BAB 5

SEVEN SEGMENT & PUSH BUTTON

Pada bab ini, praktikan akan mempelajari konsep dasar seven segment display dan push button sebagai elemen penting dalam sistem digital. Seven segment display digunakan untuk menampilkan informasi numerik atau karakter, sementara push button berfungsi sebagai input sederhana yang dapat mengubah kondisi atau memicu suatu aksi pada rangkaian. Asisten praktikum atau praktikan diharapkan membaca tujuan dan persyaratan bab ini agar praktikum dapat berjalan sesuai dengan prosedur.

Tujuan

Tujuan	Penjelasan
Memahami Konsep Seven Segment Display	Praktikan diharapkan dapat memahami cara kerja dan fungsi seven segment display dalam menampilkan angka atau karakter, serta bagaimana mengontrol tiap segmen secara digital.
Mengimplementasikan Seven Segment pada Rangkaian Digital	Praktikan diharapkan mampu menghubungkan dan memprogram seven segment display untuk menampilkan angka atau karakter yang diinginkan dalam sebuah sistem digital.
Memahami Fungsi Push Button	Praktikan diharapkan dapat memahami mekanisme kerja push button sebagai input digital dan bagaimana push button digunakan dalam sistem digital untuk memicu suatu tindakan.
Mengintegrasikan Push Button dengan Seven Segment	Praktikan diharapkan dapat merancang dan mengimplementasikan rangkaian yang menggabungkan push button dan seven segment display, seperti menampilkan angka atau menubah tampilan berdasarkan input dari push button.

Persyaratan

Disarankan praktikan menggunakan hardware dan software sesuai pada dokumentasi ini. Apabila terdapat versi hardware atau software yang cukup lama dari versi yang direkomendasikan maka sebaiknya bertanya kepada Asisten Mengajar Shift.

HARDWARE YANG DIBUTUHKAN PRAKTIKUM

PC / Laptop

FPGA Board Nexys A7 dan Nexys 4

Kabel Power USB

SOFTWARE YANG DIBUTUHKAN PRAKTIKUM

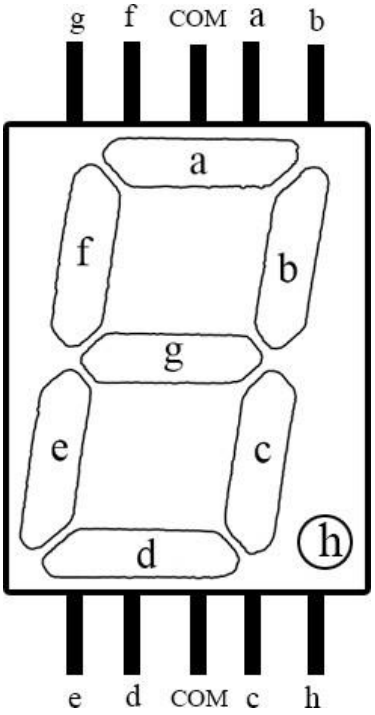
Vivado Design Suite

⚠ Diusahakan untuk memakai **Versi** dan **Aplikasi** yang sama agar tidak terjadinya kesalahan yang tidak diinginkan!

Teori

5.1. Seven Segment

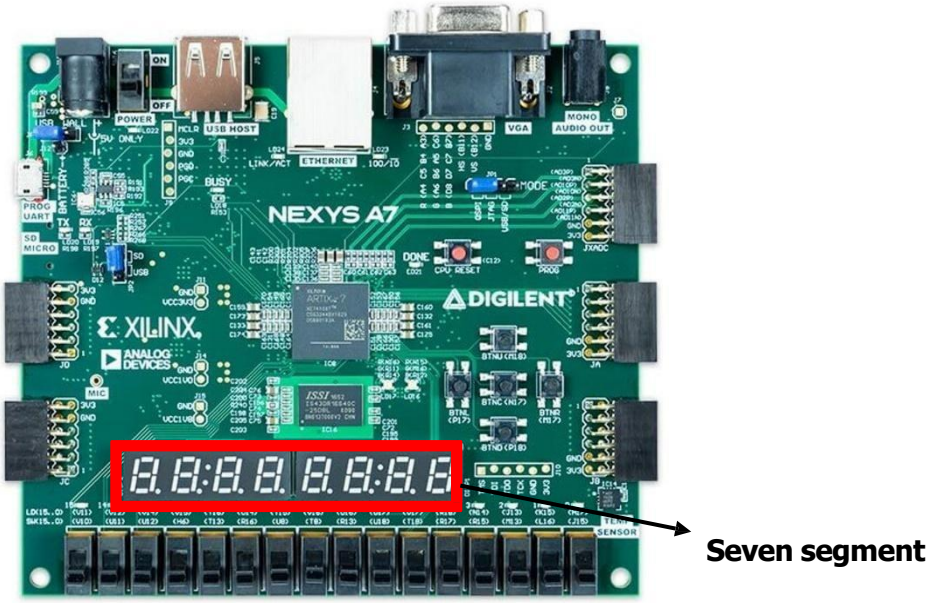
Seven Segment adalah suatu tampilan digital yang terdiri dari tujuh segmen (bagian) LED (*Light Emitting Diode*) yang dapat dinyalakan secara individu atau kombinasi untuk menampilkan angka dari 0 hingga 9, serta beberapa huruf seperti A, B, C, E, F. Tampilan ini sangat umum digunakan pada jam digital, kalkulator, dan berbagai perangkat elektronik lainnya.



Gambar 6. 1 Tampilan Seven-Segment

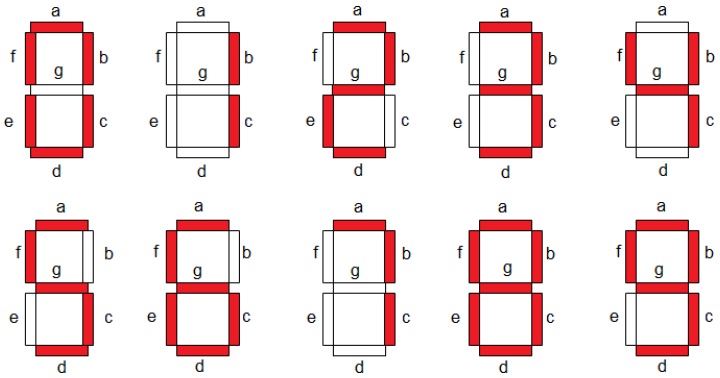
Dalam konteks FPGA, *Seven Segment* dapat dikendalikan secara penuh melalui pemrograman logika. FPGA memberikan fleksibilitas yang sangat tinggi untuk merancang dan mengimplementasikan berbagai macam sistem digital, termasuk tampilan *Seven Segment*. *Seven Segment* merupakan salah satu perangkat layar untuk menampilkan sistem angka decimal yang merupakan alternatif dari layar *dot-matrix*. *Seven Segment* ini seringkali digunakan pada jam digital, meteran elektronik, dan perangkat elektronik lainnya yang menampilkan informasi numerik. Ide mengenai *Seven Segment* ini sudah cukup tua, misalnya pada tahun 1910, dimana sudah terdapat layar tujuh segmen yang diterangi oleh lampu pijar yang digunakan pada panel sinyal kamar ketel suatu pembangkit listrik.

Tujuh bagian dari layar dapat dinyalakan dalam bermacam-macam kombinasi untuk menampilkan angka. Seringkali ketujuh segmen tersebut disusun dengan kemiringan tertentu untuk memudahkan pembacaan.



Gambar 6. 2 Seven Segment pada Nexys A7

Seven Segment memiliki 7 segmen dimana setiap segmen dikendalikan secara ON dan OFF untuk menampilkan angka yang diinginkan. Angka-angka dari 0 (nol) sampai 9 (sembilan) dapat ditampilkan dengan menggunakan beberapa kombinasi segmen. Selain 0 – 9, *Seven Segment* juga dapat menampilkan huruf heksadesimal dari A sampai F. Segmen atau elemen-elemen pada *Seven Segment* diatur menjadi bentuk angka "8" yang agak miring ke kanan dengan tujuan untuk mempermudah pembacaannya. Pada beberapa jenis *Seven Segment*, terdapat juga penambahan "titik" yang menunjukan angka koma desimal. Terdapat beberapa jenis *Seven Segment*, diantaranya adalah *Incandescent bulbs*, *Fluorescent lamps* (FL), *Liquid Crystal Display* (LCD), *dan Light Emitting Diode* (LED).



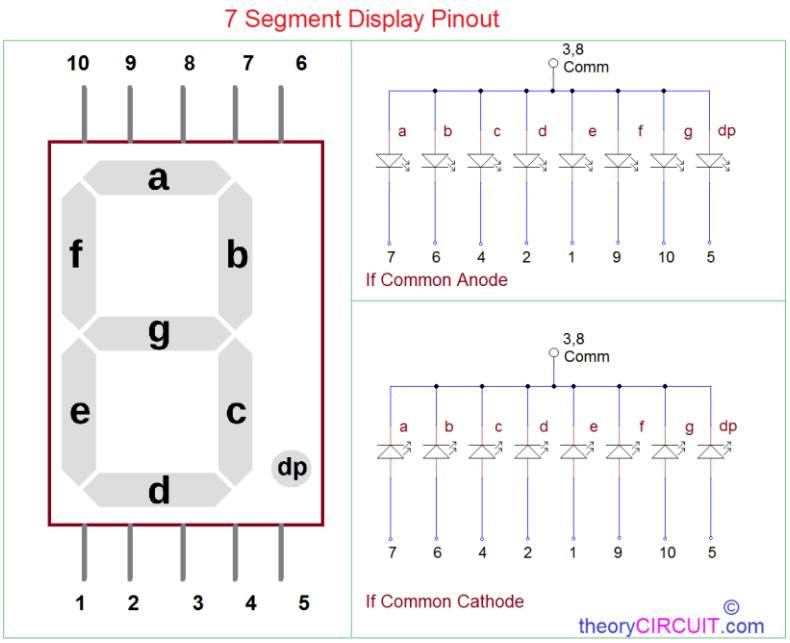
Gambar 6. 3 Tampilan Angka Seven Segment

5.1.1. LED Seven Segment

LED *Seven Segment* umumnya memiliki 7 segmen atau elemen garis dan 1 segmen titik yang menandakan 'koma' sebagai desimal atau *dot product*. Jumlah keseluruhan segmen atau elemen LED sebenarnya terdapat 8.

5.1.2. Cara Kerja Seven Segment

Setiap segmen LED pada *Seven Segment* memiliki pin yang terhubung dengan rangkaian pengendali yang berfungsi untuk menentukan segmen mana yang akan menyala dan mana yang akan mati. Dengan mengkombinasikan segmen-segmen yang menyala, pengguna bisa membentuk berbagai macam angka dan huruf. Terdapat 2 jenis LED *Seven Segment*, diantaranya adalah LED *Seven Segment Common Cathoda* dan LED *Seven Segment Common Anoda*.



Gambar 6. 4 Common Anoda dan Common Katoda pada Seven Segment

5.1.2.1. Common Katoda

Kaki katoda pada semua segmen LED adalah terhubung menjadi 1 pin, sedangkan kaki anoda akan menjadi *input* untuk masing – masing segmen LED. Kaki katoda yang terhubung menjadi 1 pin ini merupakan terminal negatif (-) atau *ground*, sedangkan sinyal kendali akan diberikan kepada masing-masing kaki anoda Segmen LED. Cara kerja *Seven Segment Common* Katoda adalah aktif pada kondisi *high* '1' untuk menyalakan sebuah segmen.

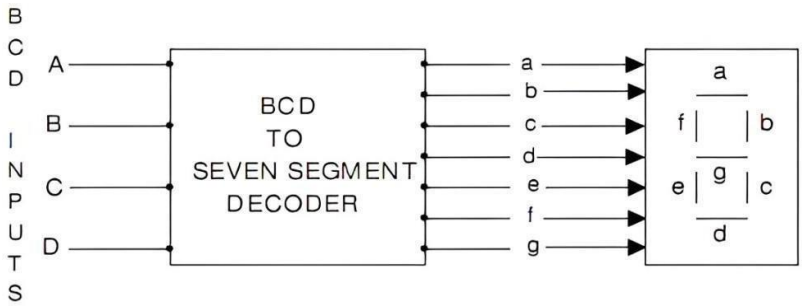
ANGKA	h	g	f	e	d	c	b	a	HEXA
0	0	0	1	1	1	1	1	1	3FH
1	0	0	0	0	0	1	1	0	06H
2	0	1	0	1	1	0	1	1	5BH
3	0	1	0	0	1	1	1	1	4FH
4	0	1	1	0	0	1	1	0	66H
5	0	1	1	0	1	1	0	1	6DH
6	0	1	1	1	1	1	0	1	7DH
7	0	0	0	0	0	1	1	1	07H
8	0	1	1	1	1	1	1	1	7FH
9	0	1	1	0	1	1	1	1	6FH

5.1.2.2. Common Anoda

Kaki anoda pada semua segmen LED adalah terhubung menjadi 1 pin, sedangkan kaki katoda akan menjadi input untuk masing-masing segmen LED. Kaki anoda yang terhubung menjadi 1 pin ini akan diberi tegangan positif (+) dan sinyal kendali akan diberkan pada masing-masing kaki katoda segmen LED. Cara kerja *Seven Segment Common* Anoda adalah aktif pada kondisi *low* '0' untuk menyalakan sebuah segmen.

ANGKA	h	g	f	e	d	c	b	a	HEXA
0	1	1	0	0	0	0	0	0	C0H
1	1	1	1	1	1	0	0	1	F9H
2	1	0	1	0	0	1	0	0	A4H
3	1	0	1	1	0	0	0	0	B0H
4	1	0	0	1	1	0	0	1	99H
5	1	0	0	1	0	0	1	0	EDH
6	1	0	0	0	0	0	1	0	12H
7	1	1	1	1	1	0	0	0	F8H
8	1	0	0	0	0	0	0	0	10H
9	1	0	0	1	0	0	0	0	90H

5.1.3. Prinsip Kerja Dasar Driver System Pada LED Seven Segment



Gambar 6. 5 Skematik Seven Segment

Blok Dekoder pada skematik diatas mengubah sinyal *input* yang diberikan menjadi 8 jalur yaitu A sampai G dan *dot product* (koma) untuk meng- ON-kan segmen sehingga menghasilkan angka atau digit yang diinginkan. Contohnya, jika *output decoder* adalah a, b, dan c, maka segmen LED akan menghasilkan angka '7'. Jika sinyal *input* adalah berbentuk analog, maka diperlukan ADC (*Analog to Digital Converter*) untuk mengubah sinyal analog menjadi sinyal digital sebelum masuk ke input *decoder*. Jika sinyal *input* sudah merupakan sinyal digital, maka *decoder* akan menanganinya sendiri tanpa harus menggunakan ADC.

Fungsi daripada *Blok Driver* adalah untuk memberikan arus listrik yang cukup kepada segmen LED untuk menyala. Pada tipe *decoder* tertentu, *decoder* dapat mengeluarkan tegangan dan arus listrik yang cukup untuk menyalakan segmen LED, maka *blok driver* ini tidak diperlukan. Pada umunya, *driver* untuk menyalakan 7 segmen ini adalah terdiri dari 8 transistor *switch* pada masing- masing elemen LED.

5.2. Komponen-komponen Seven Segment

Dalam konteks *Seven Segment* yang dikendalikan oleh FPGA, terdapat komponen-komponen *Seven Segment* yang memainkan peran penting dalam mengatur tampilan angka dan waktu.

5.2.1. Clock

Clock merupakan sinyal listrik yang berupa suatu denyutan dan berfungsi untuk mengkoordinasikan atau mensikronkan setiap aksi-aksi atau proses- proses

yang dilakukan oleh setiap komponen di dalam perangkat elektronika. Bagaimana proses A, bagaimana proses B, bagaimana proses C, dan seterusnya. Oleh karena itu, nilai *clock* sangat penting agar perangkat elektronika dapat berfungsi sebagaimana mestinya. Ada beberapa istilah penting yang berkaitan dengan clock, yaitu :

- ***Cycle***, yaitu satuan yang digunakan untuk menandakan selesainya satu siklus *clock* mulai dari denyutan dikeluarkan kemudian naik hingga nilainya mencapai 1 lalu mulai turun hingga nilainya 0.
- ***Cycle Time*** (T), adalah jumlah waktu yang diperlukan oleh sinyal *clock* untuk menyelesaikan satu siklus *clock*.
- ***Raise Time***, adalah waktu yang dibutuhkan untuk perubahan nilai *clock* dari 0 ke 1.
- ***Fall Time***, adalah waktu yang dibutuhkan untuk perubahan nilai clock dari 1 ke 0.
- ***Clock Frequency*** (F), yaitu besaran untuk menilai kemampuan suatu sinyal *clock* dalam menciptakan satu siklus denyutan setiap detiknya atau dapat disebut dengan beberapa banyak *cycle* yang dapat dihasilkan oleh sinyal *clock* dalam satuan detik. Sesuai standar internasional, satuan yang digunakan untuk mengukurnya adalah Hertz = Hz, dimana Hz sama dengan satu *cycle* per detik.

Sebagai contoh, jika sinyal *clock* membutuhkan waktu 10ms dalam menyelesaikan satu siklus denyutan (*cycle*), maka *clock frequency* nya = 1 / 0,01 = 100 Hz = 0,1 KHz.

$$F = \frac{1}{T} \; atau \; T = \frac{1}{F}$$

5.2.2. Timer dan Counter

Timer dan Counter merupakan fitur yang telah tertanam pada mikrokontroler yang memiliki fungsi terhadap waktu. Fungsi pewaktu yang dimaksud disini adalah penentuan kapan program tersebut dijalankan. Selain itu, fungsi *Timer* yang lainnya dapat ditemukan pada PWM, ADC dan *oscillator*. Prinsip kerja *Timer* dengan cara membagi frekuensi (*prescaler*) pada *clock* yang terdapat pada FPGA sehingga *Timer* dapat berjalan sesuai dengan frekuensi yang dikehendaki.

Timer merupakan fungsi waktu yang sumber *clock*-nya berasal dari *clock* internal. Sedangkan, *Counter* merupakan fungsi perhitungan yang berasal dari *clock* internal tersebut maupun eksternal. Salah satu contoh penggunaan *Timer* yaitu pada jam digital yang sumber *clock*-nya bisa menggunakan *crystal oscillator*. Sedangkan, contoh penggunaan counter pada penghitung barang konveyor yang sumber *clock*-nya berasal dari sensor yang mendeteksi barang tersebut.

Berikut ini merupakan contoh penggunaan *Timer*.

- Melaksanakan tugas secara berulang.
- Mengendalikan kecepatan motor DC (PWM).
- Mengendalikan perhitungan (*counter*).
- Membuat penundaan waktu (*delay*).

5.2.3. Prescaler

Pada dasarnya *Timer* hanya menghitung denyutan *clock*. Frekuensi *clock* yang dihitung tersebut bisa sama dengan frekuensi *crystal* yang digunakan atau dapat diperlambat menggunakan *prescaler* dengan factor 8, 64, 256, 1024.

Contoh :

Suatu mikrokontroler menggunakan *crystal* dengan frekuensi 8MHz dan timer yang digunakan adalah 16 bit. Maka, maksimum waktu *timer* yang dapat dihasilkan adalah:

Diketahui :

- Fclk (frequency clock) = 8 MHz
- FFFh (timer yang digunakan) = 16 bit
- Prescaler = 1024

Jawab :

$$Tmax = \frac{1}{Fclk}\ x\ FFFH$$

$$Tmax = \frac{1}{8}\ x\ 2^{16}$$

$$Tmax\ = 0.125\mu\ x\ 65536$$

$$Tmax\ = 0.008192\ second$$

Untuk menghasilkan *timer* yang lebih lama, kita dapat menggunakan *prescaler*. Misalnya kita menggunakan 1024, maka waktu timer yang dapat dihasilkan adalah

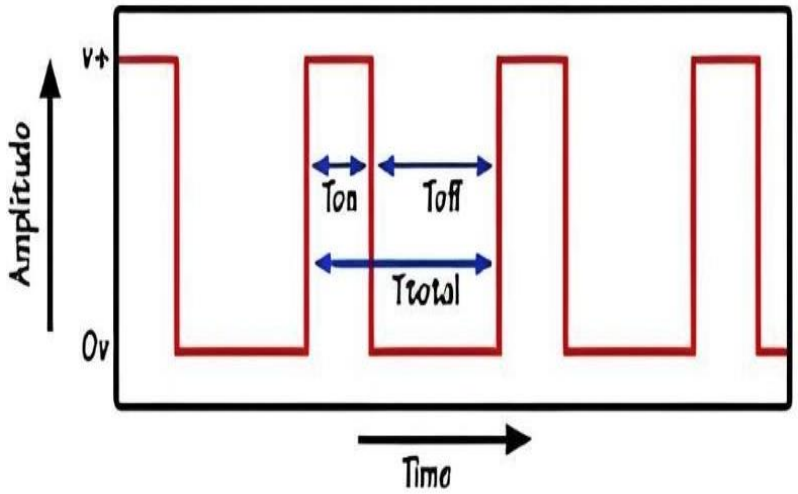
$$Tmax = \frac{1}{Fclk}\ x\ FFFH\ x\ N$$

$$Tmax\ = 0.125\ \mu s\ \text{x}\ 65536\ \text{x}\ 1024$$

$$Tmax\ = 8.388735\ second$$

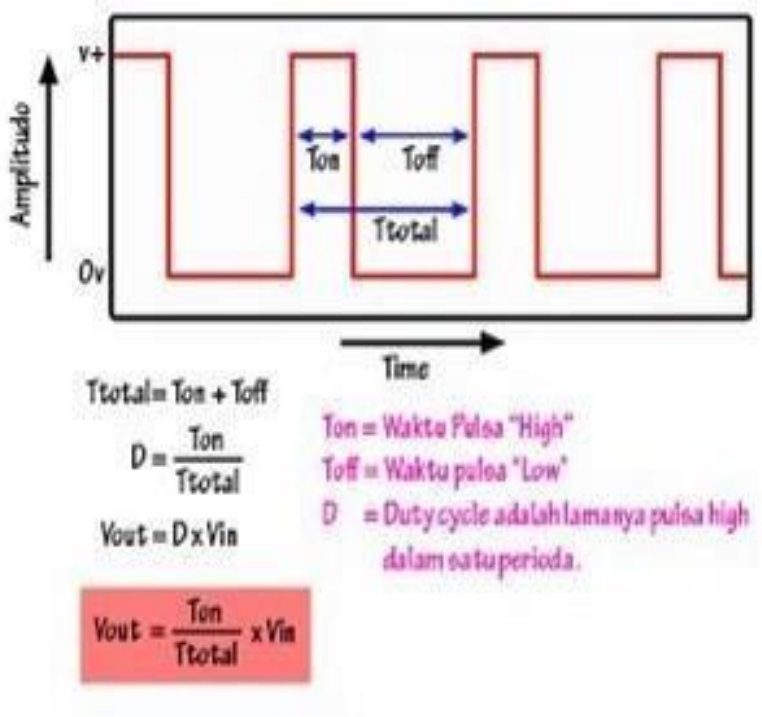
5.2.4. Pulse Width Modulation (PWM)

PWM (*Pulse Width Modulation*) secara umum adalah sebuah cara untuk memanipulasi lebar sinyal yang dinyatakan dengan pulsa dalam satu periode untuk mendapatkan tegangan rata-rata yang berbeda. Penggunaan PWM ini menggantikan output analog karena keterbatasan pada sebuah IC umumnya hanya dapat mengeluarkan sinyal digital. Beberapa contoh aplikasi PWM adalah pada kipas angin, AC, dll.



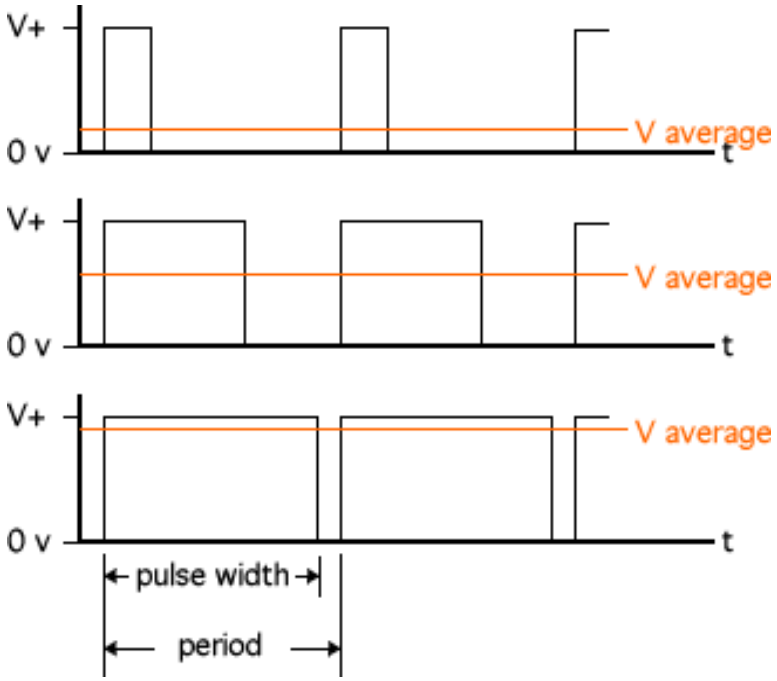
Gambar 6. 6 Grafik PWM

Sinyal PWM umumnya memiliki amplitudo dan frekuensi dasar yang tetap, namun memiliki lebar pulsa yang bervariasi. Lebar pulsa PWM berbanding lurus dengan amplitudo sinyal asli yang belum termodulasi. Artinya, sinyal PWM memiliki frekuensi gelombang yang tetap namun *duty cycle* yang bervariasi antara 0-100%.



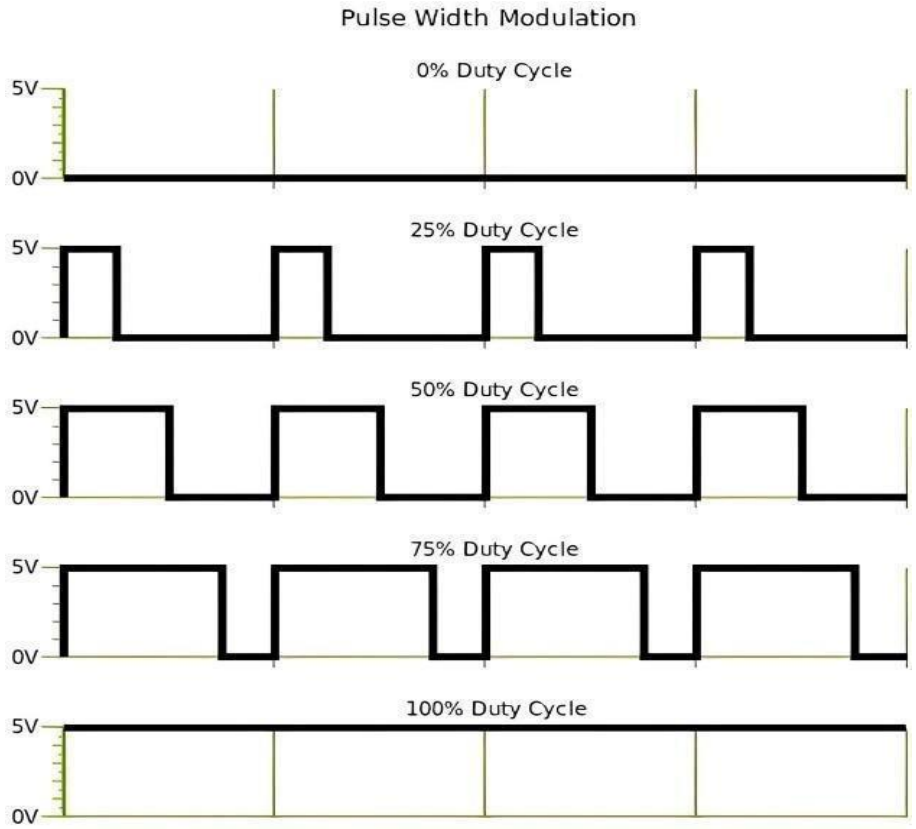
Gambar 6. 7 Perhitungan Vout PWM

Dari persamaan diatas, diketahui bahwa perubahan *duty cycle* akan merubah tegangan *output* atau tegangan rata-rata seperti gambar dibawah ini.



Gambar 6. 8 Hasil Vout Berdasarkan Panjang Pulsa

Perubahan *duty cycle* akan merubah tegangan *output* atau tegangan rata-rata, untuk mengatur nilai *duty cycle* pada timer 8 bit, nilai pada parameter berkisar antara 0 hingga 255. Bila kita hendak mengatur *duty cycle* ke 0%, maka kita dapat mengatur nilai parameter ke 0 dan untuk duty cycle nya 100%. Dari sini kita akan mendapatkan set nilai dengan parameter ke 255. Sementara itu, jika kita ingin mengatur duty cycle ke 50%, maka nilai yang harus kita atur adalah 127 (50% x 255). Nilai-nilai inilah yang nantinya akan kita gunakan untuk mengatur panjang lebar pulsa tersebut.



Gambar 6. 9 Grafik Duty Cycle PWM

5.3. Kode Program Seven Segment

- Menampilkan Angka pada Seven Segment

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sevseg1 is
    Port ( clk : in STD_LOGIC;
           pos_seg : out STD_LOGIC_VECTOR (7 downto 0);
           led_seg : out STD_LOGIC_VECTOR (7 downto 0));
end sevseg1;

architecture Behavioral of sevseg1 is

signal delay : natural range 0 to 1000 := 0;
signal number : natural range 0 to 7 := 0;

begin
process (clk)
begin
if rising_edge(clk) then
   delay <= delay + 1;
   if delay >= 1000 then
      delay <= 0;

      number <= number + 1;
      if number >= 7 then
         number <= 0;
      end if;
   end if;
end if;
end process;
```

```
process (number)
    begin
        case number is
         when 0 => pos_seg <="01111111";
         when 1 => pos_seg <="10111111";
         when 2 => pos_seg <="11011111";
         when 3 => pos_seg <="11101111";
         when 4 => pos_seg <="11110111";
         when 5 => pos_seg <="11111011";
         when 6 => pos_seg <="11111101";
         when 7 => pos_seg <="11111110";
    end case;
end process;


process (number)
    begin
        case number is        --hgfedcba
         when 0 => led_seg <= "11111001"; --1
         when 1 => led_seg <= "10100100"; --2
         when 2 => led_seg <= "10110000"; --3
         when 3 => led_seg <= "10011001"; --4
         when 4 => led_seg <= "10010010"; --5
         when 5 => led_seg <= "00000010"; --6
         when 6 => led_seg <= "11111000"; --7
         when 7 => led_seg <= "00000000"; --8
        end case;
end process;

end Behavioral;
```

- **Port yang digunakan pada I/O Planning**

All ports (17)											
led_seg (8)	OUT			✓	(Multiple)	LVCMOS18	1.800	12		NONE	FP_VTT_50
led_seg[7]	OUT		H15	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
led_seg[6]	OUT		L18	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
led_seg[5]	OUT		T11	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
led_seg[4]	OUT		P15	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
led_seg[3]	OUT		K13	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
led_seg[2]	OUT		K16	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
led_seg[1]	OUT		R10	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
led_seg[0]	OUT		T10	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg (8)	OUT			✓	(Multiple)	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg[7]	OUT		U13	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg[6]	OUT		K2	✓	35	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg[5]	OUT		T14	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg[4]	OUT		P14	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg[3]	OUT		J14	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg[2]	OUT		T9	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg[1]	OUT		J18	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
pos_seg[0]	OUT		J17	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
Scalar ports (1)											
clk	IN		E3	✓	35	LVCMOS18	1.800			NONE	NONE

- **Program Binary to BCD Seven Segment**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sevseg2 is
    Port (
        bcd_input   : in  STD_LOGIC_VECTOR(3 downto 0);  -- Input BCD dari switch
        seg_output  : out STD_LOGIC_VECTOR(6 downto 0);  -- Output ke 7-segment display (aktif low)
        display_sel : out STD_LOGIC_VECTOR(7 downto 0)   -- Selektor aktif low untuk mengaktifkan display (8 posisi)
    );
end sevseg2;

architecture Behavioral of sevseg2 is
begin
    process(bcd_input)
    begin
        -- Menentukan tampilan angka pada seven-segment display
        case bcd_input is
            when "0001" => seg_output <= "1111001";  -- 1 pada seven-segment
            when "0010" => seg_output <= "0100100";  -- 2 pada seven-segment
            when "0011" => seg_output <= "0110000";  -- 3 pada seven-segment
            when "0100" => seg_output <= "0011001";  -- 4 pada seven-segment
            when "0101" => seg_output <= "0010010";  -- 5 pada seven-segment
            when "0110" => seg_output <= "0000010";  -- 6 pada seven-segment
            when "0111" => seg_output <= "1111000";  -- 7 pada seven-segment
            when "1000" => seg_output <= "0000000";  -- 8 pada seven-segment
            when others => seg_output <= "1111111";  -- Default case, tidak menampilkan apa-apa
        end case;
        -- Menentukan posisi seven-segment display berdasarkan input BCD
        case bcd_input is
            when "0001" => display_sel <= "11111110";  -- Aktifkan display pertama (posisi 1)
            when "0010" => display_sel <= "11111101";  -- Aktifkan display kedua (posisi 2)
            when "0011" => display_sel <= "11111011";  -- Aktifkan display ketiga (posisi 3)
            when "0100" => display_sel <= "11110111";  -- Aktifkan display keempat (posisi 4)
            when "0101" => display_sel <= "11101111";  -- Aktifkan display kelima (posisi 5)
            when "0110" => display_sel <= "11011111";  -- Aktifkan display keenam (posisi 6)
            when "0111" => display_sel <= "10111111";  -- Aktifkan display ketujuh (posisi 7)
            when "1000" => display_sel <= "01111111";  -- Aktifkan display kedelapan (posisi 8)
            when others => display_sel <= "11111111";  -- Tidak ada display yang aktif jika BCD tidak valid
        end case;
    end process;
end Behavioral;
```

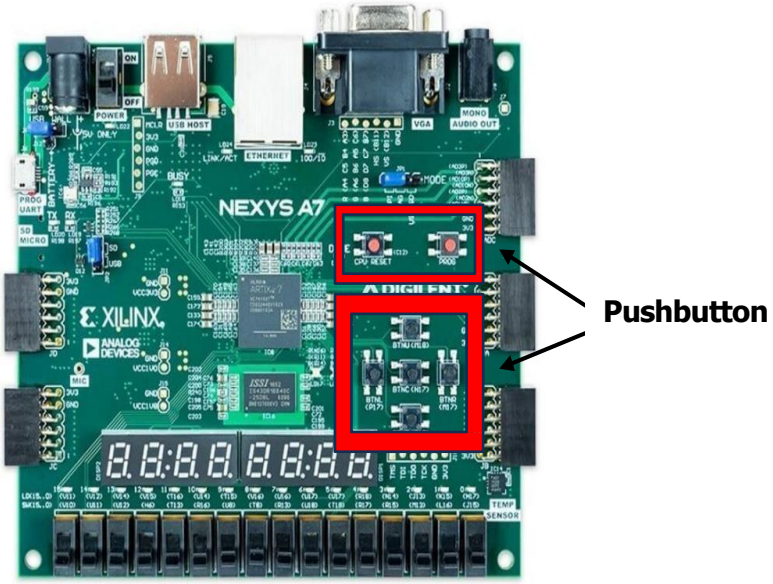
- **Port yang digunakan pada I/O Planning**

All ports (19)											
bcd_input (4)	IN			✓	(Multi	LVCMOS18	1.800				NONE
bcd_input[3]	IN		H6	✓	35	LVCMOS18	1.800				NONE
bcd_input[2]	IN		U12	✓	14	LVCMOS18	1.800				NONE
bcd_input[1]	IN		U11	✓	14	LVCMOS18	1.800				NONE
bcd_input[0]	IN		V10	✓	14	LVCMOS18	1.800				NONE
display_sel (8)	OUT			✓	(Multi	LVCMOS18	1.800	12			NONE
display_sel[7]	OUT		U13	✓	14	LVCMOS18	1.800	12			NONE
display_sel[6]	OUT		K2	✓	35	LVCMOS18	1.800	12			NONE
display_sel[5]	OUT		T14	✓	14	LVCMOS18	1.800	12			NONE
display_sel[4]	OUT		P14	✓	14	LVCMOS18	1.800	12			NONE
display_sel[3]	OUT		J14	✓	15	LVCMOS18	1.800	12			NONE
display_sel[2]	OUT		T9	✓	14	LVCMOS18	1.800	12			NONE
display_sel[1]	OUT		J18	✓	15	LVCMOS18	1.800	12			NONE
display_sel[0]	OUT		J17	✓	15	LVCMOS18	1.800	12			NONE
seg_output (7)	OUT			✓	(Multi	LVCMOS18	1.800	12			NONE
seg_output[6]	OUT		L18	✓	14	LVCMOS18	1.800	12			NONE
seg_output[5]	OUT		T11	✓	14	LVCMOS18	1.800	12			NONE
seg_output[4]	OUT		P15	✓	14	LVCMOS18	1.800	12			NONE
seg_output[3]	OUT		K13	✓	15	LVCMOS18	1.800	12			NONE
seg_output[2]	OUT		K16	✓	15	LVCMOS18	1.800	12			NONE
seg_output[1]	OUT		R10	✓	14	LVCMOS18	1.800	12			NONE
seg_output[0]	OUT		T10	✓	14	LVCMOS18	1.800	12			NONE

5.4. Push Button

Push Button adalah saklar yang beroperasi dengan cara ditekan dan bisa melakukan dua fungsi berbeda, yaitu menutup sirkuit bila ditekan atau justru membuka sirkuit bila ditekan. Jika tekanan dilepaskan atau terjadi tekanan berikutnya maka akan menormalkan kembali tombol ke posisi semula. Secara umum, komponen ini memiliki cara kerja yang sama dengan *switch*, namun hanya berbeda pada cara pengoperasiannya saja.

Komponen ini juga berfungsi sebagai pemberi sinyal pada rangkaian listrik ketika bagian knopnya ditekan, dan alat ini akan bekerja sehingga kontak- kontaknya akan terhubung untuk jenis *Normally Open* (NO) dan akan terlepas untuk jenis *Normally Close* (NC). Pada *board* FPGA Nexys 4 dan A7 terdapat total 7 *push button*, namun yang dapat digunakan hanya 5. Karena dua *push button* yang berwarna merah sudah memiliki fungsi sendiri yaitu untuk CPU reset



Gambar 6. 10 Push Button pada Nexys A7

5.5. Kode Program Push Button

- Program UpDown Counter menggunakan Push Button

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;

entity sevseg3 is
    Port ( clk : in STD_LOGIC;
           count_up : in STD_LOGIC;
           count_down : in STD_LOGIC;
           seg : out std_logic_vector(6 downto 0));
end sevseg3;

architecture Behavioral of sevseg3 is
signal temp_count : std_logic_vector(3 downto 0) :="0000";
signal up_edge : std_logic_vector(1 downto 0);
signal down_edge : std_logic_vector(1 downto 0);

begin

process(clk, temp_count)
begin
if rising_edge(clk) then
    up_edge <= up_edge(0) & count_up;
    down_edge <= down_edge(0) & count_down;
    if (up_edge = "01") then
        if temp_count <9 then
```

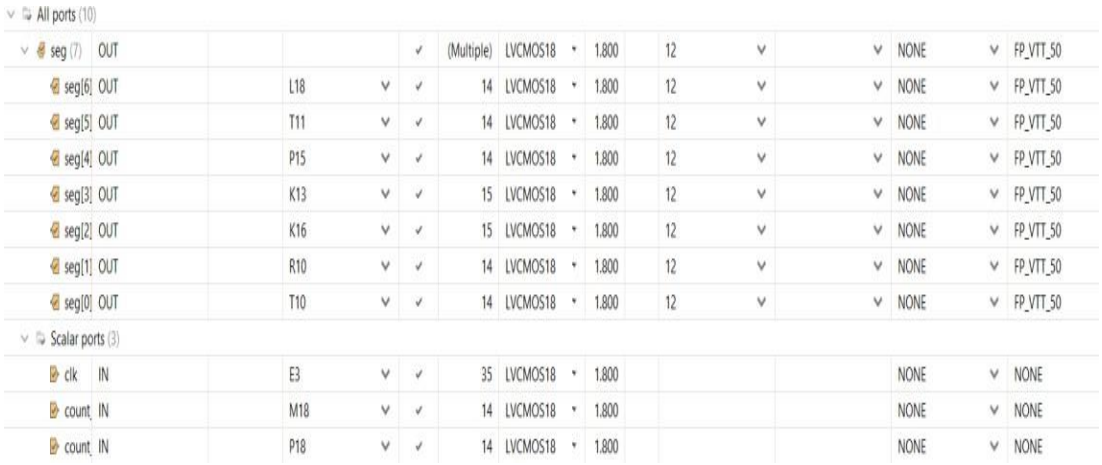
```
            temp_count <= temp_count + 1;
        else
            temp_count <= "0000";
        end if;
else
    if (down_edge = "10") then
        if temp_count >0 then
           temp_count <= temp_count - 1;
        else
           temp_count <= "1001";
        end if;
    end if;
  end if;
end if;
end process;

process (temp_count)
begin
    case temp_count is
        when "0000" => seg <= "1000000";
        when "0001" => seg <= "1111001";
        when "0010" => seg <= "0100100";
        when "0011" => seg <= "0110000";
        when "0100" => seg <= "0011001";
        when "0101" => seg <= "0010010";
        when "0110" => seg <= "0000010";

         when "0111" => seg <= "1111000";
         when "1000" => seg <= "0000000";
         when "1001" => seg <= "0010000";
         when others => seg <= "1111111";
    end case;
end process;

end Behavioral;
```

- **Port yang digunakan pada I/O Planning**



All ports (10)											
seg (7)	OUT			✓	(Multiple)	LVCMOS18	1.800	12		NONE	FP_VTT_50
seg[6]	OUT		L18	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
seg[5]	OUT		T11	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
seg[4]	OUT		P15	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
seg[3]	OUT		K13	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
seg[2]	OUT		K16	✓	15	LVCMOS18	1.800	12		NONE	FP_VTT_50
seg[1]	OUT		R10	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
seg[0]	OUT		T10	✓	14	LVCMOS18	1.800	12		NONE	FP_VTT_50
Scalar ports (3)											
clk	IN		E3	✓	35	LVCMOS18	1.800			NONE	NONE
count	IN		M18	✓	14	LVCMOS18	1.800			NONE	NONE
count	IN		P18	✓	14	LVCMOS18	1.800			NONE	NONE