

## Subject: EEC242\_Logic cct design

### Section (2 & 6)

#### Team members:

Name	Id
Ahmed Mohamed Ahmed Abogabl	18010200
Fatma Ezzat Abdelaziz	18011232
Alia Alaa Eldine metwally	18011093
Asser Mohamed Atef Zayed	18010299
Ahmed Mohamed Ahmed Abdelgawad	18010198

## PWM circuit project in VHDL

Introduction ....

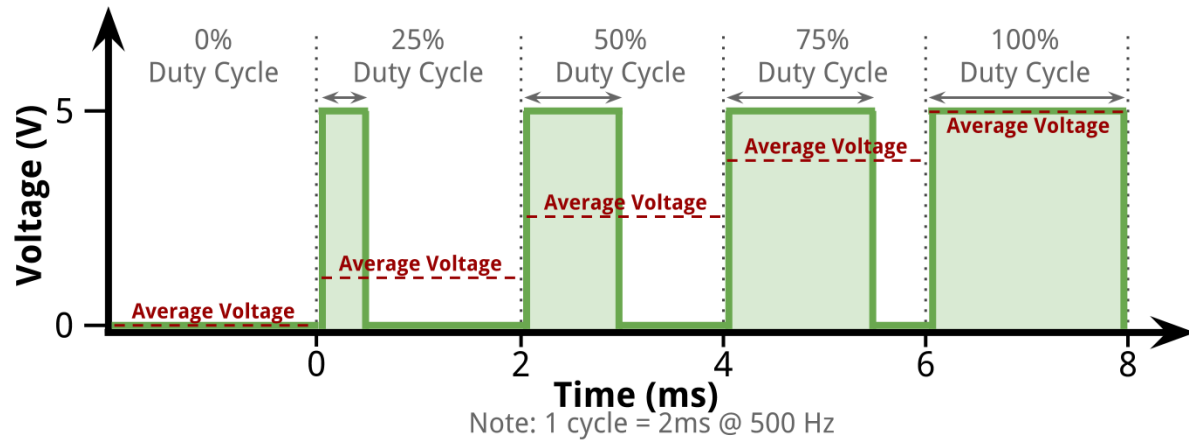
**PWM** is a technique that reduces the average amount of deliverable power of an applied electrical signal. Moreover, the process is achieved by effectively chopping up the signal into distinct parts.

PWM achieves this control by controlling the average current and voltage it delivers to the load. This method is accomplished by rapidly turning the switch between the load and the source, **on** and **off**.

It is used in a variety of applications like controlling the direction of servo, controlling speed of motor and generating analog signal.

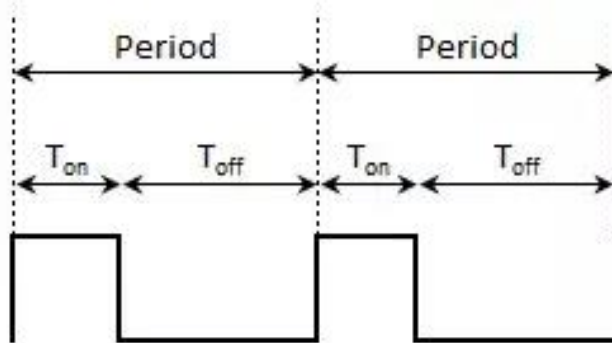
**Pulse width modulation** allows us to vary how much time the signal is high in an analog fashion. While the signal can only be high (usually 5V) or low (ground) at any time, we can change the proportion of time the signal is high compared to when it is low over a consistent time interval.

## Pulse Width Modulation Duty Cycles



**Duty cycle** is the ratio of time a load or circuit is ON compared to the time the load or circuit is OFF. and can be calculated with this formula

**100%** duty cycle would be the same as setting the voltage to 5 Volts (high). **0%** duty cycle would be the same as grounding the signal.



$$\text{Period} = 1 / \text{Frequency}$$

$$\text{Period} = T_{on} + T_{off}$$

$$\text{Duty Cycle} = T_{on} / (T_{on} + T_{off}) * 100$$

(On Percentage)

A period is equal to the time this signal completes a one-and-off cycle.

The frequency is the number of times a periodic change is completed per unit time. It controls the speed at which the signal switches between high and low states. If we turn the digital signal on and off repeatedly with a high enough frequency, the output will behave like an analogue signal with a constant voltage.

## Requirements and steps:

- 1) we want to adjust ( pwm reversed counter ) to control the duty cycle  
Whether  $1/16$  ,  $2/16$  ,..... ,  $15/16$  ,  $16/16$   
Check the clock counter if it is ending or the pwm counter (out\_tmp) is unassigned so we set the pwm counter to be as the duty cycle (w)  
Then start a comparison between pwm counter (out\_tmp) & "0000".
  - If its larger than "0000" then decrease pwm counter(out\_tmp) by 1 and set the output to be = 1
  - But if the pwm counter reaches "0000" cycle set the output to be = 0
    - ( **special case** ) when duty cycle (w) = '0000'  
Assign pwm counter(out\_tmp) with any value except "0000"  
set the output to (1)
- 2) The clk counter and the pwm counter (out\_tmp) must works with each others as to check clock the period and the input duty cycle and determine the pwm output based on them
- 3) This output is the square wave of period 16 times the clock period and with the adjustable duty cycle.

## The vhdl code :

### Counter vhdl code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
    port(Clock : in std_logic;           -- the clock
          Q    : out std_logic_vector(3 downto 0)); -- the clock counter value
end entity;

architecture behavior of counter is

    signal tmp: std_logic_vector(3 downto 0) := "1111"; -- store the clock counter value

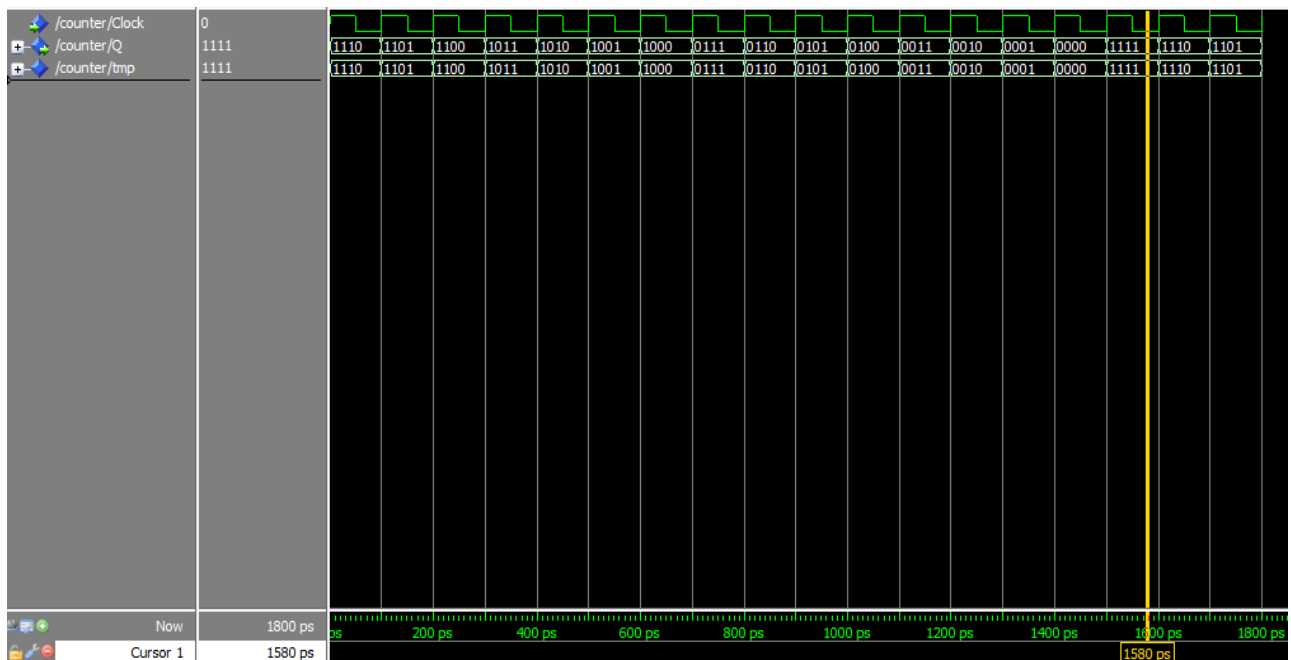
begin
    process (Clock)
    begin
        if (Clock'event and Clock = '1') then
            if(tmp > "0000" ) then
                tmp <= tmp - 1; -- decrease tmp by one
            else tmp <= "1111"; -- reset tmp or initiate it
            end if;
        end if;
    end process;
    Q <= tmp;
end architecture;
```

```

1. library ieee;
2. use ieee.std_logic_1164.all;
3. use ieee.std_logic_unsigned.all;
4.
5. entity counter is
6. port(Clock : in std_logic; -- the clock
7. Q : out std_logic_vector(3 downto 0)); -- the clock counter value
8. end entity;
9.
10. architecture behavior of counter is
11.
12. signal tmp: std_logic_vector(3 downto 0):="1111"; -- store the clock
    counter value
13.
14. begin
15. process (Clock)
16. begin
17. if (Clock'event and Clock = '1') then
18. if(tmp > "0000" ) then
19. tmp <= tmp - 1; -- decrease tmp by one
20. else tmp <="1111"; -- reset tmp or initiate it
21. end if;
22. end if;
23. end process;
24. Q <= tmp;
25. end architecture;

```

in this we count down the cycle every rise edge from “1111” till “0000” then it rests again



[Main vhdl code for PWM:](#)

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4 use ieee.std_logic_arith.all;
5
6 entity pwm is
7     port(clk      : in std_logic;          -- the clock
8           w        : in std_logic_vector(3 downto 0); -- the signal that control the PWM
9           out_pulse : out std_logic);      -- the PWM
10 end entity;
11
12 architecture behaviour of PWM is
13
14     signal out_period : std_logic_vector(3 downto 0); -- to save the counter value
15     signal out_tmp     : std_logic_vector(3 downto 0); -- to save the w and valuse less than w
16
17     component counter is -- clock counter
18         port(Clock : in std_logic; Q : out std_logic_vector(3 downto 0));
19     end component;
20
21 begin
22
23     stage0: counter port map(clk,out_period(3 downto 0)); -- gives out_period the clock counter value
24
25     process(clk,out_period)
26     begin
27         if(clk'event and clk = '1')then -- check rising edge of the clock
28
29             if(w = "0000") then -- assign out_tmp with a bit '1' in the four bits if the w "0000" => the special case
30                 out_tmp <= "1111"; -- note you can change "1111" with any four bits as long it has a bit with '1'
31
32             elsif(out_tmp > "0000") then -- if out_tmp gratear than w , it decreases the out_tmp with '1' , note: it doesnt work first step
33                 out_tmp <= out_tmp - 1;
34
35             elsif(out_period="1111" or out_tmp="0000") then -- initialise out_tmp at the beginning and end of each cycle
36                 out_tmp<=w;
37
38             end if;
39         end if;
40
41         if(out_tmp(3)='1' or out_tmp(2)='1' or out_tmp(1)='1' or out_tmp(0)='1') then -- if there is a single bit from the for
42             out_pulse <= '1'; -- bits in w with '1' it gives a signal '1'
43         else out_pulse <= '0'; -- if all '0' it gives a '0'
44         end if;
45
46     end process;
47 end architecture;

```

```

1. library ieee;
2. use ieee.std_logic_1164.all;
3. use ieee.std_logic_unsigned.all;
4. use ieee.std_logic_arith.all;
5.
6. entity pwm is
7. port(clk : in std_logic; -- the clock
8. w : in std_logic_vector(3 downto 0); -- the signal that control the PWM
9. out_pulse : out std_logic); -- the PWM
10. end entity;
11.
12. architecture behaviour of PWM is
13.
14. signal out_period : std_logic_vector(3 downto 0); -- to save the
   counter value
15. signal out_tmp : std_logic_vector(3 downto 0); -- to save the w and
   valuse less than w
16.
17. component counter is -- clock counter
18. port(Clock : in std_logic; Q : out std_logic_vector(3 downto 0));
19. end component;
20.

```

```

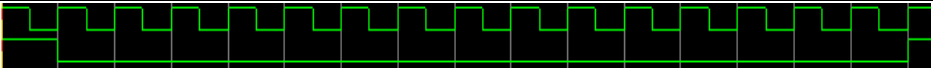
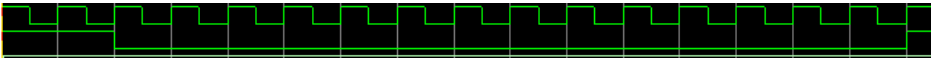






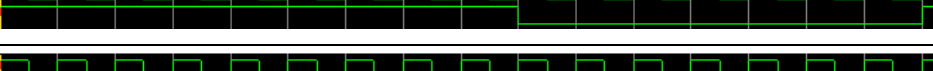



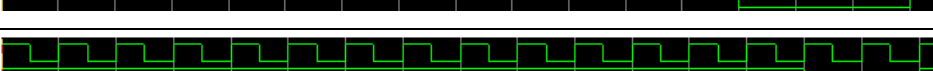


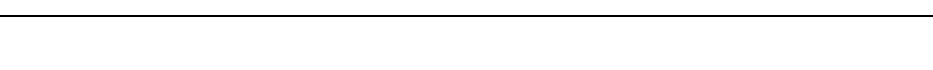
21. begin
22.
23. stage0: counter port map(clk,out_period(3 downto 0)); -- gives
    out_period the clock counter value
24.
25. process(clk,out_period)
26. begin
27. if(clk'event and clk = '1')then -- check rising edge of the clock
28.
29. if(w = "0000") then -- assign out_tmp with a bit '1' in the four bits
    if the w "0000" => the special case
30. out_tmp <= "1111"; -- note you can change "1111" with any four bits as
    long it has a bit with '1'
31.
32. elsif(out_tmp > "0000") then -- if out_temp gratear than w , it
    decreases the out_temp with '1' , note: it doesnt work first step
33. out_tmp <= out_tmp - 1;
34.
35. elsif(out_period="1111" or out_tmp/="0000") then -- initialise out_tmp
    at the beginning and end of each cycle
36. out_tmp<=w;
37.
38. end if;
39. end if;
40.
41. if(out_tmp(3)='1' or out_tmp(2)='1' or out_tmp(1)='1' or
    out_tmp(0)='1') then -- if there is a single bit from the for
42. out_pulse <= '1'; -- bits in w with '1' it gives a signal '1'
43. else out_pulse <='0'; -- if all '0' it gives a '0'
44. end if;
45.
46. end process;
47. end architecture;

```

### **Inputs and outputs :**

clk	input	std_logic
w	input	Std_logic_vector(3 downto 0)
out_pulse	output	std_logic

**W** is the signal that control the PWM output as following :

w		Duty cycle%	output
0001	1/16	6.25%	
0010	2/16	12.5%	
0011	3/16	18.75%	
0100	4/16	25%	
0101	5/16	31.25%	
0110	6/16	37.5%	
0111	7/16	43.75%	
1000	8/16	50%	
1001	9/16	56.25%	
1010	10/16	62.5%	
1011	11/16	68.75%	
1100	12/16	75%	
1101	13/16	81.25%	
1110	14/16	87.5%	
1111	15/16	93.75%	
0000	16/16	100%	

The main idea of the code that there is a counter that keep counting and give a 1 as output signal till reaching the input duty cycle. When it reaches the duty cycle it gives 0 signal for the rest of the 16 bits.

### Explanation of PWM vhdl code:

The main idea is to output a signal '1' when one bit or more from out\_temp which starts from the duty cycle entered and kepp decreasing till "0000" has a vlue '1' , when out\_temp becomes "0000" it output a signal '0'

This part of the code is the last if in the process

```
if(out_tmp(3)='1' or out_tmp(2)='1' or out_tmp(1)='1' or out_tmp(0)='1') then -- if there is a single bit from the for
    out_pulse <= '1'; -- bits in w with '1' it gives a signal '1'
else out_pulse <='0'; -- if all '0' it gives a '0'
end if;
```

1. if(out\_tmp(3)='1' or out\_tmp(2)='1' or out\_tmp(1)='1' or out\_tmp(0)='1')
- then -- if there is a single bit from the for
2. out\_pulse <= '1'; -- bits in w with '1' it gives a signal '1'
3. else out\_pulse <='0'; -- if all '0' it gives a '0'
4. end if;

the next part controls the out\_tmp

```
if(clk'event and clk = '1')then -- check rising edge of the clock

    if(w = "0000") then -- assign out_tmp with a bit '1' in the four bits if the w "0000" => the special case
        out_tmp <= "1111"; -- note you can change "1111" with any four bits as long it has a bit with '1'

    elsif(out_tmp > "0000") then -- if out_tmp greater than w , it decreases the out_tmp with '1' , note: it doesnt work first step
        out_tmp <= out_tmp - 1;

    elsif(out_period="1111" or out_tmp/="0000") then -- initialise out_tmp at the beginning and end of each cycle
        out_tmp<=w;

    end if;
end if;
```

1. if(clk'event and clk = '1')then -- check rising edge of the clock
- 2.
3. if(w = "0000") then -- assign out\_tmp with a bit '1' in the four bits if the w "0000" => the special case
4. out\_tmp <= "1111"; -- note you can change "1111" with any four bits as long it has a bit with '1'
- 5.
6. elsif(out\_tmp > "0000") then -- if out\_tmp greater than w , it decreases the out\_tmp with '1' , note: it doesnt work first step
7. out\_tmp <= out\_tmp - 1;
- 8.
9. elsif(out\_period="1111" or out\_tmp/="0000") then -- initialise out\_tmp at the beginning and end of each cycle
10. out\_tmp<=w;
- 11.
12. end if;
13. end if;

- In the first if we check the special case which is duty cycle "0000" , in this case the out put always a '1' signal then we need to assign out\_tmp with any value except "0000" so it has at least a single '1' bit which lead to out put a signal '1'
- In the second if we check if the out\_tmp (after it was assigned in the next if) is bigger than "0000" , if its bigger we decrease it with one and produce a '1' signal and keep doing till it reaches "0000" then it stays at "0000" so it keep producing '0' signal till the period finishes



- In the third if it checks if we are at the end of the period of the PWM or if we are at first step of generation and out\_temp needs to be assigned so it assigns it (this case happens at the first of the code and the last step of PWM period)

## Hardware implementation

