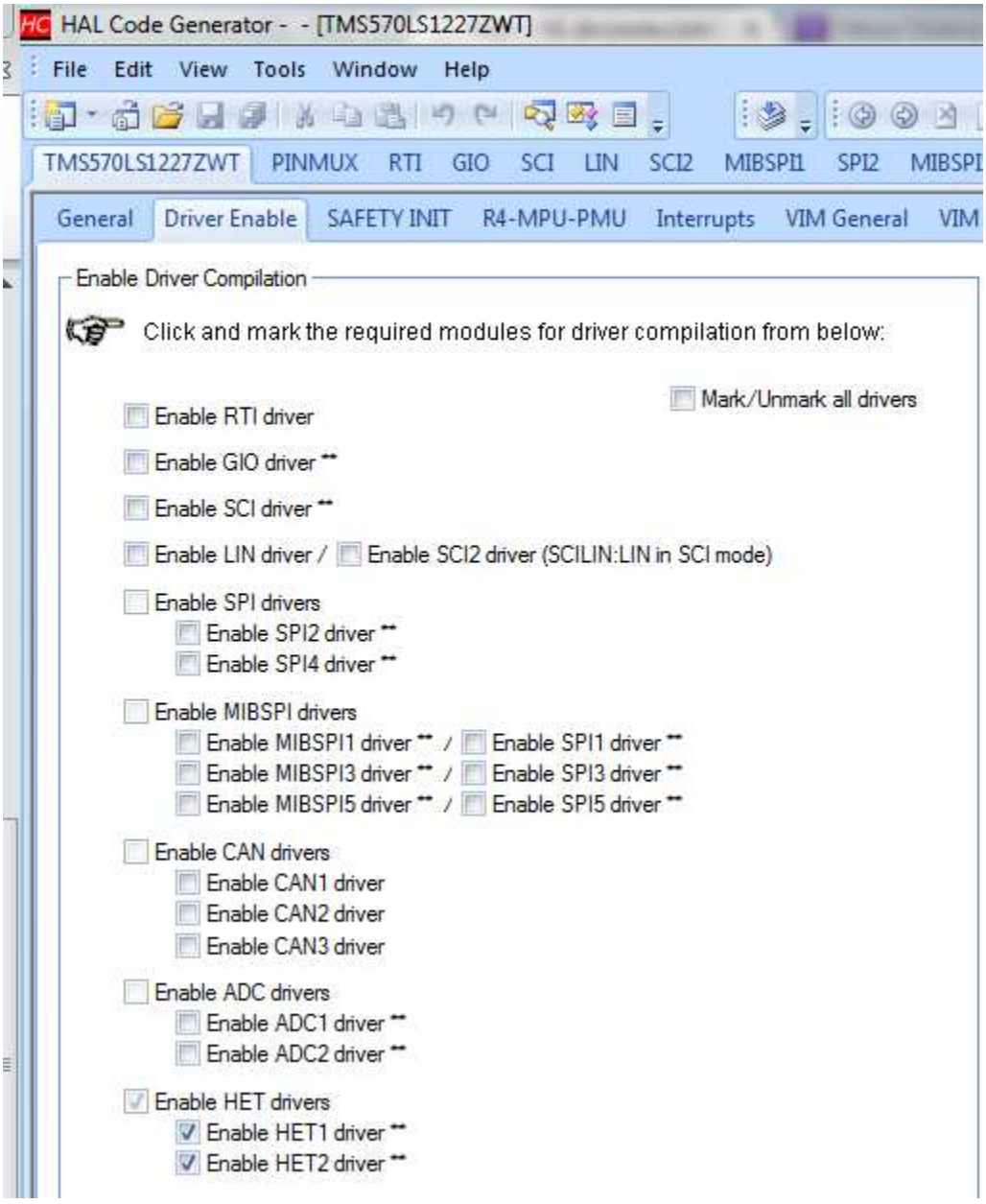
Note the argument -n1 that is passed to the assembler. The -n1 argument specifies the "x"-th N2HET module in the device. The valid value of x is 0-9. Normal convention is to use -n0 for N2HET1 in the device and -n1 for N2HET2. The -hc32 argument produces C header file (.h) and source file (.c) for the Texas Instruments TI's C compiler. Specifying the -nx argument will allow the assembler to produce unique header and source files per N2HET instance. A different N2HET program that generates the triangle wave should be assembled and loaded into the N2HET1 using the command below:

```
hetp -n0 -hc32 Async_PWM_Triangle_Wave.het
```

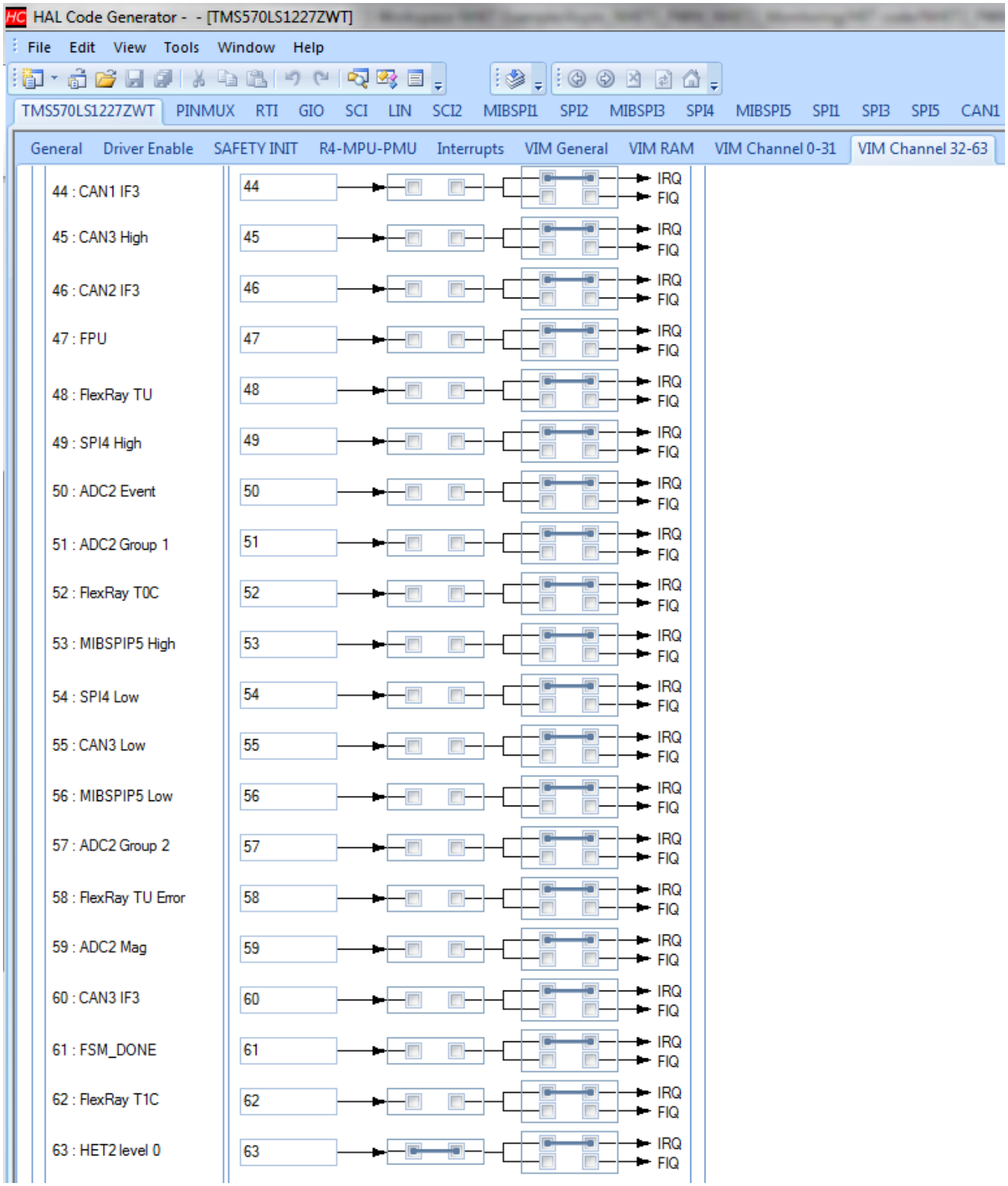
2.5 Example 1 HalCogen Setup

This example utilizes the HalCogen tool to configure the device. The target device selected in the HalCoGen for this example is the TMS570LS1227. It can be easily ported to other devices by following the steps below:

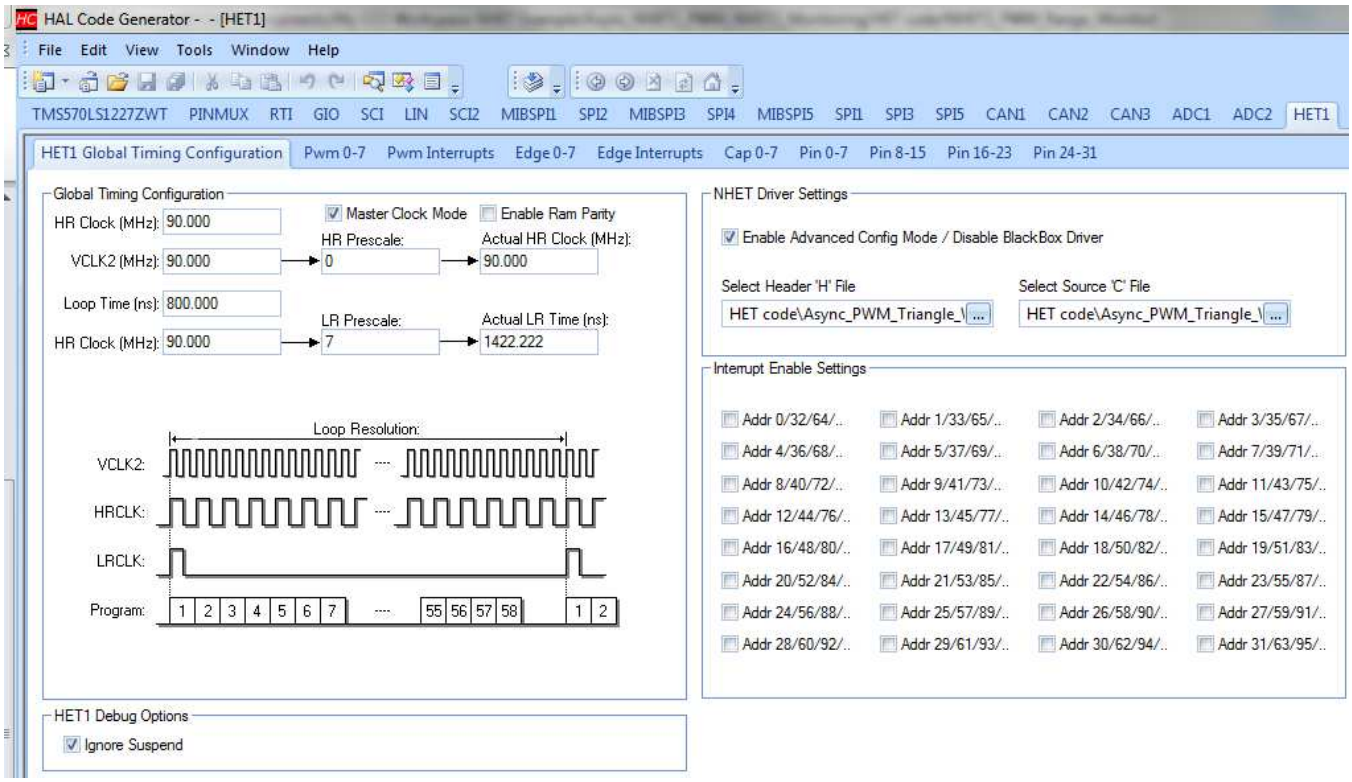
1. Create a new project: File → New → Project.
2. Enable only the N2HET drivers and disable the rest.



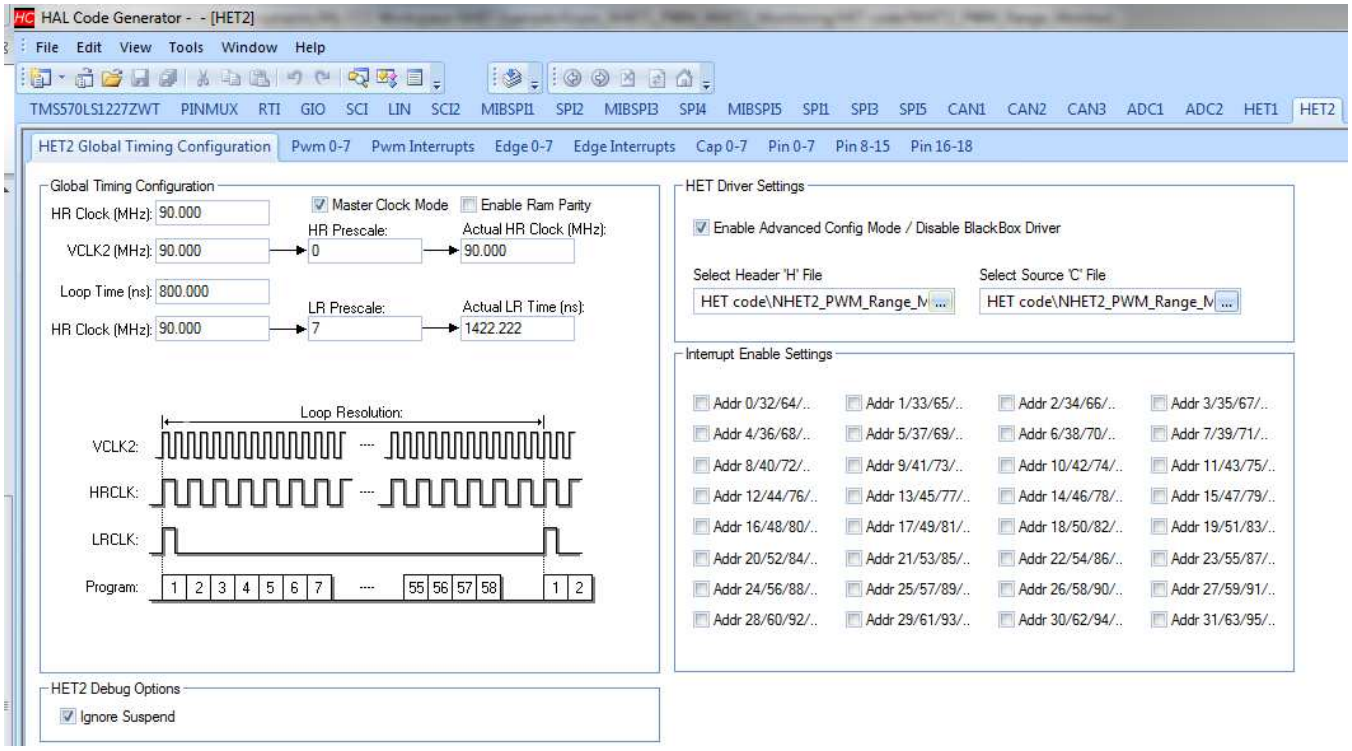
3. Enable N2HET2 Interrupt. N2HET2 is mapped to VIM Channel 63.



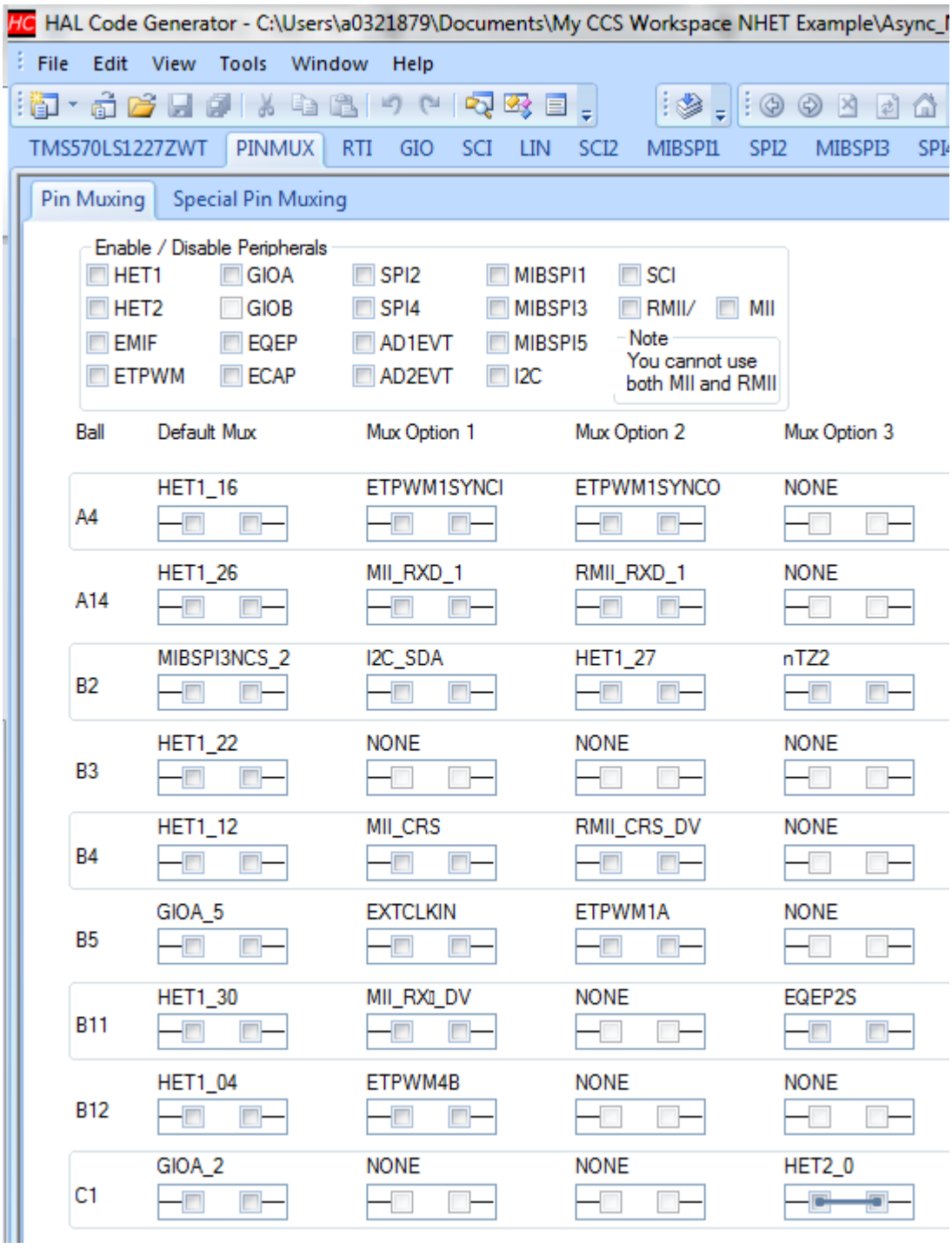
4. Configure N2HET1. Make sure to enable the checkbox for "Enable Advanced Config Mode/Disable BlackBox Driver" and provide the header file (Async_PWM_Triangle_Wave.h) and source file (Async_PWM_Triangle_Wave.c) generated in Section 2.4.1. This step will bypass the default blackbox N2HET code provided by HalCoGen so that the custom N2HET code can be loaded to the N2HET module.

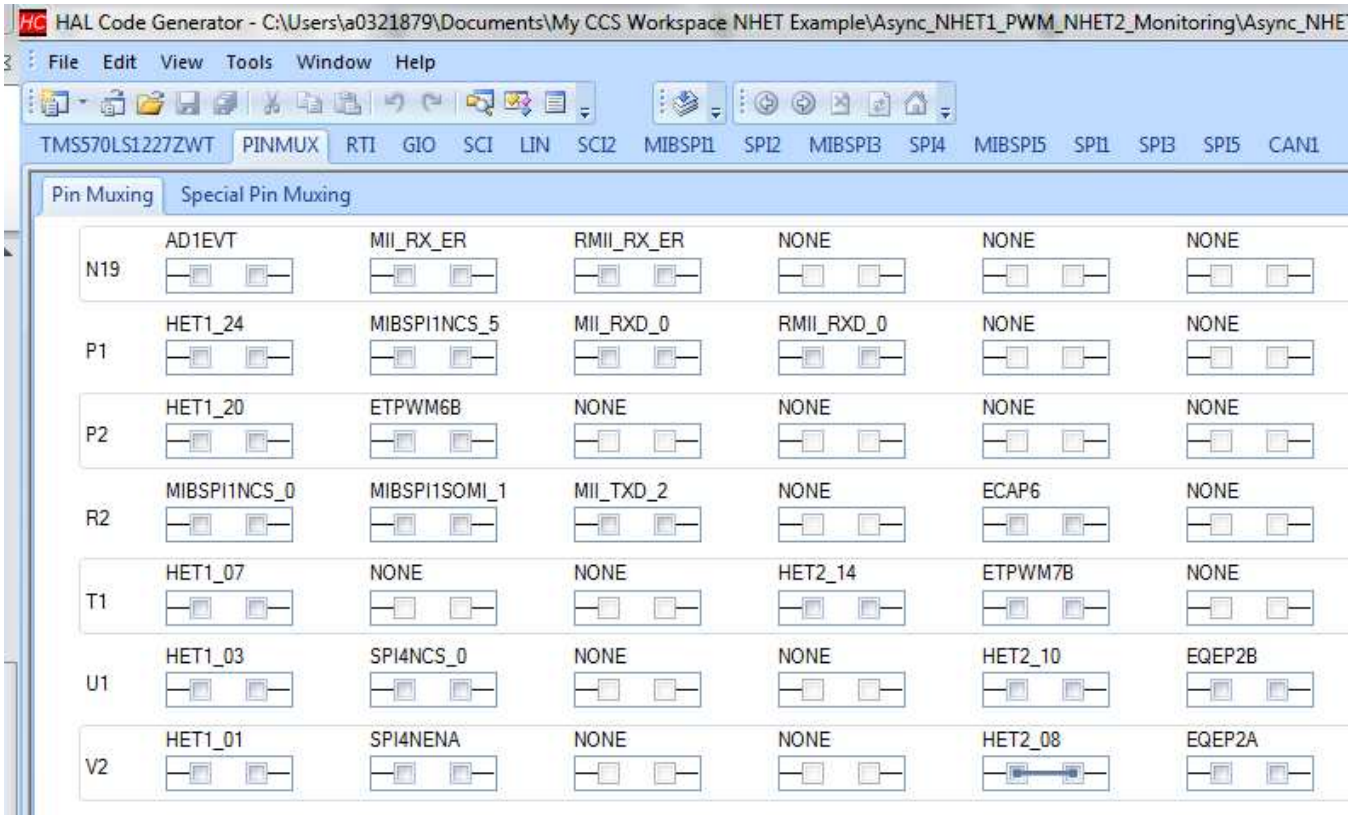


5. Configure N2HET2. The custom code to be executed by the N2HET2 is the PWM Monitoring with the header file (NHET2_PWM_Range_Monitor.h) and source file (NHET2_PWM_Range_Monitor.c).



6. Configure the PINMUX to bring the N2HET2[0] functional pin to the device C1 terminal on the HDK board. The N2HET2[0] is multiplexed with the GIOA[2]. The N2HET2[0] is used as the MONITOR_ERROR_PIN. This pin is set high when the PWM is out of the valid range. To probe the N2HET2[0] on the HDK, put the probe point at the GIOA[2] on the J11 expansion connector. Note that if a different N2HET2 pin is to be used for MONITOR_ERROR_PIN on the HDK then the PINMUX needs to be configured accordingly. For the Launchpad, the default pin selected for MONITOR_ERROR_PIN is the N2HET2[8]. N2HET2[8] is multiplexed with N2HET1[1]. The N2HET1[1] is brought out on the J9 expansion connector on the LaunchPad. Also note that on the application side the macro #define NHET2_MONITOR_ERROR_PIN PIN_HET_0 is selecting N2HET2[0] by default. This setting will work if working with the HDK. You will need to change to #define NHET2_MONITOR_ERROR_PIN PIN_HET_8 and rebuild the project in order to run the example on the LaunchPad.





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

2.6 Example 1 Host CPU Side Application Setup

On the host CPU side, its main task is to first initialize the device and configure both the N2HET1 and N2HET2 modules. Various macros are utilized to configure both the N2HET modules. Below are the list of changeable macros.

```
/*****************************************************************************/
/* Macros for configuring N2HET2                                             */
/*****************************************************************************/

/* Pin number in N2HET2 to monitor the PWM. */
#define NHET2_PIN_MONITOR PIN_HET_16

/* Max Duty cycle threshold beyond which N2HET2 will detect the PWM
 * generated by N2HET1 as fail.
 * If PWM_DUTY is larger than MAX_MONITOR_DUTY_THRESHOLD then N2HET2
 * will detect the fail. */
#define MAX_MONITOR_DUTY_THRESHOLD 80.0

/* Min Duty cycle threshold beyond which N2HET2 will detect the PWM
 * generated by N2HET1 as fail.
 * If PWM_DUTY is less than MIN_MONITOR_DUTY_THRESHOLD then N2HET2
 * will detect the fail. */
#define MIN_MONITOR_DUTY_THRESHOLD 20.0

/* Pin number in N2HET2 to assert when a monitor detects fail */
#define NHET2_MONITOR_ERROR_PIN PIN_HET_0

/*****************************************************************************/
/* Macros for configuring N2HET1                                             */
/*****************************************************************************/
```

```
/* Change to desired PWM Period in terms of (uS)
 * the default 45.52uS will generate a 12-bit resolution on the PWM
 * with each resolution equal to HR=VCLK2=11.11nS.
 * 45.52uS / 11.11ns = 4096.
 */
#define PWM_PERIOD    45.52F

/* Change to desired maximum duty in terms of (%)
 * N2HET1 will generate a varying PWM from 0% duty to
 * the maximum duty specified by PWM_DUTY */
#define PWM_DUTY      100.0

/* allowable LR Prescaler values are 5, 6 and 7. Anything less will
 * will not have enough time slots for the N2HET program. HR prescaler
 * is always divide by 1 from VCLK2.
 * 7 -> one LR = 128 HR
 * 6 -> one LR = 64 HR
 * 5 -> one LR = 32 HR
 */
#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. This is expressed in terms
 * of (%). For example specifying DUTY_INCREMENT equal to
 * 2 will mean the duty cycle will start at 0% and the next
 * duty cycle will be 2% at a 2% increment. If 0 is
 * specified, then the N2HET1 will increment the duty cycle
 * at 1 HR (High Resolution) clock  */
#define DUTY_INCREMENT   0.0F

/* DUTY_INCREMEMT 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   0.0F

/* The increment delay is the delay at which the PWM modulation
 * will increase the duty cycle from one duty cycle to the
 * next duty cycle. This is expressed in terms
 * of (uS).  For example, if INCREMENT_DELAY is specified for
 * 10.0F (equal to 10uS) then the N2HET1 will wait for 10uS
 * before changing to the new duty cycle */
#define INCREMENT_DELAY  0.0F

/* Decrement delay. The Decrement delay is the delay at
 * which the PWM will decrease the duty cycle. This is expressed
 * in terms of (uS).  */
#define DECREMENT_DELAY  0.0F

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

By default the NHET1_PIN_PWM is set to PIN_HET_9 (a.k.a. N2HET1[9]) and NHET2_PIN_MONITOR is set to PIN_HET_16 (a.k.a N2HET2[16]). The reason for this default setting is because N2HET1[9] and N2HET2[16] are multiplexed onto the same device package terminal. This means that when N2HET1 module outputs its PWM onto the device pin, the state of the pin is fed back via the input buffer to the N2HET2[16] input. Therefore, with this default setup, you need not have any external connection between the PWM output and the monitor input. If you want to try out different N2HET2 pin for monitoring, then you need to make sure an external connection is made.

The triangle waveform PWM generation as well as the PWM monitoring are mainly handled by both the N2HET modules without CPU intervention . The host CPU is mainly handling the device initialization and configuration of the two N2HET modules. The rest of the time the host CPU stays in a loop unless an interrupt is generated to notify the host CPU that the N2HET2 has detected an out of range failure.

```
void main(void)
{
/* USER CODE BEGIN (3) */
    /* This example uses the N2HET1 generate a triangle wave PWM and uses
     * N2HET2 to monitor the PWM.
     * N2HET1 program: Async_PWM_Triangle_Wave.het
     * N2HET2 program: N2HET2_PWM_Range_Monitor.het
     */

    /* Enable CPU IRQ interrupt */
    _enable_interrupt_();

    /* initialize N2HET1 and N2HET2 based on HalCoGen settings */
    hetInit();

    /* Configure additional settings of N2HET2 based on the macros settings */
    configNHET1();

    /* Configure additional settings of N2HET2 based on the macros settings */
    configNHET2();

    while(1);


/* USER CODE END */
}
```

2.7 Example 1 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.

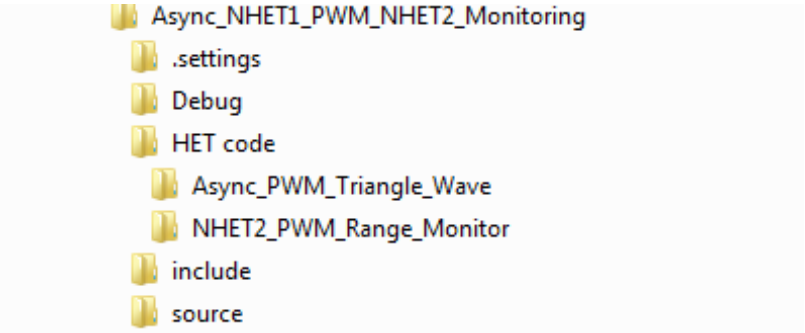


Figure 6. Async_NHET1_PWM_NHET2_Monitoring Project Directory Structure

3 Example 2 Description

In this example, the ePWM1 module is used to generate a fixed PWM on its ePWM1B pin. The N2HET2 is used to measure the PWM. The N2HET will constantly measure both the period and the duty cycle of the incoming PWM. If either the PWM period or duty cycle is not correct then the N2HET2 will assert the interrupt to the host CPU. The interrupt ISR will assert the nERROR pin. You should see the nERROR LED turned on. Upon detection of a failure, the N2HET2 will also stop itself.

Initially the ePWM1 is configured to generate a correct PWM with the right period and duty cycle. In order to experiment with different duty cycles and see how the N2HET2 will respond, you can press either the GIOA[7] switch on the HDK board or the GIOB[2] switch (User Switch A) on the LaunchPad. When pressed, a new but incorrect duty cycle is generated. The N2HET2 is expected to detect the failure. The nERROR LED will turn on for confirmation. Pressing the switch the second time, it will restore the ePWM to generate the correct PWM. The nERROR LED should turn off for confirmation. Every alternative time the switch is pressed, a new duty cycle with a 10% duty increment is generated. So the first time the switch is press, a 0% duty cycle is generated, the third time it is pressed, a 10% duty cycle PWM is generated, the fifth time it is pressed, a 20% duty cycle is generated and so on so forth. Every even times, the PWM is restored to the correct value. You can probe ePWM1B on the scope.

3.1 Example 2 N2HET2 Monitoring Flow Chart

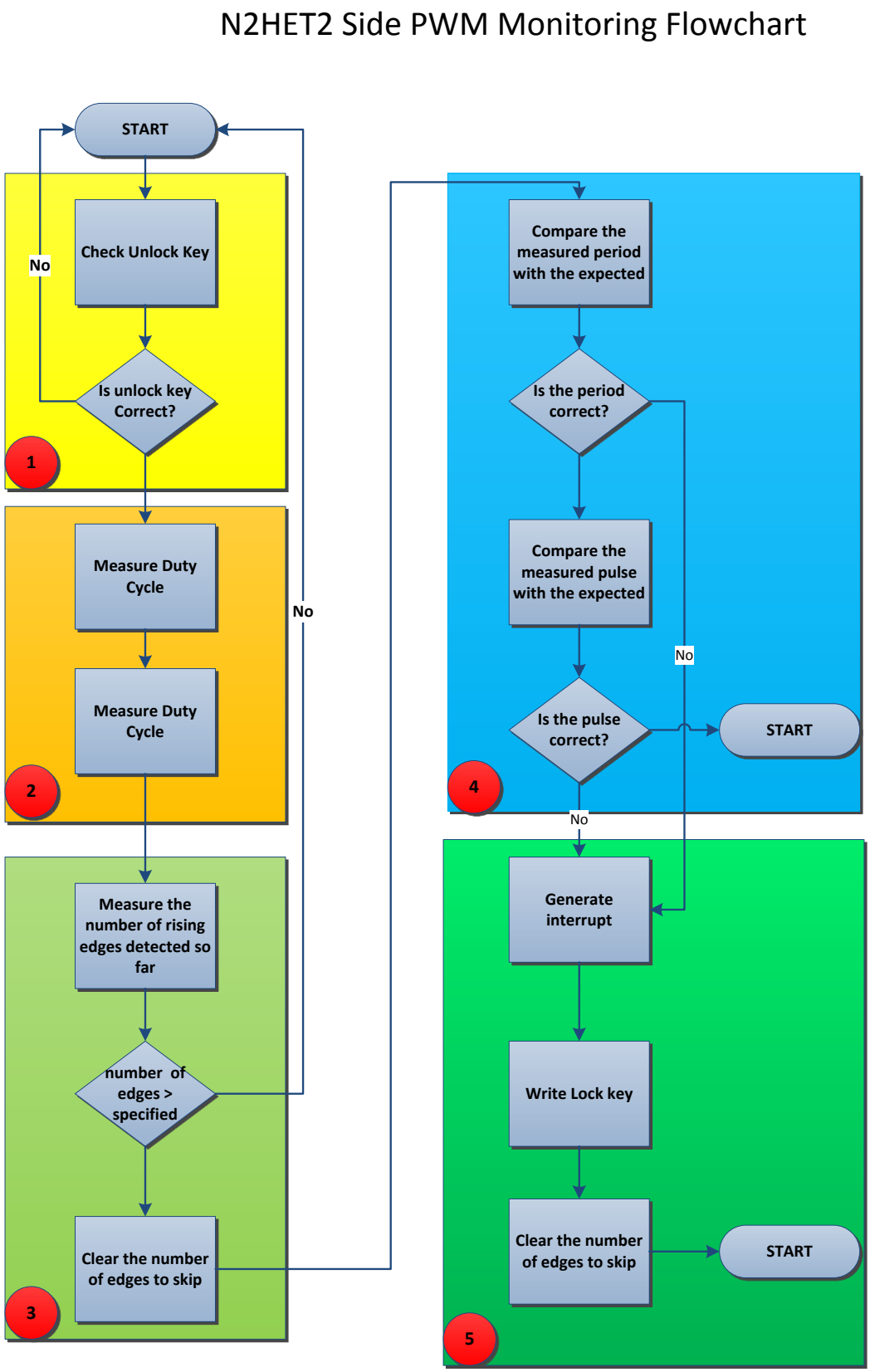


Figure 7. Example 2 N2HET2 Monitoring Flow Chart

The example 2 N2HET2 program code is illustrated below:

Step 1, the N2HET2 is put into a self-loop until a proper unlock key is written to the N2HET2. While the N2HET2 is locked, the host CPU can setup various parameters for the N2HET2 program.

Step 2, two PCNT instructions are used to measure both the period and the duty cycle of the incoming PWM.

Step 3 is to skip the initial edges so erroneous measurements can be thrown out.

Step 4, the measured period and pulse are compared against the expected values.

If failure is detected, generate interrupt to the host CPU and lock the N2HET2. Once the N2HET2 is locked, the state of the N2HET2 is frozen for your examination. The host CPU can choose to unlock the N2HET2 again in the N2HET2 ISR after it handles the error situation.

3.2 Example 2 N2HET2 Monitoring Program

The example N2HET2 program code for this example is illustrated below:

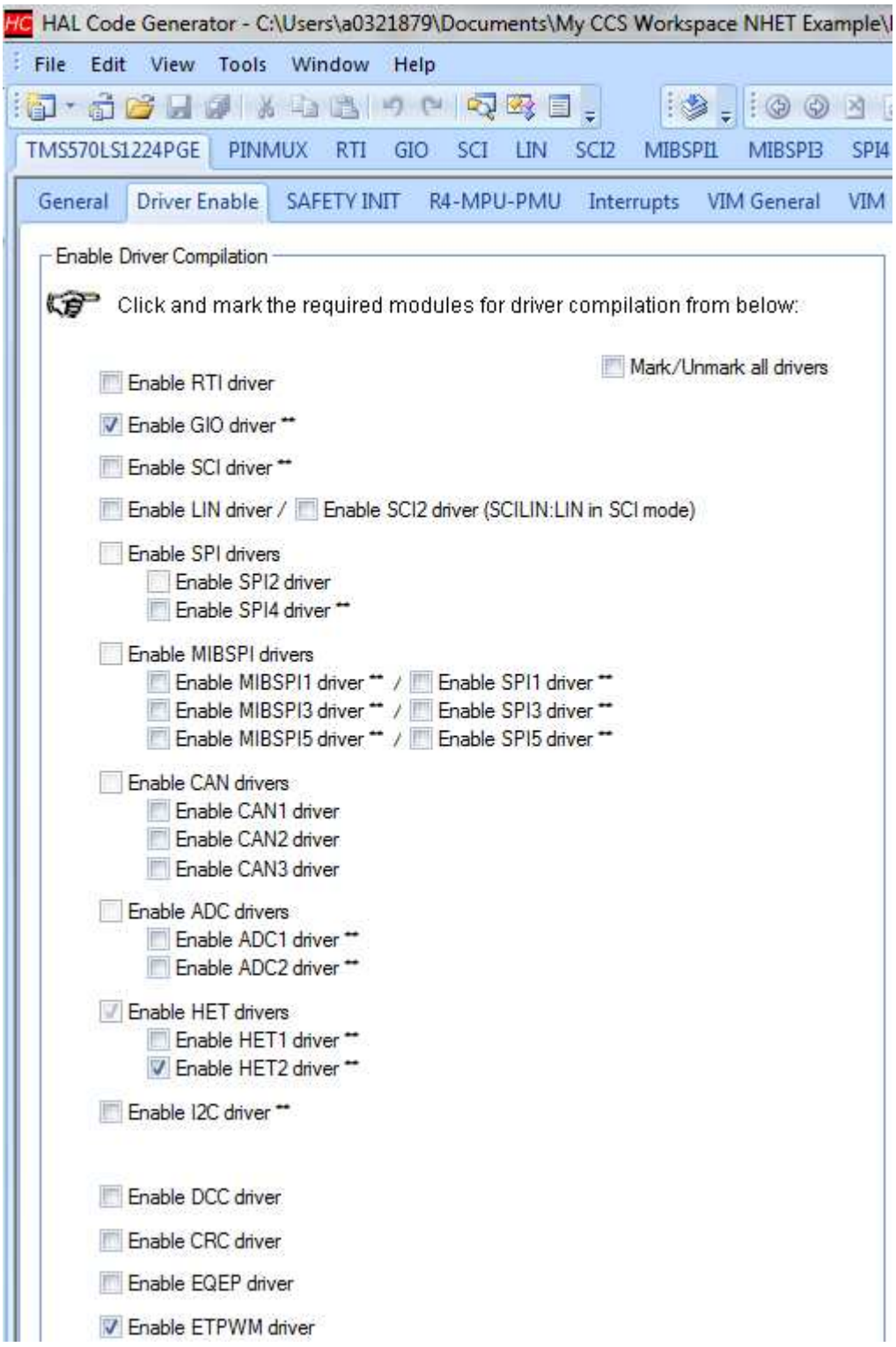
```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This example code is to be loaded to N2HET2 to monitor the PWM waveform generated
; by another source such as the ePWM module. The N2HET2 expects a known good period and
; duty cycle to be measured. If the ePWM generates an unexpected PWM waveform then
; the NHET2 will generate an interrupt to the host CPU.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Pin number that is used to measure the PWM period
PERIOD_MONITOR_PIN_NUM .equ 4
; Pin number that is used to measure the PWM duty. At the device level, this pin
; can be the same pin as for period measurement by using the HRSHARE feature.
DUTY_MONITOR_PIN_NUM   .equ 5
; Number of rising edges to skip for the measurements
SKIP_EDGES             .equ 3
; 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 NHET program. First the MOV32 instruction moves the
; initial value toa temporaray register T
L00   MOV32 { remote=DUMMY,type=IMTOREG,reg=T,data=0x5,hr_data=0};

; Compare the register T value with the key to unlock NHET. 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 NHET
L01   ECMP { next=L00,hr_lr=LOW,cond_addr=L02,pin=0,reg=T,data=UNLOCK_KEY,hr_data=0};

; Use PCNT RISE2RISE to measure period on the specified pin
L02   PCNT { hr_lr=HIGH,type=RISE2RISE,pin=PERIOD_MONITOR_PIN_NUM};

; Use another PCNT to measure the high phase of the PWM using type=RISE2FALL
; Note that the pin selected to measure pulse is pin=4. User must enable the HRSHARE
; feature in the NHET module so that PERIOD_MONITOR_PIN_NUM and DUTY_MONITOR_PIN_NUM
; will both share PERIOD_MONITOR_PIN_NUM on the device.
; Also note that the pin can be changed by the CPU. PERIOD_MONITOR_PIN_NU is used
; as the default pin  in this example. The reason to choose pin4 is that it shares
; the same muxed pin with eTPWM1B pin so that pin4/5 can be used to monitor eTPWM1B
; without any external connnection.
L03   PCNT { hr_lr=HIGH,type=RISE2FALL,pin=DUTY_MONITOR_PIN_NUM};

; Use ECNT to first detect the number of rising edges. The purpose here is to throw
; away the first rising edges becasue the period/pulse measurement from PCNT will
; not be accurate for the first edge.
L04   ECNT { cond_addr=L05,pin=PERIOD_MONITOR_PIN_NUM,event=RISE,reg=R,data=0};

; Compare with 3. Only when the number of edges is greater or equal to 3 will we
; take the pulse measurement from PCNTs
```

```
L05   MCMP { next=L00,hr_lr=LOW,cond_addr=L06,pin=0,order=REG_GE_DATA,reg=R,data=SKIP_EDGES};

; After the specified edges are detected, reset the compare value to 0
; so that when ECNT counter overflows it will not need to skip edges again.
L06   MOV32 { remote=L05,type=IMTOREG&REM,reg=NONE,data=0x0,hr_data=0};

; Move the period captured by PCNT RISE2RISE to a temp register T
L07   MOV32 { remote=L02,type=REMTOREG,reg=T};

; Subtract the captured value in T to the expected period value stored in the
; label REM_PERIOD. Note that the expected period stored in REM_PERIOD
; is actually the expected period plus one. For example, if the expected period is
; 10 then REM_PERIOD will store 11. The PCNT can measure the period with accuracy
; of +/-1 from the expected meaning it may measure within the bound of 9 to 11.
; If we set expected value to 11 then it eases our comparision so that the measured
; value should never be greater than REM_PERIOD. The largest difference between
; the measured value against the expected value will be between 9 and 11. To
; tolerate this difference of 2 we first do the subtraction and then perform a right
; shift of the result by two bits.
L08   SUB { src1=REM,src2=T,dest=IMM,rdest=NONE,remote=REM_PERIOD,smode=LSR,scount=2,data=0};

; If the compare does not match then enable interrupt to the CPU
L09   BR { cond_addr=L13,event=NZ,irq=ON};

; Move the pulse measurement stored in L03 to a temp register T
L10   MOV32 { remote=L03,type=REMTOREG,reg=T};

; Compare the measured pulse value with the epxected pulse value stored at
; REM_DUTY.
L11   SUB { src1=REM,src2=T,dest=IMM,rdest=NONE,remote=REM_DUTY,smode=LSR,scount=2,data=0};

;The subtract result should be zero if the measured pulse is equal to the expected pulse
; width. If not equal we will assert interrupt to the CPU
L12   BR { next=L00,cond_addr=L13,event=NZ,irq=ON};

; Once the mimatch is detected, we have a simple handler starting at L13 where we
; first move a disable value to L00 data field to stop the NHET program.
L13   MOV32 { remote=L00,type=IMTOREG&REM,reg=NONE,data=5,hr_data=0};

; We also clear the ECNT data field and also the R register. The R register also
; stores the current number of rising edges detected. The purpose is that once
; the CPU handles the mismatch in the interrupt ISR, it will restore the PWM
; generation and we want to throw away the PCNT period/pusle measurement for
; the first three rising edge again.
L14   MOV32 { next=L15,remote=L04,type=IMTOREG&REM,reg=R,data=0};

; Reload the compare value for number of risng edgeds to skip
L15   MOV32 { next=L00,remote=L05,type=IMTOREG&REM,reg=NONE,data=SKIP_EDGES};

; REM_PERIOD data field stores the expected period of the input. CPU needs
; to first initialize the expected value by writing to the data field here.
REM_PERIOD   BR { next=REM_DUTY,cond_addr=REM_DUTY,event=NOCOND,irq=OFF};

; REM_DUTY data field stores the expected high phase pulse of the input. CPU
; needs to first initialize the expected value by writing to the data field here.
REM_DUTY   BR { next=REM_DUTY,cond_addr=REM_DUTY,event=NOCOND,irq=ON};

; This is a dummy instruction. It is more a HET IDE issue where in L00 if a
; remote address is not given it is throwing an error even though that in
; L00 there is no need to move data from/to the remote field.
DUMMY   BR { next=DUMMY,cond_addr=DUMMY,event=NOCOND,irq=OFF};
```

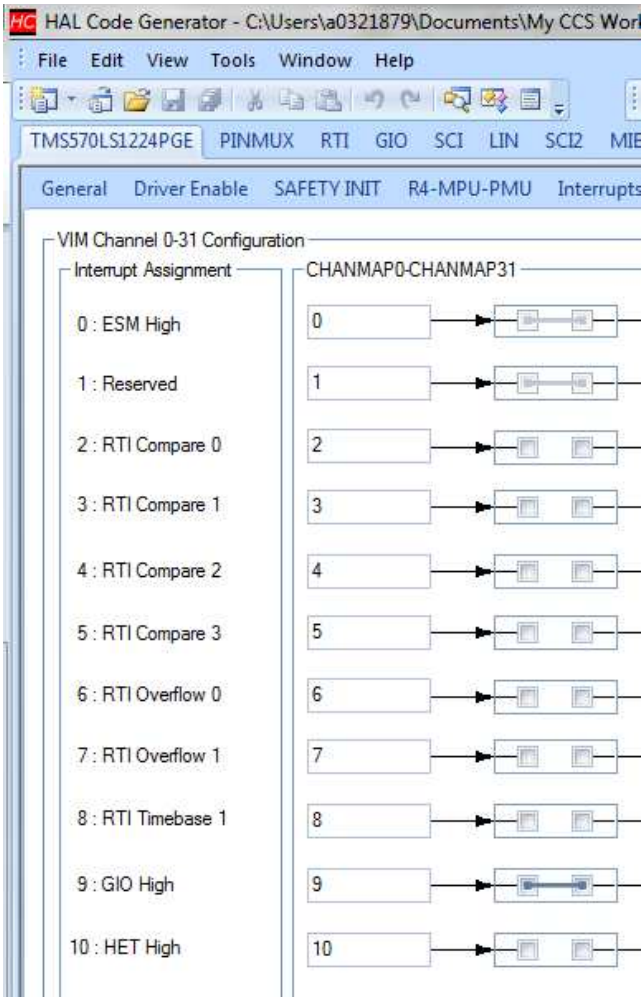
3.3 Example 2 HalCoGen Setup

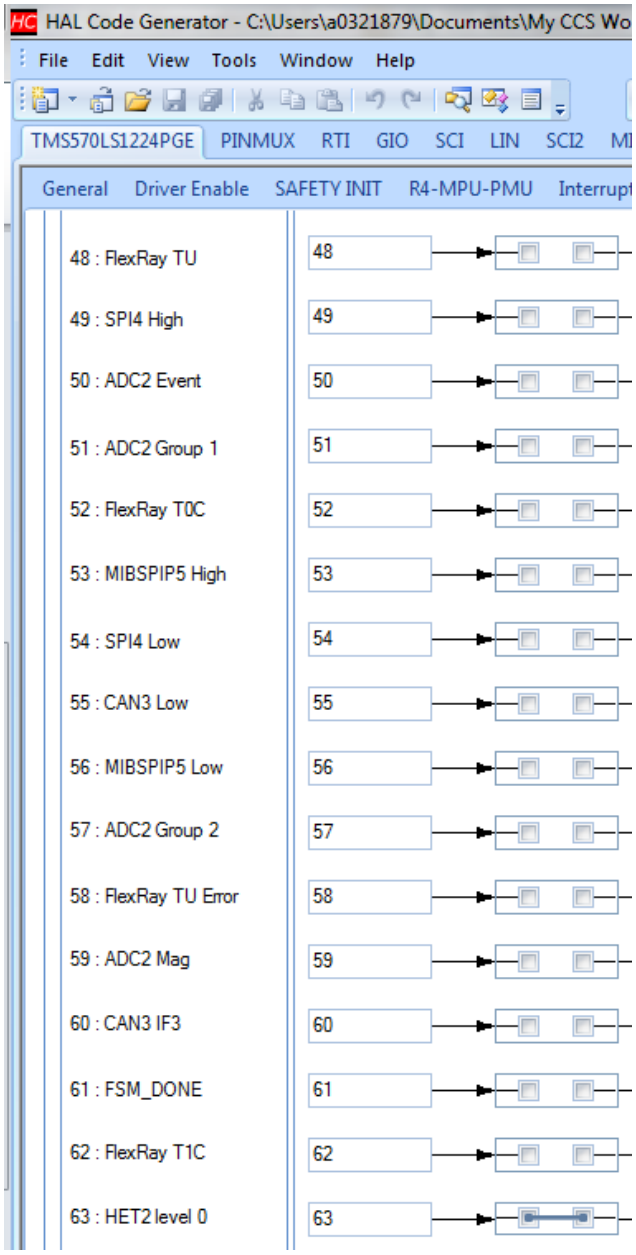
This example utilizes the HalCogen tool to configure the device. The target device selected in the HalCoGen for this example is the TMS570LS1224PGE. It can be ported to other variant devices that also contain the ePWM modules.

1. Create a new project: File → New → Project.
2. Enable GIO, N2HET2 and ePWM drivers.

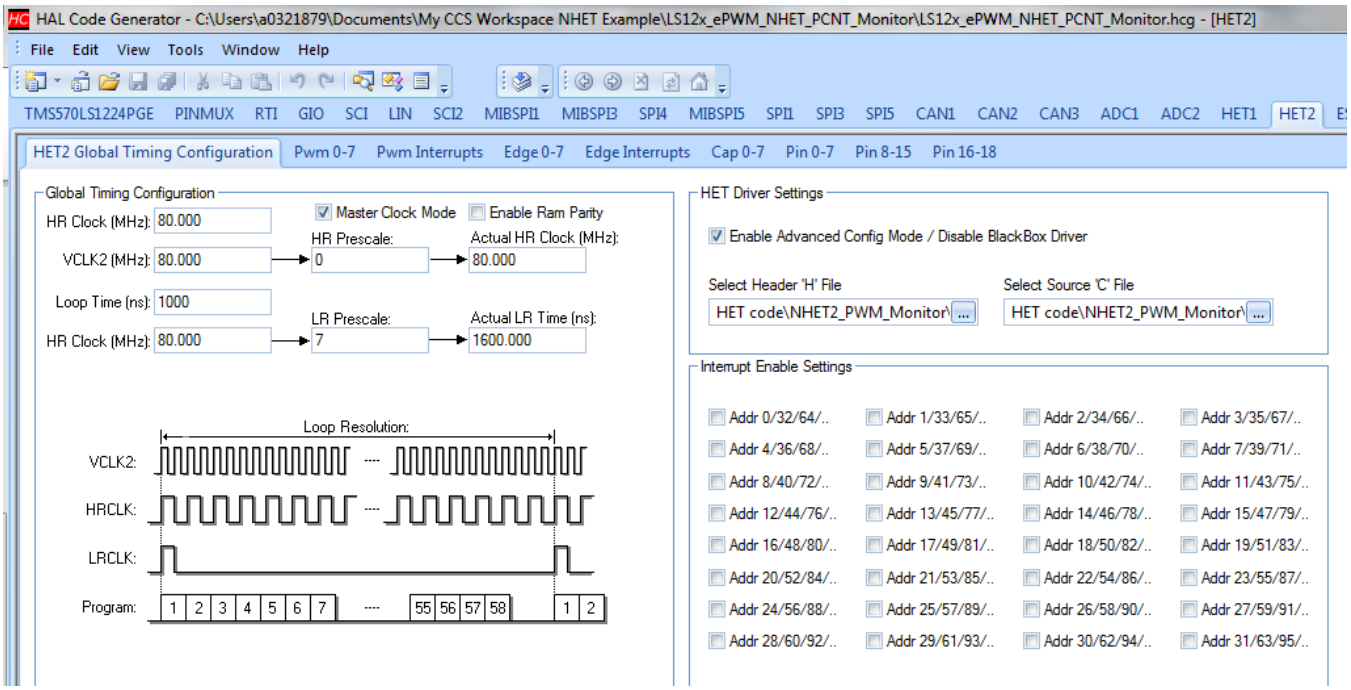


3. Enable GIO and N2HET interrupts.

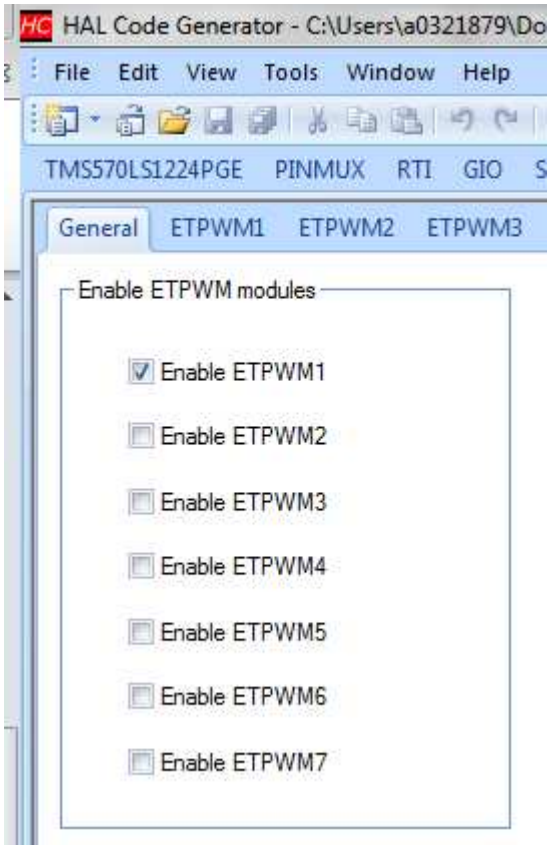




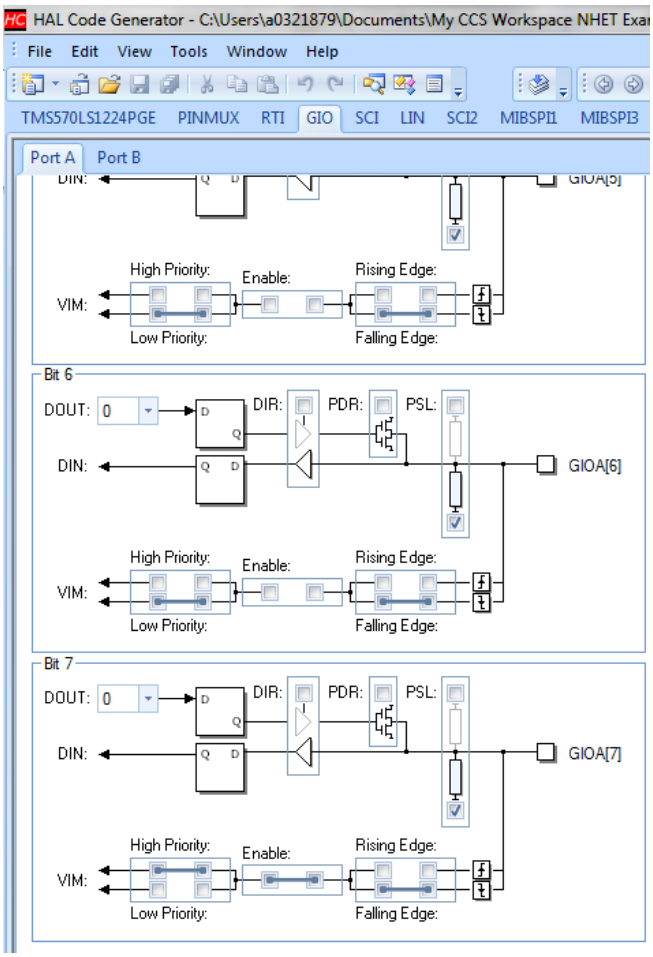
4. Configure N2HET2 to load the custom program.

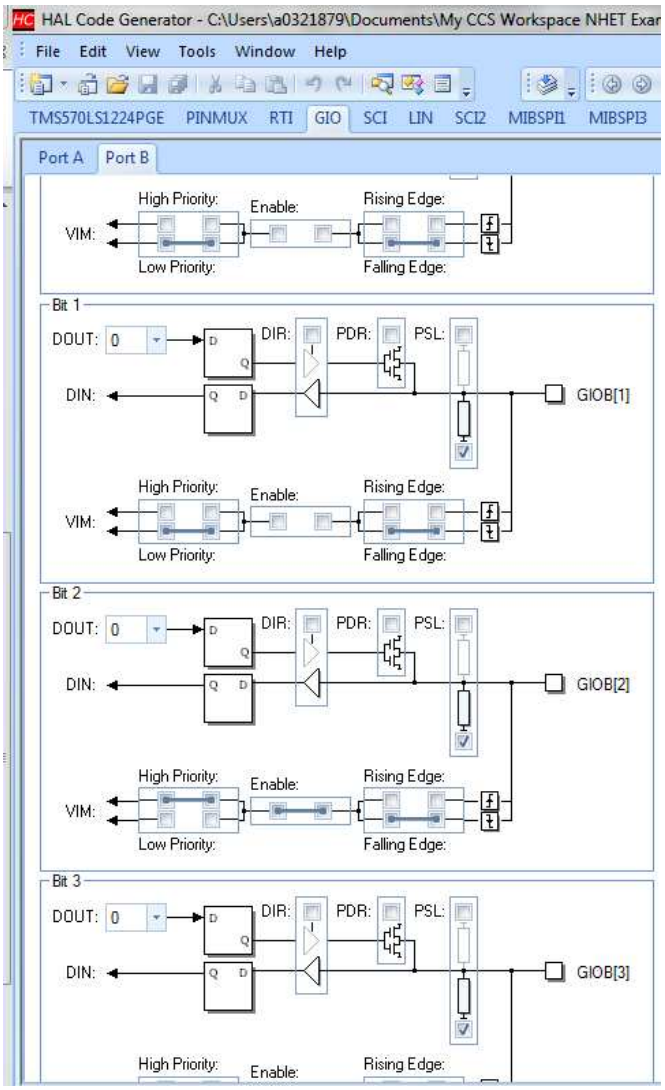


5. Enable ePWM1.

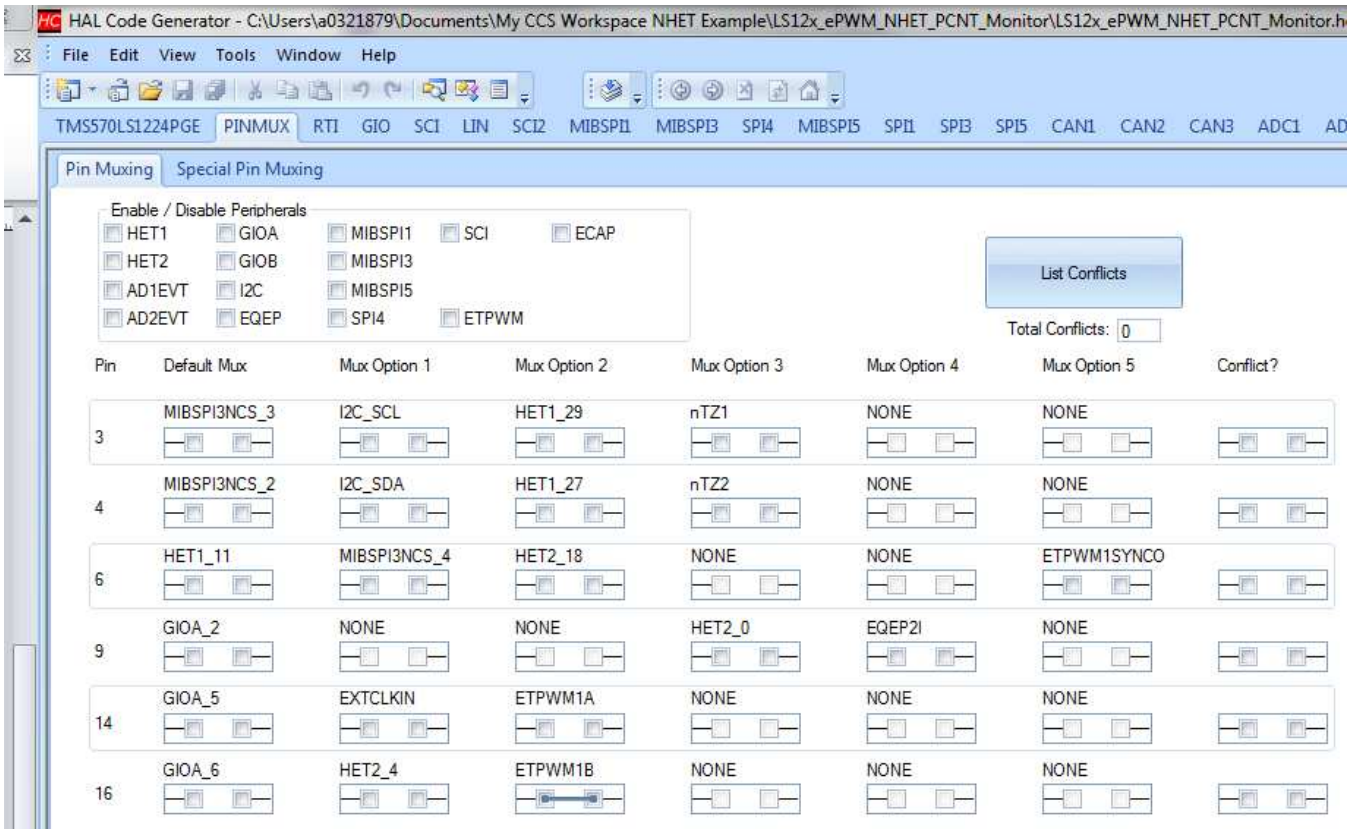


6. Enable GIOA[7] and GIOB[2] to generate interrupt on falling edge.

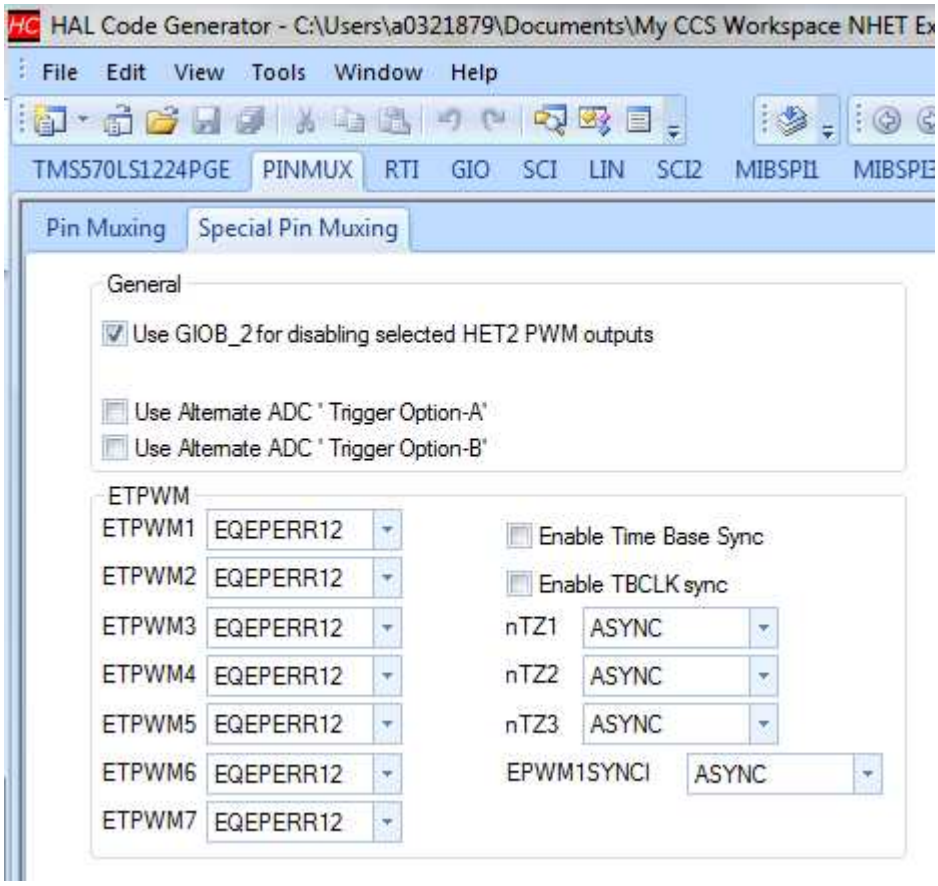




7. Configure ePWM1B as the functional pin on the device package pin 16 if the Launchpad is used. The launchpad contains a 144-pin PGE package. Note that if the HDK is used for this example where the 337ZWT BGA is on board then the device terminal would have been H3. However, from the PINMUX configuration point of view, they are the same. As can be seen in the Pin Muxing tab screenshot, the HET2_4 (a.k.a N2HET2[4] is also multiplexed with ePWM1B. This means that when ePWM1 module outputs its PWM onto the device pin, the state of the pin is fed back via the input buffer to the N2HET2[4] input. Therefore, with this default setup, you need not have any external connection between the PWM output and the monitor input. If you want to try out different N2HET2 pin for monitoring, then you need to make sure an external connection is made.



8. Configure GIOB[2] pin to receive the switch input on the Launchpad. This step is only necessary for example running on Launchpad.



9. File → Generate code.

3.4 Example 2 Host CPU Side Application Setup

On the host CPU side, its main task is to first initialize the device and configure both the ePWM1 and N2HET2 modules. Various macros are utilized to configure both the ePWM1 and N2HET2 modules. Below are the list of the changeable macros:

```
/* Change to desired PWM Period in terms of (uS) */
#define PWM_PERIOD    100.0F
/* Change to desired High Phase Duty in terms of (%) */
#define PWM_DUTY      55.0F
/* High Speed Time-Base Clock Prescale */
#define HSPCLKDIV     1
/* Time-base Clock Prescale */
#define CLKDIV        1
/* Pin number in N2HET2 to monitor the ePWM1B. */
#define NHET2_PIN_MONITOR PIN_HET_4
```

The PWM generation as well as the PWM monitoring are mainly handled by the ePWM1 and N2HET2 modules without CPU intervention. After the CPU finishes the configuration, it stays in a loop unless an interrupt is generated to notify the host CPU that the N2HET2 has detected an out of range failure. Here is the code snippet of the *main()*, the N2HET2 ISR and the GIO ISR.

```
void main(void)
{
/* USER CODE BEGIN (3) */
    /* NOTE:
     * the HET program loaded into N2HET2 is a custom program created using HET IDE. The N2HET2
     * program is written to capture and measure the PWM signal generated by the ePWM1 module.
     *
     * N2HET2 program: NHET2_PWM_Monitor.het
     *
     *
     * ePWM1B and N2HET2[4] are multiplexed together, a PWM generated by ePWM1 on ePWM1B can
     * be captured by the N2HET2[4] without configuring the PINMUX and without creating an
     * external connection. If user wants to use a different N2HET2 pin to monitor the ePWM1B
     * then an external connection must be made between the nPWM1B and the specified
     * N2HET2 pin.
     *
     */

    /* Enable CPU IRQ interrupt */
    _enable_interrupt_();
    /* initialize N2HET2 based on HalCoGen settings */
    hetInit();
    /* Initialize GIO based on HalCoGen settings */
    gioInit();
    /* Initialize ePWM based on HalCoGen settings */
    etpwmInit();
    /* Configure additional settings of ePWM based on the macros settings */
    configEPWM();
    /* Configure additional settings of N2HET2 based on the macros settings */
    configNHET2();
    while(1);

/* USER CODE END */
}

void hetNotification(hetBASE_t *het, uint32 offset)
{
    /* Turn ON nERROR LED */
    esmREG->EKR = 0xA;

    if (offset == (pHET_L12_1+1) ){
        /* bad_duty keeps track the number of times
           the N2HET2 ISR is entered due to pulse mismatch */
        bad_duty++;
    } else if (offset == (pHET_L09_1+1)) {
```

```
            /* bad_period keeps track the number of times
               the N2HET2 ISR is entered due to period mismatch */
            bad_period++;

        } else {
            /* Should never come here */
            while (1);
        }
    }

    void gioNotification(gioPORT_t *port, uint32 bit)
    {
        uint16 bad_CMPB;

        if ((bit == gioB_2) || bit == (gioA_7)){
            /* each time a switch is press, it either changes the duty cycle to an
             * failure value or restore to the expected value using variable "odd"
            */
            if ((odd % 2) == 0) {
                /* change the high phase width generated by ePWM1 by stepping through
                 * different duty cycle % from 0% to 100% at a 10% increment.
                 */
                bad_CMPB = ((odd >> 1) % 11)  * (TBPRD) / 10 ;
                etpwmSetCmpB(etpwmREG1, bad_CMPB);
            } else {
                /* Restore the epxected duty width */
                etpwmSetCmpB(etpwmREG1, CMPB);
                /* Unlock the NHET program again. */
                hetRAM2->Instruction[pHET_L00_1].Data = UNLOCK_KEY << 7;
                /* Turn OFF nERROR LED */
                esmREG->EKR = 0x0;
            }
        } else {
            /* should never come here since only GIOB[2] and GIOA[7] are enabled for interrupt */
            while (1);

        }
        odd++;

    }
```

3.5 Example 2 Project Directory Structure

This example project is named LS12x_ePWM_NHET_PCNT_Monitor; Figure 8 shows the project directory structure. The N2HET source program is stored under the HET code folder.

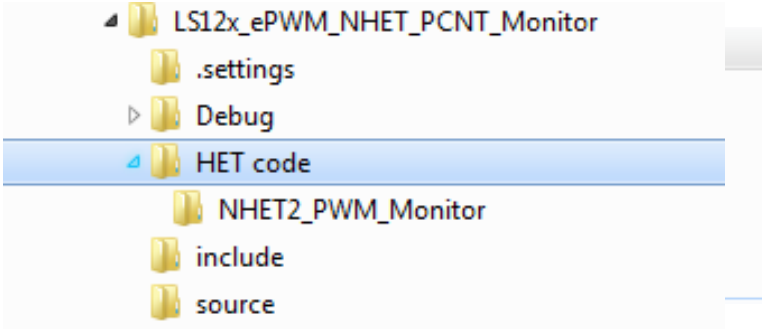


Figure 8. LS12x_ePWM_NHET_PCNT_Monitor Project Directory Structure

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