



nqBASIC

PROGRAMMING THE PWM OBJECT

www.nqBASIC.com
www.TechnologicalArts.com

NanoCore12 Family

BEFORE READING THIS GUIDE . . .

All code used in this document is written in nqBASIC, using the free nqBASIC IDE.

If you are new to nqBASIC, look for documentation, tutorials, and examples on the website www.nqBASIC.com. Before using this manual, you should already know how to create a project, where to write your source code, how to compile the code, and how to download it to your NanoCore12 module.

PURPOSE

If you want to use your NanoCore12 module to control the speed of a motor, vary the brightness of a light, or generate a variable voltage, the PWM object can do the job very well!

The **Pulse Width Modulator (PWM)**, built into NanoCore12, is a hardware subsystem that is controlled by the nqBASIC PWM object.

Objectives of this guide:

- Introduce the concept of pulse width modulation
- Discuss some uses and applications for a pulse width modulator
- Identify the pulse width modulators (channels) in NanoCore12
- Demonstrate how to derive the values needed for the PWM Registers
- Provide an example program demonstrating the PWM
- Analyze the program source code

WHAT IS PULSE WIDTH MODULATION?

Pulse Width Modulation is the process of controlling (modulating) the pulse width (duty cycle) of a logic level signal.

Duty cycle is the percentage of one cycle that a logic level signal is high. A cycle is defined as the smallest part of the waveform that repeats.

Figure 1 shows a logic level waveform with period T . During the period, the signal is high for 50% of the period and low for the remaining 50% of the period. We say that the signal has a **duty cycle** of 50%, and we sometimes call this a "square wave".

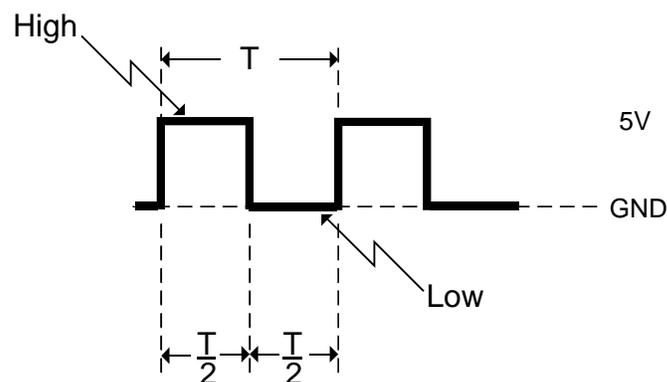


FIGURE 1

Figure 2 shows three independent logic level waveforms with duty cycles of 70%, 80%, and 100% respectively. In this case the duty cycle of the waveform is increasing.

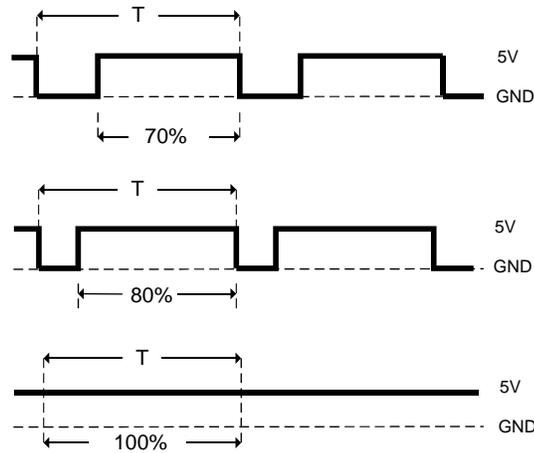


FIGURE 2

WHAT IS A PULSE WIDTH MODULATOR USED FOR?

A pulse width modulator is a device that can be programmed to vary a logic level waveform's duty cycle. The duty cycle produced can be used to control the amount of voltage or current applied to a load.

Consider **FIGURE 3**. The pulse width modulator generates a signal whose duty cycle controls the power delivered to a motor, M.

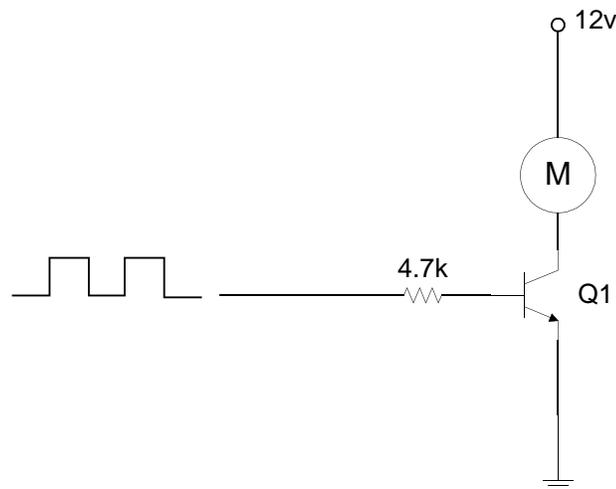


FIGURE 3

If the duty cycle produced by the PWM is 0%, then transistor Q1 is turned off. Therefore, no power is delivered to the motor M.

Table 1 shows what happens to the power delivered to the motor as the duty cycle increases.

Duty Cycle (%)	Percentage of Time Transistor Conducts	Average Power Delivered to Motor (%)
0	0	0
20	20	20
50	50	50
80	80	80
100	100	100

TABLE 1

In general, **the power delivered to a load is proportional to the duty cycle applied to the circuit.** As a result, the PWM can be used for varying light intensity, motor speed, and average voltage.

THE PWM CHANNELS IN NANOCORE12

In NanoCore12, a PWM is referred to as a **channel**. For example, a NanoCore12 that can generate four simultaneous PWM waveforms is said to have four PWM channels (or PWMs).

NanoCore12DX, **Figure 4**, has five PWMs.

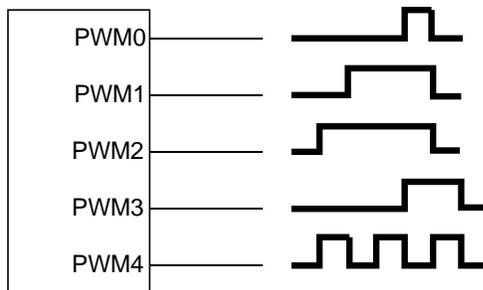


FIGURE 4

Each channel can be individually programmed to produce a signal with its own duty cycle without affecting the operation of neighboring PWM channels.

DERIVING THE PULSE WIDTH MODULATED SIGNAL

The pulse width modulated signal, produced by a PWM channel, is derived from the microcontroller's bus clock. This is accomplished by programming a series of registers with values that scale the bus clock down to the desired pulse width modulated frequency.

NOTE FOR ADVANCED USERS: By default, nqBASIC uses the phased-locked loop (PLL) feature of the microcontroller to boost the bus clock to 24 MHz. If you use the System PLL object to change the bus clock, it will affect the PWM subsystem.

CALCULATING PWM SETTINGS FOR 8-BIT MODE

Each PWM channel has four programmable PWM variables used to reduce the bus clock to the desired pulse width modulated frequency. These variables are:

- BUS CLOCK DIV
- SCALED DIV
- PeriodValue

A few things to keep in mind: you want to have the largest period value possible, as this produces the most accurate PWM duty cycle. Hence, you should use BUS CLOCK DIV and SCALED DIV to divide down the clock so as to end up with the best possible range for the period. The following example calculations are based on this approach, of trying to maximize DutyCycleValue.

The overall formula is this:

$$\text{PWM Frequency} = \text{Bus Clock} / \text{BUS CLOCK DIV} / (2 * \text{SCALED DIV}) / \text{PeriodValue}$$

The possible division factors for BUS CLOCK DIV are 1, 2, 4, 8, 16, 32, 64, 128 which in turn are associated with constant keywords in nqBASIC (refer to Table 2).

<i>BUS CLOCK DIV Value</i>	<i>BUS CLOCK DIV</i>
TIMER_DIV_1	divided by 1
TIMER_DIV_2	divided by 2
TIMER_DIV_4	divided by 4
TIMER_DIV_8	divided by 8
TIMER_DIV_16	divided by 16
TIMER_DIV_32	divided by 32
TIMER_DIV_64	divided by 64
TIMER_DIV_128	divided by 128

TABLE 2

The possible values for SCALED DIV are 0 to 255. So, using a value of 50 would further divide the bus clock by 100, for example.

PeriodValue is the maximum value your DutyCycleValue can be. That's why, the larger it is the more discrete pulse-widths you'll be able to generate. The maximum value is 255, and this is the final divisor of the bus clock for the PWM object.

EXAMPLE

PROBLEM:

It is desired to generate a pulse-width modulated signal with a frequency of 10Hz and a duty cycle of 50%. Determine the values for the PWM constants:

- BUS CLOCK DIV
- SCALED DIV
- PeriodValue

SOLUTION:

The bus clock = 24MHz. So we need to reduce 24MHz to 10Hz by programming the various PWM registers with appropriate values.

STEP 1: PERIODVALUE

Let **PeriodValue=250**

Bus clock / Period Value = 4MHz/(250) = 16kHz

The value 250 was selected to give us a nice round figure which can be further reduced by BUS CLOCK DIV and SCALED DIV.

STEP 2: BUS CLOCK DIV

BUS CLOCK DIV = TIMER_DIV_32

(96kHz) / (16) = 3kHz

3kHz can be easily reduced to 10 Hz by dividing by 300.

STEP 3: SCALED DIV

SCALED DIV = 150

3kHz / (2*150) = 10 Hz

A COUPLE OF NOTES BEFORE SHOWING YOU THE CODE . . .

Changing PWM signals at runtime

You can change the values of the DutyCycleValue Register and the PeriodValue Variables during runtime. Each PWM channel has its own counter. Changes to the period and duty cycle do not take effect until the counter resets back to 0. Thus there is no signal overlap when the signal transitions from one frequency to another.

PWM Pin-multiplexing

The microcontroller PWM module normally connects to Port P pins on the microcontroller, by default. However, on NanoCore12, most PortP pins are not available. Fortunately, the PWM pins can also be routed to Port T pins, via an internal logic block called a MUX. This re-routing is done automatically by nqBASIC, so that PP0 through PP4 are connected to PT0 through PT4, respectively. On the 40-pin module, a sixth PWM channel is available, via PP5.

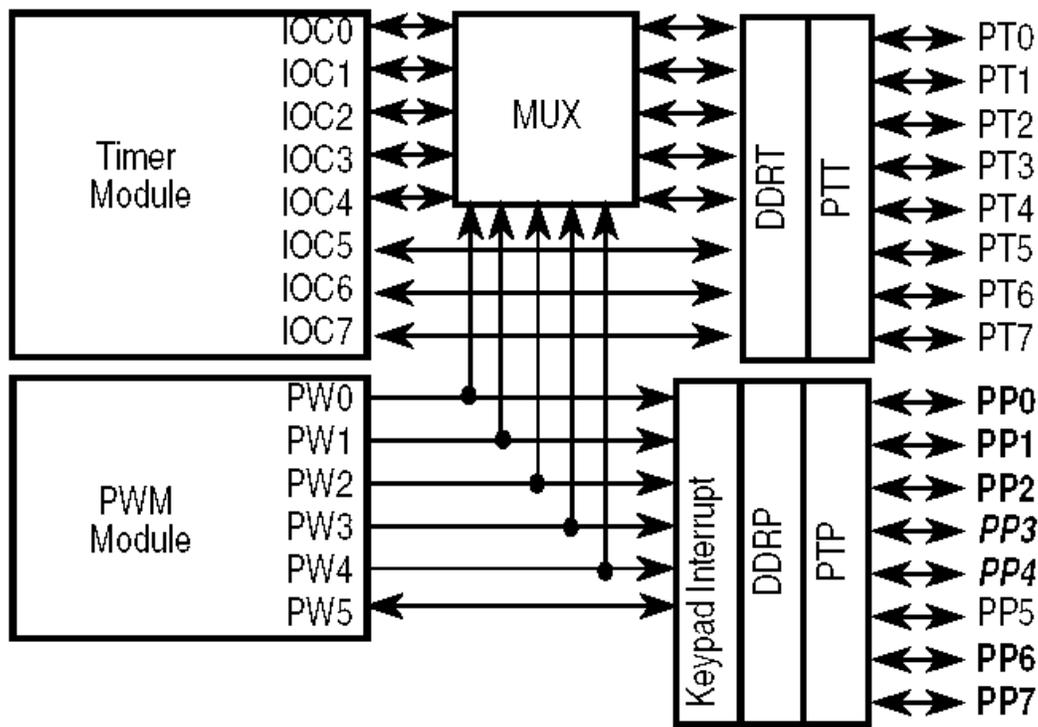


FIGURE 5

The nqBASIC code for our 10Hz 50% duty cycle PWM signal is shown in LISTING 1.

```
//Object declaration
dim PWM1 as new PWM (PP0) //PWM Object associated with PP0
main
  //Setup the clocks for PP0, PP1, PP4, PP5 PWM.PWM_Res_PP0145
  (TIMER_DIV_16, 50)

  //Generate a 50% PWM duty cycle
  //PWM Period as 250
  //PWM duty as 125
  PWM1.PWM_Start(PWM_MAIN_CLK, PWM_NORMAL, 250, 125)
end main
```

LISTING 1

After creating a project and entering the code shown above, set your NanoCore12 module to LOAD mode, hit reset and then use Build and Load (F6) to download the program to NanoCore12. After download, switch NanoCore12 to RUN mode and reset it once more. If you have done everything correctly the LED connected to PT0 should be flashing!

NOTE: If you aren't using a Docking Module or School Board, you should connect an LED to pin PT0 through a current-limiting resistor to Ground. The anode of the LED goes to the PT0 pin. Use the data sheet for your NanoCore12 module to identify the pin number corresponding to PT0.

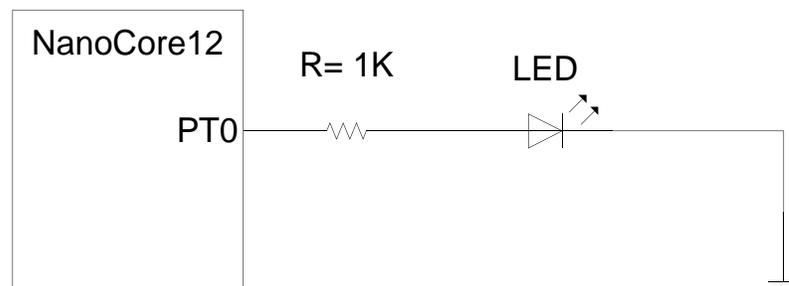


FIGURE 6

ANALYZING THE SOURCE CODE

Don't worry if you don't understand the source code, because now we will analyze it.

```
dim PWM1 as new PWM (PP0)
```

Declares a PWM object named PWM1 that is linked to PP0 (i.e. PT0).

```
PWM.PWM_Res_PP0145 (TIMER_DIV_16, 50)
```

Sets up the PWM subsystem by initializing the clock, which in turn determines the frequency of the PWM waveform generated.

```
PWM1.PWM_Start(PWM_MAIN_CLK, PWM_NORMAL, 250, 125)
```

Generates the PWM waveform on PWM1, using the main clock, with normal polarity, and with a duty cycle of 50% (i.e. 125/250 represents 50% on-time).

GENERATING A 16-BIT PWM WAVEFORM

The Period and Duty values can each extend their range up to 65535 to allow more precise values for duty cycle. The microcontroller does this by concatenating two PWM channels. It is done transparently by nqBASIC, but it results in the number of available PWM channels being halved (i.e. only three channels, instead of six).

Below is a simple 16-bit PWM example:

```
//Object declaration
dim PWM1 as new PWM(PP0) //PWM Object associated with PP0

main
  //Setup the clocks for PP0, PP1, PP4, PP5
  PWM.PWM_Res_PP0145 (TIMER_DIV_16, 50)
  //Generate a 50% PWM duty cycle
  //PWM Period as 65000
  //PWM duty as 32500
  PWM1.PWM_Start_ext(PWM_MAIN_CLK, PWM_NORMAL, 65000, 32500)
end main
```

SUMMARY

- The pulse width modulator is a hardware subsystem which can produce a logic level waveform whose frequency and duty cycle are programmable
- Each PWM channel has four registers which can be programmed to derive a desired frequency and duty cycle from the bus clock
- The method for selecting the values of these registers was described in this guide
- The duty cycle and frequency can be programmed to changed during runtime