Application Report
SPRABA0B–August 2010

NHET Getting Started

Frank Noha — *AEC Automotive*

ABSTRACT

This application report describes the basic steps necessary to generate a simple pulse width modulation (PWM) with the next generation high-end timer (NHET) module. It reviews the calculations necessary to achieve a PWM with a certain frequency and duty cycle, setting up the NHET registers and writing the NHET program. Finally, it shows code excerpts of the setup to run the simple example code.

This document assumes that you have some basic understanding of the NHET terms used and the module operation.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/spraba0.zip.

Table 1. Abbreviations

Abbreviation/Accronym	Description
CNT	Virtual counter instruction
CNT_max	Maximum counter value of virtual count instruction CNT
data	Loop-resolution data field of an instruction
HCLK	Main system clock
hr	High resolution divide ratio
hr_data	High resolution data field of an instruction
HRP	High resolution period
lr	Loop-resolution divide ratio
LRP	Loop-resolution period
MCMP	Magnitude compare instruction
MCMP_DF[31:0]	32-bit data field of the MCMP instruction
PFR	Prescale Factor Register
PWM_duty	PWM duty cycle
RAM	Random access memory
ts	Timeslots
VCLK2	Peripheral clock used as master clock for NHET

Contents

List of Figures

List of Tables

1 PWM Calculation

This section shows how to calculate the PWM frequency and duty cycle of an output signal. In order to do this, certain assumptions are made regarding the frequency the device and the NHET module are working with and the desired PWM frequency, duty cycle and duty cycle update accuracy.

First, some of the basic settings of the later applications have to be assumed for the calculations. For example:

- Main system clock HCLK = 50 MHz
- NHET clock VCLK2 = 50 MHz
- PWM frequency = 10 kHz
- PWM duty cycle dc = 25%
- PWM accuracy = 20 ns steps

The accuracy requirement of 20 ns makes it necessary to program the high-resolution divide ratio (hr) to 1. This ensures that the hardware counters on the NHET pins are working with VCLK2 frequency and will generate the PWM signal with desired accuracy.

→ hr = 1

Next, the loop-resolution prescaler has to be selected. The loop-resolution period determines the stepsize between PWM frequency changes, but also influences how many NHET instructions can be executed in one loop-resolution period. A simple PWM can be generated with only 2 NHET instructions. Since there is no frequency stepsize requirement and the NHET program itself is very simple, it allows you to set the loop resolution prescaler to divide-by-8.

→ lr = 8

With this, there will be eight high-resolution clock cycles in one loop resolution.

The number of timeslots (cycles for instruction execution) in one loop resolution can be determined by:

ts = lr × hr = 8

This means that the total execution time of the NHET program must not exceed eight cycles. Most instructions only take one cycle, but a few are taking multiple cycles. In this simple case, the program will only take two cycles.

With these settings, the LRP and HRP can be calculated.

$$\text{LRP} = \frac{\text{lr} \times \text{hr}}{\text{VCLK2}} = \frac{8 \times 1}{50\ \text{MHz}} = 160\ \text{ns}$$

$$\text{HRP} = \frac{\text{hr}}{\text{VCLK2}} = \frac{1}{50\ \text{MHz}} = 20\ \text{ns}$$

Now a closer look at the NHET program implementation can be taken, since the base parameters of the NHET configuration have been determined.

As mentioned before, a simple PWM can be generated with only 2 NHET instructions. The PWM frequency is defined by setting up a virtual counter (CNT instruction) and the duty cycle of the PWM signal can be generated with a MCMP instruction.

The max parameter of the CNT instruction needs to be determined to achieve a PWM frequency of 10 kHz. The max setting defines the max value of the counter. The CNT increments every LRP cycle; once it reaches the max value, the counter is set back to 0.

$$CNT\_max = \frac{\text{Signal Period}}{LRP} - 1 = \frac{100\mu s}{160\ ns} - 1 = 624$$

With the above, the number of high-resolution clock cycles in one PWM period is:

HRC_pwm = (CNT_max + 1) × lr = (624 + 1) × 8 = 5000

Next, it is necessary to calculate the compare value for the MCMP instruction to generate the desired duty cycle. The MCMP instruction can be configured to set the pin to a logic *1* when the compare matches the virtual counter value. With a duty cycle requirement of 25% (high time of signal), the number of high-resolution clock cycles can be calculated to:

PWM_duty = ((CNT_max + 1) × lr) × (1 – dc) = ((624 + 1) × 8) × (1 – 0.25) = 3750

However, this value cannot simply be programmed as the compare value for the MCMP instruction. The implementation of the MCMP instruction splits the compare value into loop-resolution (data) and high-resolution (hr_data) fields. See Figure 1 for details.

Figure 1. MCMP Data Field

31	7	6	0
data		hr_data	
R/W-x		R/W-x	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

The lower 7 bits of the data field hold the high resolution part and the upper 25 bits hold the loop resolution part. The high-resolution field is fixed in size, but there are not always 128 high-resolution cycles in one LRP. This means that only certain fields of the MCMP hr_data field are valid depending on the lr setting. Table 2 shows the valid bits of the hr_data field based on the chosen lr divide ratio.

Table 2. Table 1. Interpretation of the 7-Bit HR Data Field

Loop Resolution Prescale Divide Rate (lr)	Bits of HR Data Field							HRP Cycles Delay Range
	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]	
1	X	X	X	X	X	X	X	0
2	V	X	X	X	X	X	X	0 to 1
4	V	V	X	X	X	X	X	0 to 3
8	V	V	V	X	X	X	X	0 to 7
16	V	V	V	V	X	X	X	0 to 15
32	V	V	V	V	V	X	X	0 to 31
64	V	V	V	V	V	V	X	0 to 63
128	V	V	V	V	V	V	V	0 to 127

With the setting (lr = 8), chosen as the project settings, only the upper three bits of the hr_data field are valid, so the calculated value has to be shifted left by four bits (or multiplied with 16).

The compare value written into the 32 bits of the MCMP data field is:

MCMP_DF[31:0] = PWM_duty × 16 = 60000 = 0xEA60

or in the representation of the MCMP data field structure:

data = 0x1D4

hr_data = 0x60

The necessary parameters have been calculated, now the NHET program can be written.

2 NHET PWM Program

It has been determined that a virtual counter (CNT instruction) and a compare (MCMP) instruction for the program is needed; however, a decision needs to be made on which NHET pin the PWM should be generated. In this example, the NHET[0] was chosen as the output pin.

The following code shows the implementation details of the two instructions in the NHET assembler.

```
L00: CNT{next       = L01, reg = A, max = 624}
L01: MCMP{next      = L00, reg = A, en_pin_action = ON, hr_lr=HIGH, pin = CC0, action
     PULSEHI, order = REG_GE_DATA, data = 0x1D4, hr_data = 0x60}
```

The labels *L00* and *L01* are optional, but help make it easier to follow the flow of the program. Labels have to start in the 1st column of a new line of code. Instructions cannot start in the 1st column of a new line, since they would be interpreted as labels.

The NHET does not know the concept of a program counter, however, each instruction points to the next instruction to be executed. With this, and the ability of certain instructions to divert the program flow under special conditions, it is very easy to implement sub-functions in a NHET program.

The following is a review of the instructions one-by-one.

```
CNT{next = L01, reg = A, max = 624}
```

The CNT instruction defines the virtual counter and is executed each loop resolution once. The *next* field points to the next instruction to be executed. In this case, it's the instruction at label *L01*. *reg = A* defines that the NHET internal register *A* is used to store the intermediate counter value. When the CNT is executed, the datafield of the instruction is incremented by one and the newly generated value is also copied into the specified register. The instruction keeps on incrementing each time it is executed, until it matches the *max* field coded in the instruction. In the very next loop, after reaching the max value, the data field of the CNT instruction is then reset to *0*.

```
L01: MCMP{next = L00, reg = A, en_pin_action = ON, hr_lr=HIGH, pin = CC0, action =
          PULSEHI, order = REG_GE_DATA, data = 0x1D4, hr_data = 0x60}
```

The MCMP instruction defines the compare value that should be compared to the virtual counter and also which action should be taken on a compare match. The *next* field points back to the CNT instruction, which means that the two instructions are executed in sequence. *reg = A* defines that the compare value stored in the *data* field should be compared to the value in register *A*. *en_pin_action = ON* tells the NHET that an action has to be performed at the specified pin at a compare match. *hr_lr = HIGH* defines that the high-resolution pin structure on the pin should be used to control the state change of the pin with high-resolution accuracy. *pin = CC0* determines that the pin action has to be performed on pin NHET[0]. CC0 is an alias for *0*, which is used by the assembler to generate the according bit encoding for the specified pin. *action = PULSEHI* defines that the pin should be set to a logic 1 state when the compare matches and that it should be set to a logic 0 state when the virtual counter overflows. *data* holds the loop-resolution compare value that is compared to the specified register and *hr_data* holds the high-resolution delay that is applied at the pin once the compare matches.

2.1 NHET Assembler

Now that the NHET program has been written and saved under a specific name (in this case, PWM_test.het), it needs to be translated into the opcode that the NHET can execute. This is done with the NHET assembler *hetp*. You should have the assembler already installed on your system, if not make sure to install it first.

The assembler can be executed on the command line. Here's an example of the command line to use:

```
hetp -n0 -hc32 PWM_test.het
```

There can be multiple NHET modules on a device. *-n0* tells the tool to assemble the code for NHET module 1. The NHET program is executed on the device in the NHET RAM. The application needs to copy the code during initialization of the device into this NHET RAM. To use the code later in the main project, the assembler generates C-code with the *-hc32* option. More details will be discussed in Section 3.

Before programming the microcontroller to run the program, you can check the configuration of the PWM with the NHET simulator.

2.2 NHET Simulator

To simulate the PWM, you need to invoke the simulator and select which device you want to use for simulation.

Do the following steps to create a new project.

1. Select *Project → New Project*
2. Define a name for the project, e.g., PWM_Test, and where you want to store the project.
3. Click *Next* to select the device that should be used for the simulation. In this case, select the TMS570LS20216SZWT device.

   The next window asks you about the clock frequency used to run the NHET. This is equal to the VCLK2 frequency that was defined as 50 MHz before.
4. Wizard asks to create a new NHET program file or if an existing one should be used. Select the previously generated PWM_test.het file, since the program has been written already.
5. Click *Finish* and the project will be created.

You should see something similar to the picture shown in Figure 2.

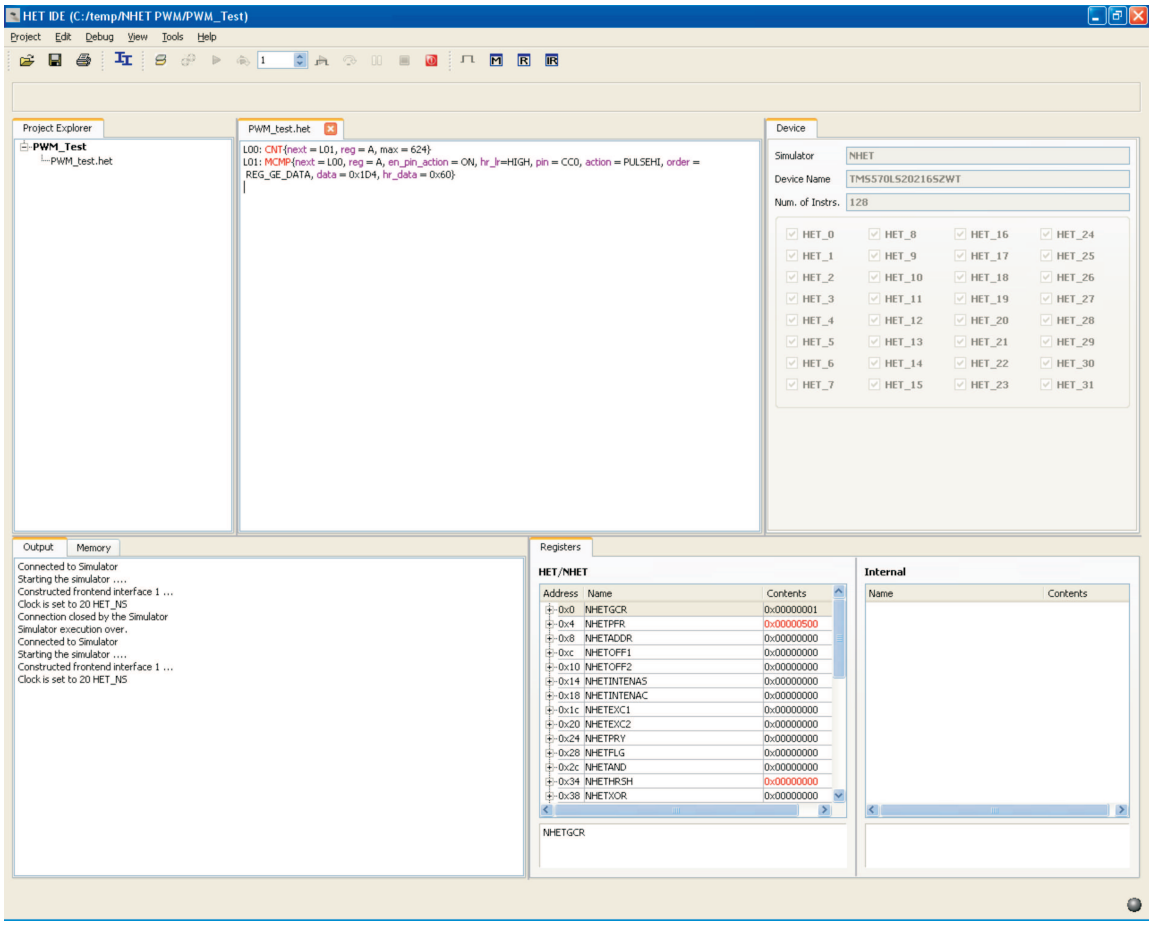

Figure 2. PWM Simulation Project Example

In the next step, set the correct simulator settings for the loop-resolution and high-resolution prescaler. This can be done by selecting *Project → Project Properties → Clock*. The loop-resolution value should be set to 8 and the HR value to 1. See Figure 3 for the setting.

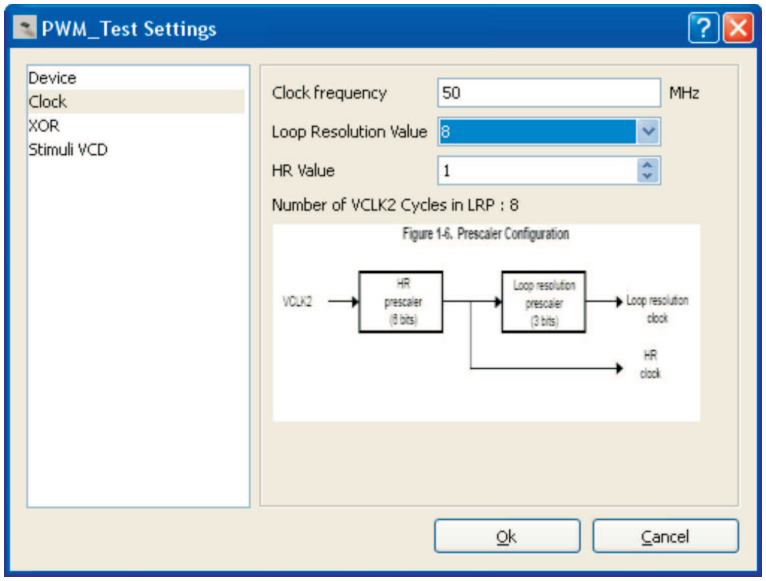

Figure 3. Simulation Clock Settings

After the clock settings are done, you can assemble the program by selecting *Debug → Assemble*. This generates the output file for the simulator to load. Once it's assembled without errors, the program can be loaded by selecting *Debug → Load HET Program*. This changes to the disassembly view and you can start running or single stepping the program.

Before doing this, you have to specify which pins you want to look at in the simulation results. This can be done with the *Tools → Waveform Wizard* menu item. In the window that opens, click the *Pins* tab and select *HET_0*. Figure 4 shows the pin configuration.

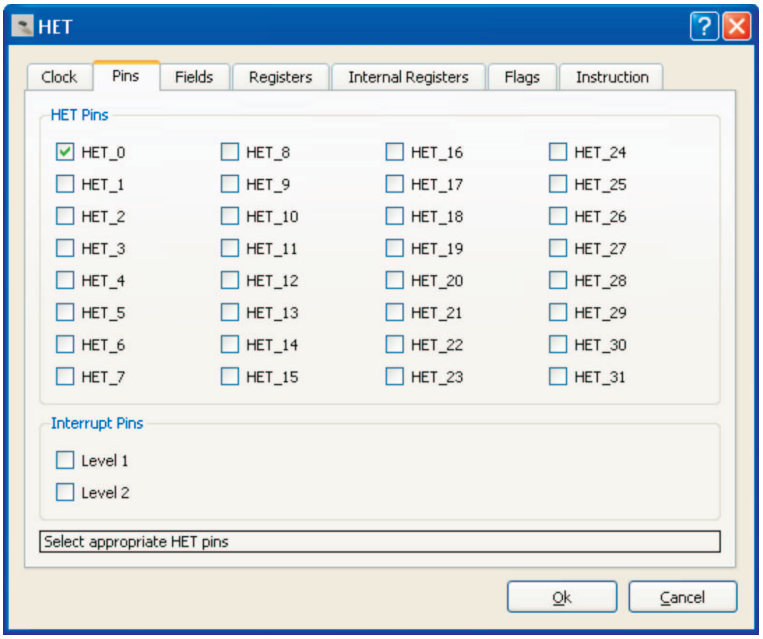

Figure 4. Simulation Pin Configuration

Once the pin has been selected, click the *Ok* button.

For a simple test, run the process for a certain amount of loops by inserting the number of loops in the spinbox toolbar and clicking the *Run For Loops* button. Figure 5 illustrates the setting.

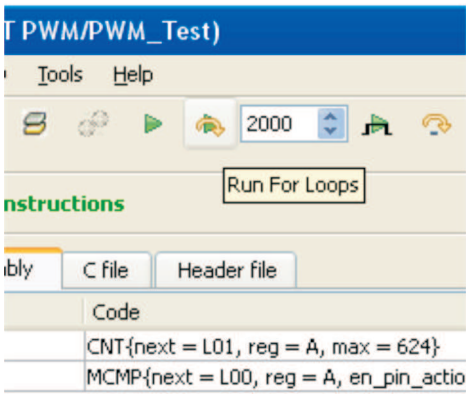

Figure 5. Simulation Running Specific Number of Cycles

After the simulation is complete, you can look at the generated waveform (see Figure 6).

Figure 6. Simulation Completion

To do this, stop the simulator by selecting *Debug* → *Stop*. Then, you can look at the generated waveform by selecting *View* → *Waveform*. You should see something similar to the picture shown in Figure 7.

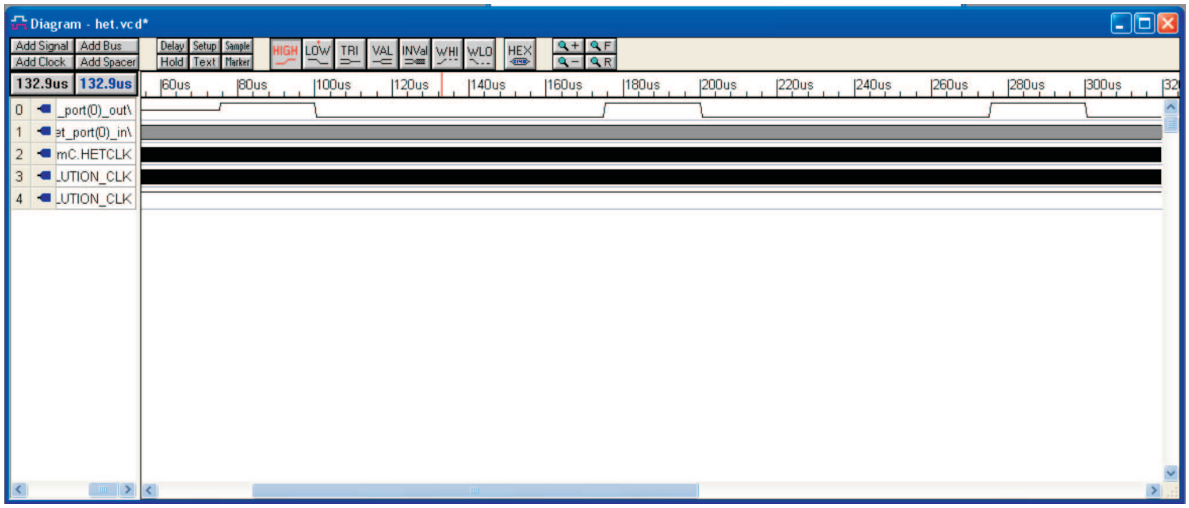

Figure 7. Simulation Waveform

As you can see, a PWM signal has been generated with 100 µs period (10 kHz frequency) and 25% duty cycle.

Now it is time to program a real device to get the same output.

3 Device Implementation

The previous run of the NHET assembler should have generated the files PWM_test.c and PWM_test.h. PWM_test.c defines a constant array that holds the values of the instructions (CNT, MCMP) in their program-, control-, data- and reserved-field representation (see the following).

```
HET_MEMORY const HET_INIT0_PST[2] =
{

    /* L00_1 */
    {
              0x00002C20,          /* Program Field */
              0x00000270,          /* Control Field */
              0x00000000,          /* Data    Field */
              0x00000000           /* Reserved      */
    },

    /* L01_1 */
    {
              0x00000000,          /* Program Field */
              0x00404058,          /* Control Field */
              0x0000EA60,          /* Data    Field */
              0x00000000           /* Reserved      */
    }
};
```

The contents of the array are stored in the main memory and need to be copied to the NHET RAM at application runtime, before the NHET is turned on. This can be accomplished by inserting.

```
memcpy( (void *)&e_HETPROGRAM0_UN, HET_INIT0_PST, sizeof (HET_INIT0_PST));
```

in the initialization section of the application.

The header file PWM_test.h defines a union to provide easy access to the two instructions.

```
typedef union
{
      HET_MEMORY    Memory0_PST[2];
      struct
      {
                    CNT_INSTRUCTION L00_0;
                    MCMP_INSTRUCTION L01_0;
      } Program0_ST;

} HETPROGRAM0_UN;

extern volatile HETPROGRAM0_UN e_HETPROGRAM0_UN;
```

This can be used to modify the instructions, e.g., duty cycle updates for the PWM, during the runtime of the application. The union is overlayed over the NHET memory and the address where to place the union can be defined in the linker command file. Below is an example how to do this in the linker command file.

```
SECTIONS
{

    .
    .HETCODE : { _e_HETPROGRAM0_UN = .;} > 0xFF460000      /* HET PROGRAM */

}
```

Now you have to configure the NHET registers for the correct settings to execute the example program.

Below is the code to set up all necessary registers:

```
typedef volatile struct hetBase
{
    unsigned GCR;          /**< 0X0000: Globoal control register             */
    unsigned PDFR;         /**< 0X0004: Prescale factor register             */
    unsigned ADDR;         /**< 0X0008: Current address register             */
    unsigned OFF1;         /**< 0X000C: Interrupt offset register 1          */
    unsigned OFF2;         /**< 0X0010: Interrupt offset register 2          */
    unsigned INTENAS;      /**< 0X0014: Interrupt enable set register        */
    unsigned INTENAC;      /**< 0X0018: Interrupt enable clear register      */
    unsigned EXC1;         /**< 0X001C: Exception control register 1         */
    unsigned EXC2;         /**< 0X0020: Exception control register 2         */
    unsigned PRY;          /**< 0X0024: Interrupt priority register          */
    unsigned FLG;          /**< 0X0028: Interrupt flag register              */
    unsigned : 32U;        /**< 0X002C: Reserved                             */
    unsigned : 32U;        /**< 0X0030: Reserved                             */
    unsigned HRSH;         /**< 0X0034: High resolution share register       */
    unsigned XOR;          /**< 0X0038: XOR share register                   */
    unsigned REQENS;       /**< 0X003C: Request enable set register          */
    unsigned REQENC;       /**< 0X0040: Request enable clear register        */
    unsigned REQDS         /**< 0X0044: Request destination select register  */
    unsigned : 32U;        /**< 0X0048: Reserved                             */
    unsigned DIR;          /**< 0X004C: Direction register                   */
    unsigned DIN;          /**< 0X0050: Data input register                  */
    unsigned DOUT;         /**< 0X0054: Data output register                 */
    unsigned DSET;         /**< 0X0058: Data output set register             */
    unsigned DCLR;         /**< 0X005C: Data output clear register           */
    unsigned PDR;          /**< 0X0060: Open drain register                  */
    unsigned PULDIS;       /**< 0X0064: Pull disable register                */
    unsigned PSL;          /**< 0X0068: Pull select register                 */
    unsigned : 32U;        /**< 0X006C: Reserved                             */
    unsigned : 32U;        /**< 0X0070: Reserved                             */
    unsigned PCREG         /**< 0X0074: Parity control register              */
    unsigned PAR;          /**< 0X0078: Parity address register              */
    unsigned PPR;          /**< 0X007C: Parity pin select register           */
    unsigned SFPRLD;       /**< 0X0080: Suppression filter preload register  */
    unsigned SFENA;        /**< 0X0084: Suppression filter enable register   */
    unsigned : 32U;        /**< 0X0088: Reserved                             */
    unsigned LBPSEL;       /**< 0X008C: Loop back pair select register       */
    unsigned LBPDIR;       /**< 0X0090: Loop back pair direction register    */
} hetBASE_t;

#define hetREG ((hetBASE_t *)0xFFF7B800U)


void NEHT_init(void)
{
         hetREG -> PFR = 0x00000300;        /* lr = 8; hr = 1          */
         hetREG -> DIR = 0x00000001;        /* NHET[0] = output        */
         hetREG -> GCR = 0x00070001;        /* Protect Program field   */
                                            /* Ignore Suspend          */
                                            /* NHET is master          */
                                            /* Turn NHET on            */
}
```

Writing to the Prescale Factor Register (PFR) sets up the timebases for the loop-resolution and high-resolution periods. Then, pin NHET[0] is configured as output and, finally, the NHET is turned on.

Once all these steps are followed and the device is set up with the correct HCLK and VCLK2 frequency, you should then be able to monitor the output waveform on pin NHET[0] with an oscilloscope. You should see that you are generating a PWM signal with 10 kHz frequency and 25% duty cycle.

A complete Code Composer Studio™ project with the entire device setup can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/spraba0.zip.

4 Conclusion

You should now be able to have a basic understanding of the steps necessary to write a NHET program. The NHET is a very powerful module and can be used for many application scenarios that would often be difficult to implement with a general-purpose hardware timer.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps

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