CPEN 403 Embedded Systems

Lecture 6

PWM in Embedded Systems

March 7, 2023

Godfrey Mills, PhD

Email: gmills@ug.edu.gh

Phone: 020-549-6944

TA: Douglas Gidimadjor

Lecture 06 Outline

- What is PWM
- PWM duty cycle
- Why PWM in embedded systems
- Advantages of PWM
- PWM application for devices
  - Regulation of lights
  - Motor control
- PWM frequency
- Implementation of PWM

What is Pulse Width Modulation (PWM)

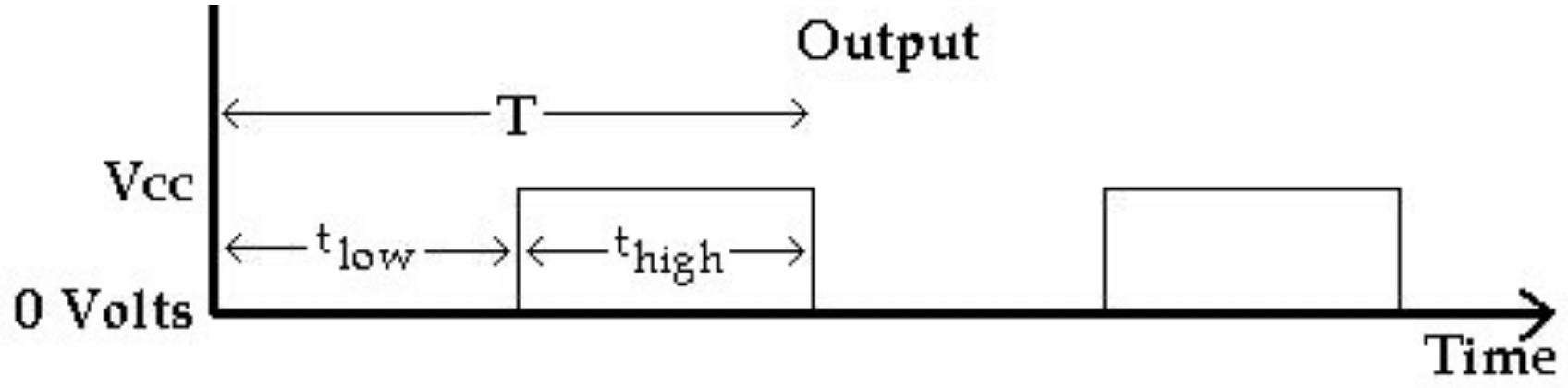
- In Lab 1, we interfaced MCU with LED and control duration of ON and OFF state (blinking) of the LED using software (delay function) and later, using the timers on the MCU
- In some applications, we may want MCU to vary the amount of power delivered to the LED (or connected devices) or change the speed of operations intermittently
- An effective way to implement this power or speed variation is to use the **PWM** on the MCU >> this is commonly used technique for controlling power to devices
- ***PWM is a peripheral timer device that is used to vary the duty cycle (ON time versus total time) of a voltage signal***

What is Pulse Width Modulation (PWM)

- **Cont'd…**
- Many MCU have hardware PWM feature that make it possible for the configuration of pins that toggles based on timer value
- PWM is used in many applications such as DC motor control, piezo speaker control, DAC operations
- Example, commonly used method for dimming LED is PWM
  - brightness of LED is controlled by varying duty cycle where longer duty cycle gives brighter LED and lower duty cycle gives dim LED

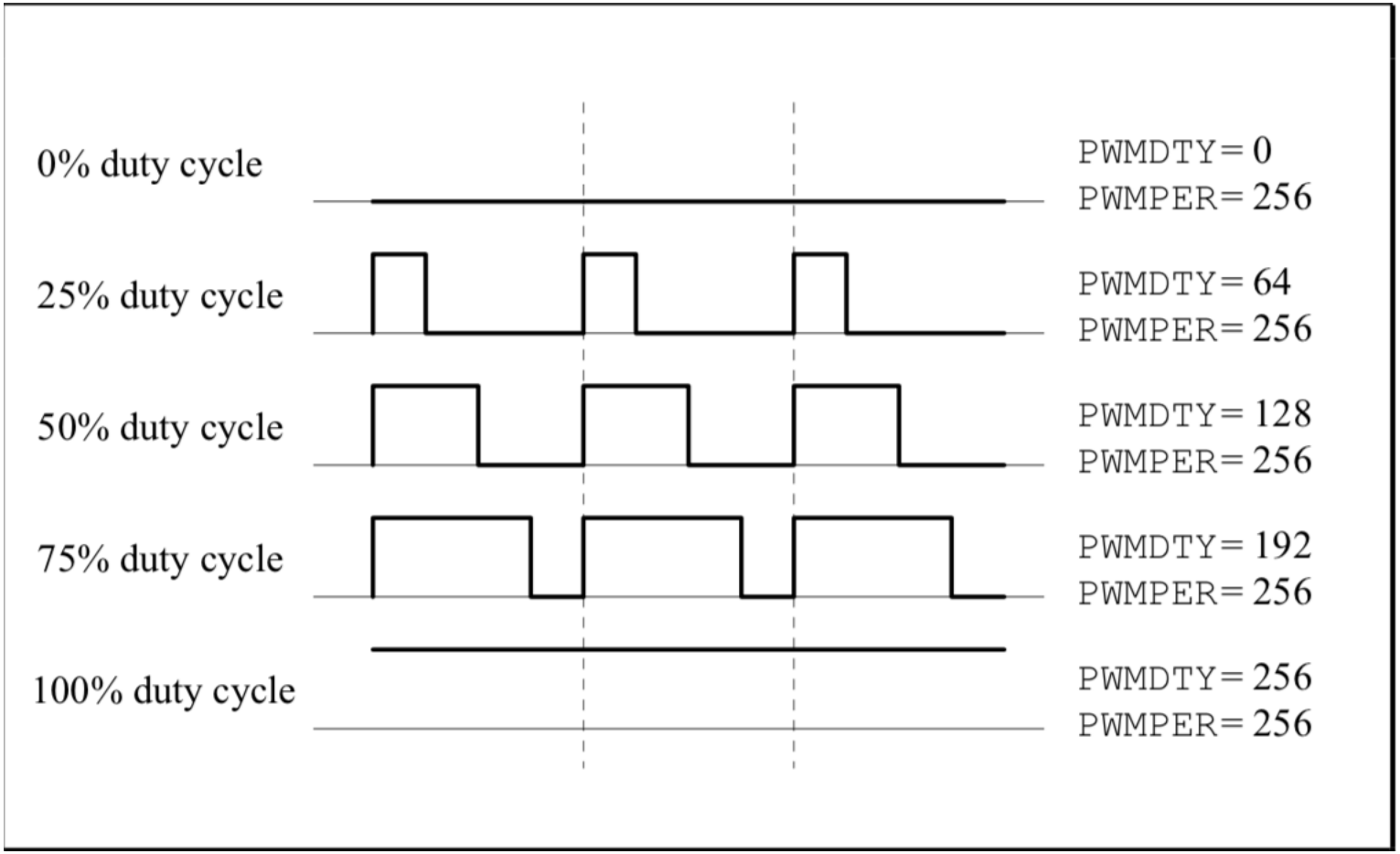
What is Pulse Width Modulation (PWM)

- **Cont'd…**
- Output of PWM basically switches repeatedly between **high** (logic "1" or 5V) for portion of time and **low** (logic "0" or 0V) for remainder of the time
- For application, the system is designed in such a way that the H+L is constant >> frequency is fixed



What is Pulse Width Modulation (PWM)

- Cont'd…



Characteristics of PWM

- A PWM signal is characterized by the following :
  - ***Modulation frequency*** >> number of pulses/sec (fixed)
  - ***Period*** >> arbitrary time period (1/modulation frequency) within which the PWM takes place >> it is chosen to give best results for a particular use
  - ***ON-time*** >> amount of time that each pulse is ON
  - ***Duty cycle*** >> fraction of time the signal is high compared to signal period >> ON-time/Period >> **H**/(**H**+**L**)
  - ***Resolution*** >> maximum number of pulses that can be packed into a given PWM period

Why the need for PWM

- Although PWM is used in variety of applications, one of the important uses of PWM is DAC operation for MCU that do not have on-board DAC capabilities
  - To achieve DAC capabilities, the PWM duty cycle is used to represent analog values when connected to the PWM output to an op-amp >> common design in applications.
  - Most systems that regulate something or some form of operation make use of PWM and op-amp to give analog output, which can be used to control a hardware device.
  - Analog output can be read back to the MCU through an ADC, and the MCU can alter the PWM duty cycle.

Why the need for PWM

- **Cont'd….**
- Another important reason for using the PWM is the ability enable device control by making it possible to turn ON connected devices for a limited amount of time and turn OFF the device for a limited amount of time.
- Another reason for using the PWM is the ability to enable power loss control through switching
  - When a switch is OFF, no current flows, and when it is ON, current flows but there is no voltage drop across switch >> power loss (V x I), is thus in both cases close to zero.
  - The longer a switch remains in ON state over OFF period, the higher the power supplied to the device

PWM application for LED control

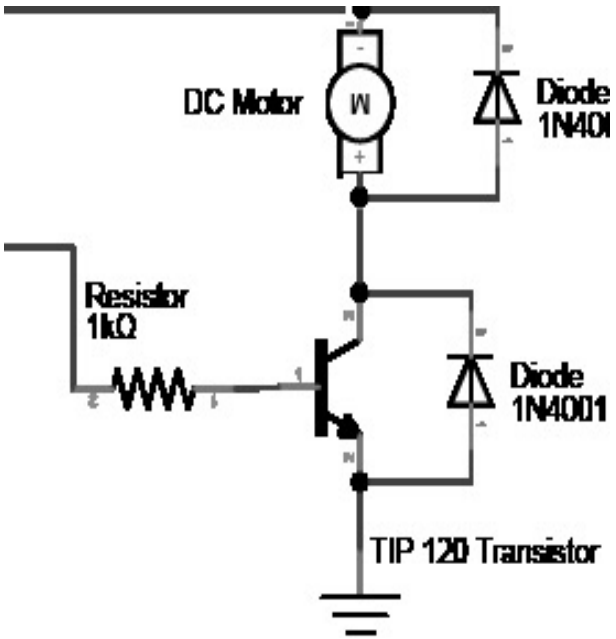
- Dimming an LED involves controlling amount of current that flows through LED or device.
- When we quickly turn ON and OFF the LED by reducing the current, there are unintended consequences like color shifts and voltage dropouts >> blinking that is undesired
- Rather than changing number of times the output goes ON and OFF, we can change how long output stays ON and OFF
- Better way to achieve this is to use the PWM mechanism
- With PWM, we can turn an LED ON and OFF many times per second >> through adjusting the % of ON time and the OFF time, the brightness of the LED can be controlled.

PWM application for dc motor control

- PWM can also be used for DC motor control
  - Basic concept of controlling DC motor is to use a variable resistor >> this turns to generate heat and waste power
    - We can solve this problem by using a PWM
  - Speed of motor can be controlled or regulated by changing the width (or duration) of the voltage pulses
    - The longer the voltage pulses, the faster the motor turns >> cost effective solution
  - ***It is important to ensure that the interface to the MCU can handle the current required by coils >> use a limiting resistor to reduce current to the transistor***

PWM application for dc motor control

- **Cont'd….**
- Sample DC motor control circuit using transistor to serve as a switch and diode to minimize the effect of return current
- NB the BJT is connected to MCU via a limiting resistor for ??



PWM application for dc motor control

- **Cont'd….**
- To control the speed of a dc or servo motor using directly the PWM digital pin it is important to note the specifications of the load (V and I requirements, e.g, 1000mA, 5V).
- Since USB power source cannot handle more than 250 mA of current, connecting such source to drive a motor can destroy the USB port >> similarly, connecting such source directly to drive Arduino can also destroy the board.
- The voltage to the transistor can be regulated using limiting resistor to base of transistor to control large current demand from motor >> small I is used to regulate large I.

PWM application for dc motor control

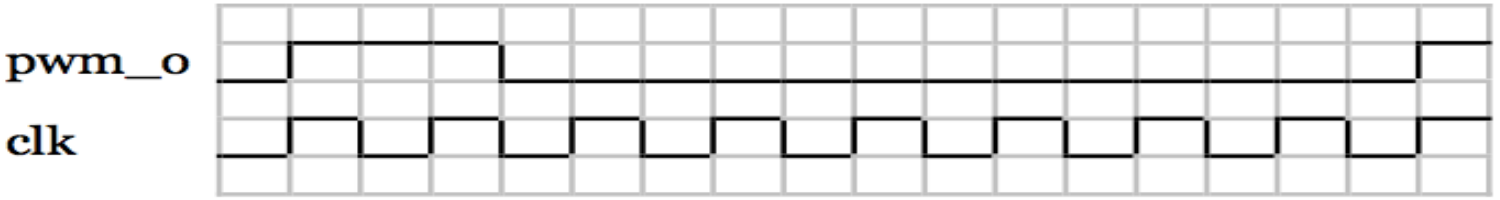
- **Cont'd….**
- To determine value of limiting resistor for the transistor, we can use the basic transistor equation:
  - >> $R_B = (V_B - V_{BE})/I_B$, where $I_B = I_E/\beta$, where $\beta = 100$
  - >> $I_C = I_E$, and $V_{BE} = 0.7V$ and I_C = load I at saturate
- To avoid issues of possible spike current produced when motor is off to avoid damage to transistor or the Arduino, a diode is employed to prevent such effects >> note operation of diodes.

Duty cycle of PWM for system control

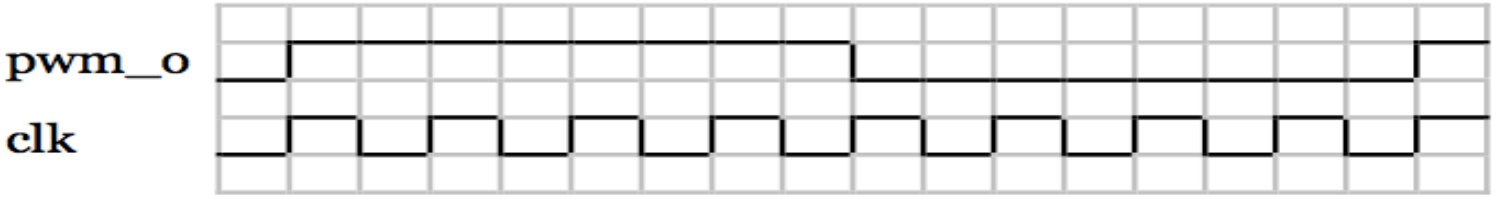
- PWM signal consist of a **period** and **duty cycle**
  - ***Period*** >> time that pulse is both ON and OFF >> **H + L**
    - >> given a signal with 5Hz, the period = 1/5s = 0.2s
      >> within a period, we will have both ON and OFF
  - ***Duty cycle*** >> describes proportion of '**ON**' time to the regular interval or period of time
    - Example, for a given period of 0.2s if we assume 10% ON time, then this will translate to a duty cycle = 0.10 * 0.2s = 0.02s >> 0.2s duration will be ON and the rest of the time will be OFF

Duty cycle of PWM for system control

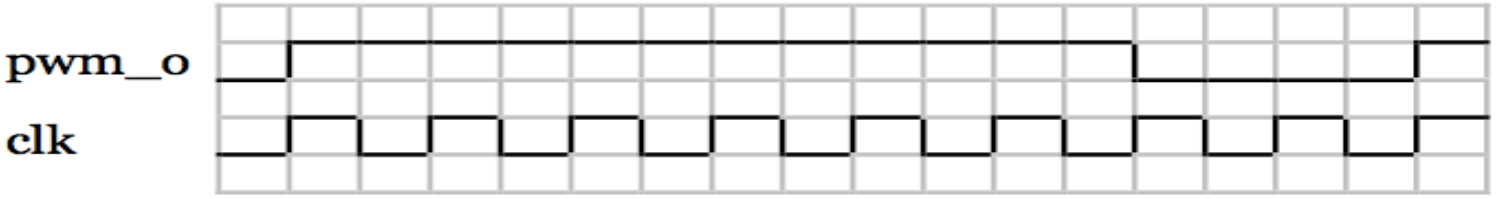
- Cont'd…..



25% duty cycle – average pwm_o is 1.25V



50% duty cycle – average pwm_o is 2.5V.



75% duty cycle – average pwm_o is 3.75V.

Duty cycle of PWM for system control

- **Cont'd…..**
- Suppose a pulse has a low of 0V and a high of 10V, then :
  - 50% duty cycle gives average of 5V, while
  - 10% duty cycle will produce average of 1V.
- A high will lead to high power consumption since we will have maximum voltage whereas a low will lead to no power consumption since voltage will be low (logic 0)
- We interface this digital output to an external actuator (a DC motor or and device) such that power is applied to the device when signal is high and no power is applied when signal is low

Period and Frequency of PWM for control

- We determine power delivered to a hardware device that is connected to PWM using the duty cycle information as:
  - >> delivered power = duty_cycle*Power
  - >> duty_cycle = H/(H+L) >> voltage at port
- Suppose the port P0 of the MCU signal is such that H = 200 and L = 50, then we can find the duty cycle and the power delivered to the connected motor if P = 10W
- The PWM duty cycle is specified by the timer value that is written to the register of the timer as:
  - **>>T_ON = (register value + 1)/F_clock** where F-clock is the system clock

Period and Frequency of PWM for control

- **Cont'd…..**
- PWM period is determined using the value of the period register (PR) of the selected timer_*y* and it is computed as:
  - >> PWM_T = (PR value +1)*TCY* (TMR prescaler_value)
    - PR is PWM period register that decides freq of PWM,
    - TCY is the timer cycle period
    - prescaler value of timer the TMR register
- The PWM frequency is determined using PWM period as :
  - >> PWM_f = 1/PWM_T

Period and Frequency of PWM for control

- **Cont'd…..**
- From the PWM duty cycle specified by the value that is written to the register as:
  - >> T_ON = (register value + 1)/F_clock
- We can determine the PWM resolution (maximum) in bits using the PWM period as :
  - >> PWM_reso = [$\log_{10}$(PWM_T) – log(TCY*prescaler value of TMR_y)] /log(102)

Review of timer count calculations

- In lecture 5 on timers, we indicated that a 16 MHz clock to a MCU essentially means 16,000,000 clock cycles per second
- This means to find the time taken for just one clock cycle (which is clock period), we find 1/16,000,000 = 62.5ns
- If we select a timer with 16 bits, then the maximum number of counts the timer can hold is 2^16 = 65,536 >> note that this maximum value is different from the count number which starts from 0 to (2^16)-1
- To find the maximum delay that could be achieved, all we do is to multiply **62.5ns* 65,536** = 4.096 ms

Review of timer count calculations

- **Cont'd…..**
- If we want to create a delay of 1s, all we have to do is just load the 16,000,000 into the register
- Because the timer will either count from zero upwards in the upward mode and count downwards to zero in the downward mode, we have to subtract 1 from the 16,000,000
- Now, if we want to create a 1ms (0.001s) timer delay, then the number of counts to load to timer register will be calculated:
  - >> 0.001*16,000,000 = 16,000
- For a 32 bit timer mode, then the maximum time will be computed as : 2^32 * 62.5ns = 268.435s

Review of timer count calculations

- **Cont'd…..**
- Now, if we want to calculate the time a timer would take to overflow, we can use the formula:
  - >> Time to overflow = clock frequency/(prescaler + 1) (period + 1)
- If our clock to the MCU is 72 MHz and we want to achieve a time to overflow of 1s, then if we set the prescaler to 1098 and period to 65514 ,we can get the desired time overflow as:
  - >> Time to overflow = 72 000 000/ 71 999 886 = 1s
  - Thus, we get a timer overflow of exactly one second.

PWM calculation example 1

- **Question 1:**
- Suppose we have a DC motor that operates at 10 rev/s when its controlling input voltage is 3.7V. Assume the motor is controlled by a microcontroller with a PWM whose output port can be set to high (5V) or low (0V).
  - (a) find the duty cycle necessary to obtain 10 rev/sec
  - (b) find the width of the pulse and the period required to achieve this duty cycle.

PWM calculation example 1

- **Cont'd…..**
- **Solution 1:**
- (a) duty cycle = Vout/Vin = 3.7V/5.0V = 74%
- (b) duty cycle = t_ON/period
  - pulse width (t_ON) = duty cycle*period
  - If we choose a period = 100ns then the pulse width will be
    - pulse width (t_ON) = 0.74*100ns = 74ns
  - Alternatively, if we select a period = 10ns,
    - pulse width (t_ON) = 0.74*10ns = 7.4ns
  - Infinitely many answers, we thus select any one value for the design implementation

PWM calculation example 2

- **Question 2 :**
- A DC motor at a plant is connected to port P0 of a MCU that has two 16-bit timers. The MCU is to monitor the RPM operations of the motor. The timer count terminal is connected to the motor and is pulsed once for each revolution. The clock input to the MCU is a 10MHz clock oscillator. Find the minimum and maximum RPM that can be measured by the controller.

PWM calculation example 2

- **Cont'd…..**
- **Solution 2 :**
- First, we find the pulse period from the system clock
  - period = 1/f = 1/10MHz = 10^{-7}s
  - We then convert the period to mins = 1.667 x 10^{-7} min
- We determine the maximum time range
  - >> maximum time range = 2^{16}*period
  - >> = 65536*10^{-7} = 1.092 x 10^{-4} min
- We find the minimum RPM as:
  - >> min_rpm = 1 rev/(max range)
  - >> = 1/ 1.092 x 10^{-4} min = 9155.3

PWM calculation example 2

- **Cont'd…..**
- We find the maximum RPM as :
  - >> max_rpm = (max revs)/(1 resolution)
  - >> $= 2^{16} / 1.66 \times 10^{-7}$ min $= 3.93 \times 10^{11}$
- Thus, the microcontroller can measure a RPM ranging from 9155 to 3.93×10^{11}

PWM calculation example 3

- **Question 3 :**
- Using PWM example in Question 1 above, find the value that will be assigned to the PWM terminal of the microcontroller to achieve an RPM of 8,050 assuming the input voltage necessary to achieve this RPM is 4.375V. Assume a count value of 254 (highest 8 bits minus two = 2^8 - 2)

PWM calculation example 3

- **Cont'd…..**
- **Solution 3 :**
- Using input voltage of 4.375 will translate to a duty cycle of:
  - duty cycle = 4.373/5 = 87.5%
  - count value given = 254
  - PWM = duty cycle x count value = 87.5%*254 = 222
- Thus, PWM value required for entering into the period register (in hex) = 0 x CD

PWM calculation example 4

- **Question 4 :**
- A microcontroller is driven by a 10MHz clock. If the timer 2 pre-scalar has a setting of 4,
- (a) calculate PWM period and frequency for the 16 bit period register (max value of period register PR = 0 x FFFF = 65535 = 2^{16} -1)
- (b) calculate the maximum resolution at a PWM frequency of 48 Hz

PWM calculation example 4

- **Cont'd…..**
- **Solution 4 :**
- Given information:
  - Clock frequency = 10 MHz;
  - timer prescaler setting = 4;
  - period register resolution = 16 bits
- PWM period is calculated using formula as:
  - PWM_T = (PR value +1)*TCY*(TMR prescaler_value)
  - PWM_T = (65535+1)*(1/10MHz)*(4) = 26.21ms

PWM calculation example 4

- **Cont'd…..**
  - >> PWM_F = 1/PWM_T = 38.14Hz
  - (b) Maximum resolution for 48Hz PWM frequency is calculated using formula as:
  - >> PWM_max = [$\log_{10}$(PWM_T) – $\log_{10}$(TCY *prescale value TMR)]/$\log_{10}$(102)
  - >> PWM_max = [log(1/48) – log(1/10MHz * 4)]/log(102) = 15.66 bits

PWM calculation example 5

- **Question 5 :**
- Suppose we have a 20MHz clock connected to a 10-bit microcontroller. Find the number or the count value that will need to be put into the period register if a duty cycle of 25% is to be achieved for a PWM frequency of 78.125KHz

PWM calculation example 5

- **Cont'd…..**
- **Solution 5 :**
- At PWM of 78.125kHz, the number of counts for the PWM period will be:
  - >> PWM_count = 20,000,000/78125
  - >> = 256 = 0 x 100
- Highest 8 bits minus one (256 – 1 = 255) will be put into PR
  - >> because (PR+1) = (PWM period)/4
- The 8 highest bits = 0 x 40
- >> we should put 0 x 3F into the PR >> will be left with 2 lower bits for the 10-bit counter

PWM calculation example 5

- **Cont'd…..**
- For a duty cycle of 25%, the number of count will be estimated as:
  - >> 256*25% = 64 = 0 x 40
- Thus, the number that has to be put into the register is 0 x 40, which will contain the 8 highest bits and the two lower bits will be contained by other elements

PWM calculation example 6

- **Question 6 :**
- Suppose we have a 1MHz clock that is driving a 10-bit microcontroller. Assume the value in the PR is 999 which gives 1 kHz rate. Find the T_ON (on time) for the 1KHz if the period register has a value of 399.
- **Solution 6 :**
- The on time T_ON is computed for the period:
  - >> T_ON = (PR + 1)/F_clock
  - >> = (399+1)/1kHz = 400/1000 = 40%

PWM calculation example 7

- **Question 7 :**
- Suppose we have a 20MHz clock connected to a 10-bit microcontroller. Find the value that must be put into the PWM period register for a duty cycle of 25% if the PWM frequency is made 4 times smaller the original i.e., 19. 5KHz