MAKALAH DIGITAL SYSTEM

Ardhian Mahendar 2301919813

Irvine 2301924990

Kelvin Cendra 2301919460

Raymond Winsher 2301919574

Richie Eviendy 2301919763

AUTOMOTIVE AND ROBOTICS ENGINEERING

BINUS ASO SCHOOL OF ENGINEERING

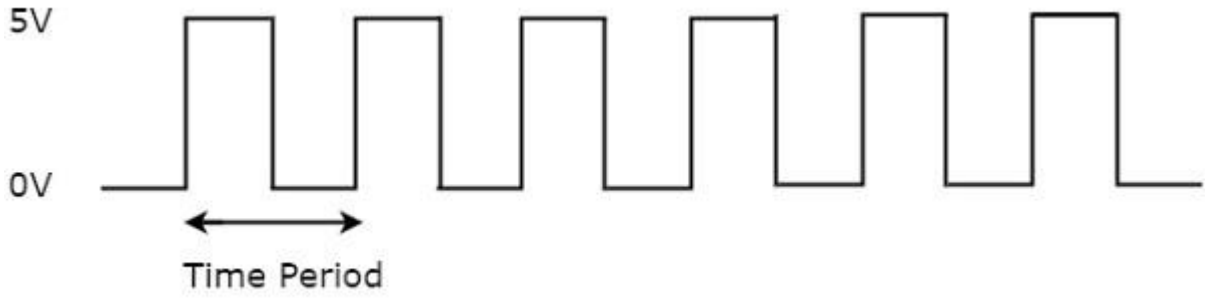
BAB I

LANDASAN TEORI

1.1 Clock Signal

Sinyal clock didefiniskan sebagai jenis sinyal tertentu yang berosilasi antara status tinggi dan rendah. Sinyal bertindak seperti metronom, yang diikuti rangkaian digital pada waktunya untuk mengoordinasikan urutan tindakannya. Sirkuit digital mengandalkan sinyal jam untuk mengetahui kapan dan bagaimana menjalankan fungsi yang diprogram (Wojciechowicz, 2018).

Sinyal clock ini ditunjukkan pada gambar berikut.



Gambar 1.1 Gambar Sinyal Clock (Hub, Electronic, 2019)

Clock speed setara dengan pergerakan data dalam aplikasi yang menerima dan memproses ratusan megabyte data setiap detik. Aplikasi yang terlibat dalam memindahkan volume data yang sangat besar termasuk stasiun pangkalan seluler dan prosesor backhaul, sistem pencitraan medis, dan instrumen uji laboratorium. Aplikasi ini bergantung pada penerima yang dapat mengenali beberapa data dalam satu aliran dan mereka bergantung pada sirkuit waktu presisi untuk mengirim dan mengekstrak sinyal jam utama (Design, Electronic, 2016).

1.2 Clock Generator

Clock generator menggabungkan osilator dengan satu atau lebih PLL, pembagi keluaran, dan buffer keluaran. Clock generator dan buffer jam berguna ketika beberapa frekuensi diperlukan dan IC target berada di board/FPGA yang sama . Dalam beberapa aplikasi, FPGA / ASIC memiliki banyak domain waktu untuk jalur data, bidang kontrol, dan memory controller interface, dan sebagai hasilnya, memerlukan beberapa frekuensi referensi yang unik. Dalam kebanyakan kasus, osilator berada di luar clock generator, meskipun menjadi lebih umum bahwa osilator

digabungkan ke dalam paket yang sama dengan clock generator dalam upaya untuk mengkonsolidasikan tagihan biaya material dan kompleksitas, di samping keuntungan lainnya. Ada berbagai jenis clock generator dan masing-masing dioptimalkan untuk kinerja dan target biaya yang berbeda tergantung pada aplikasinya.

1.3 Comparator Digital

Rangkaian komparator adalah satu jenis untuk menerapkan pada rangkaian kombinasional yang mempunyai fungsi utama membandingkan dua data digital. Hasil perbandingan itu akan mempunyai hasil sama, lebih kecil, atau lebih besar. Komparator banyak digunakan dalam kehidupan sehari-hari seperti pada mesin penyeleksi surat, baik ukuran dimensinya, berat surat, kode area (berdasarkan bar-code), dsb (Trisnawati, 2016).

Dari dua data digital yang hanya terdiri dari 1bit yang dibandingkan, maka akan dapat diperluas menjadi dua data digital yang terdiri dari lebih dari 1 bit seperti dua bit, tiga bit, dst. Ada dua jenis utama dari Digital komparator yaitu:

1. Pembanding Identitas (*Identity Comparator*): Komparator digital yang hanya memiliki satu terminal *output* ketika A=B baik “Tinggi” A=B=1 atau “*Low*” A=B=0

Besaran Komparator (*Magnitude Comparator*) : Komparator digital yang memiliki tiga terminal *output*, masing-masing untuk kesetaraan A = B lebih besar dari , A>B, dan kurang dari A<B (Hutahaean, n.d.)

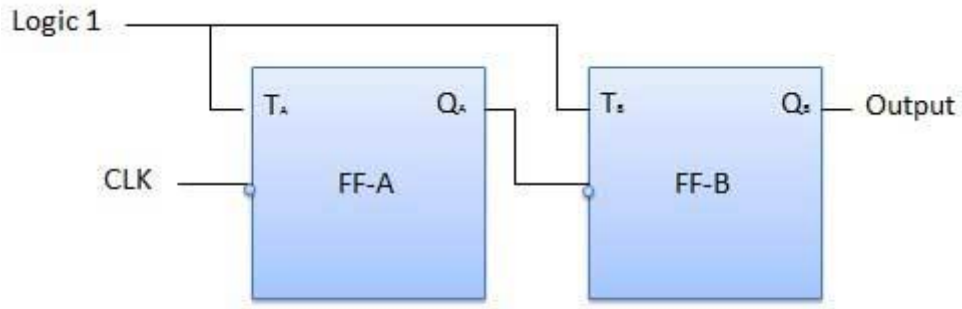
1.4 Counter Digital

Counter merupakan salah satu rangkaian pada elektronika digital yang menggunakan urutan logika digital yang dipicu dengan clock (rangkaian sekuensial). Counter biasanya untuk menghitung dalam biner dan dapat dibuat untuk berhenti dan berulang kembali ke dalam hitungan awal pada setiap saat. Counter yang mengulang, jumlah kondisi biner yang berbeda menunjukkan modulus (MOD) counter. Sebagai contoh, counter yang mencacah dari 0-1-2-3-4-5-6-7 secara berulang disebut juga

modulus 8 atau MOD-8. Rangkaian dasar counter adalah beberapa flip-flop yang jumlahnya bergantung pada modulus yang diperlukan. Secara umum, counter terbagi menjadi 2 jenis, yaitu asynchronous counter (ripple counter) dan synchronous counter.

Synchronous counter merupakan counter yang *input* clock pemicunya dihubungkan pada setiap flip-flop, sehingga setiap flip-flop akan beroperasi dengan masukan transisi clock yang sama. Pada synchronous counter harus dipastikan bahwa setiap flip-flop menahan transisi luarannya sampai tiba gilirannya. Asynchronous counter diagram logika penghitung ripple 2-bit ditunjukkan pada gambar. Toggle (T) flip-flop sedang digunakan. Tetapi kita dapat menggunakan flip-flop JK juga dengan J dan K yang terhubung secara permanen ke logika 1. Clock eksternal diterapkan ke input clock flip-flop A dan output QA diterapkan ke input clock flip-flop berikutnya yaitu FF -B.

Logical Diagram *Asynchronous counter*



Gambar 1.2 Block Diagram *Asynchronous Counter*

Truth Table

Clock	Counter output		State number	Deciimal Counter output
	Q	Q		
Initially	0	0	—	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

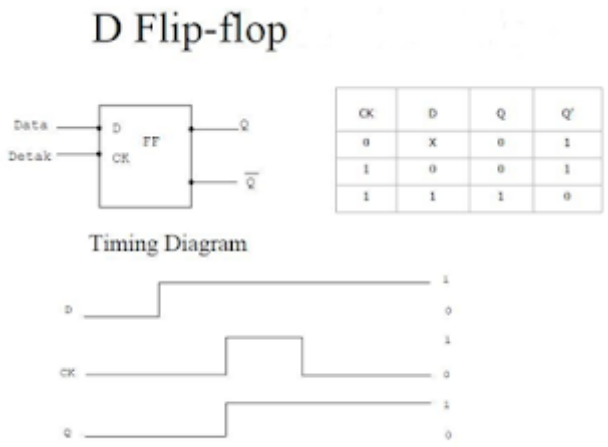
Gambar 1.3 Truth Table *Asynchronous Counter* (, tutorialspoint, n.d.)

1.5 Toggle flip flop

D Flip flop merupakan suatu jenis flip flop yang dibangun dengan menggunakan flip flop RS. D merupakan singkatan dari "data", flip-flop ini menyimpan nilai yang ada pada jalur data. Ini dapat dianggap sebagai sel memori dasar (D Flip-Flops, 2021).

Flip-flop D dapat dibuat dari flip-flop set/reset dengan mengikat set ke reset melalui inverter. Hasilnya dapat di-clock. Flip Flop RS memiliki 2 masukan R dan S, tetapi pada D flip-flop terdapat perbedaan yang hanya memiliki hanya 1 masukan yaitu D.

D flip-flop bisa diaplikasikan untuk berbagai macam peralatan elektronik dan berbagai bidang. Beberapa bidang tersebut yang memanfaatkan D flip-flop pada aplikasinya adalah pengiriman daya, infrastruktur jaringan, medis, kesehatan, penerangan, sistem otomasi gedung, infrastruktur komunikasi, dan pengukuran. Berikut adalah data sheet dari D Flip-flop:



Gambar 1.4 Timing Diagram *D – Flip Flop(* (Widianto, 2019)

Flip Flop D bertindak sebagai komponen memori elektronik karena *output* konstan ,kecuali ketika sengaja diubah dengan mengubah status *input* D diikuti oleh sinyal clock yang meningkat dan D Flip Flop adalah register geser blok bangunan. Misalnya, dengan mengalirkan delapan D Flip Flops secara berurutan, satu byte (8-bit) informasi dapat disimpan setelah 8 siklus clock.

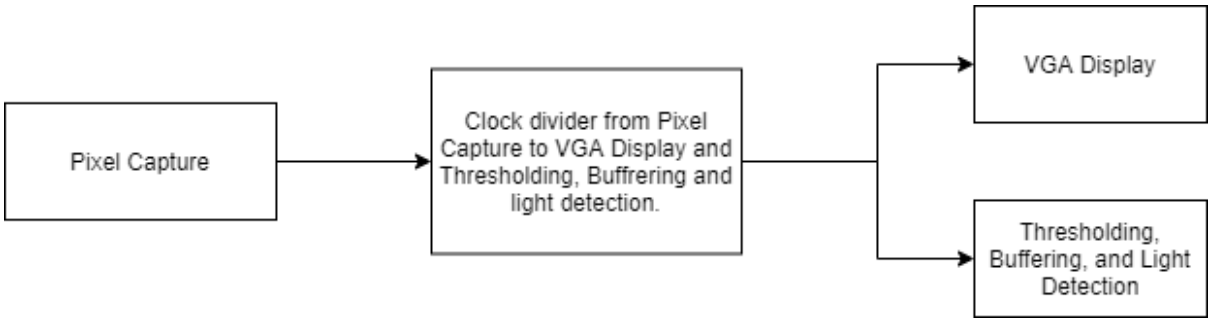
BAB II

PERANCANGAN SISTEM

2.1 BLOK DIAGRAM

a) Percobaan Pertama

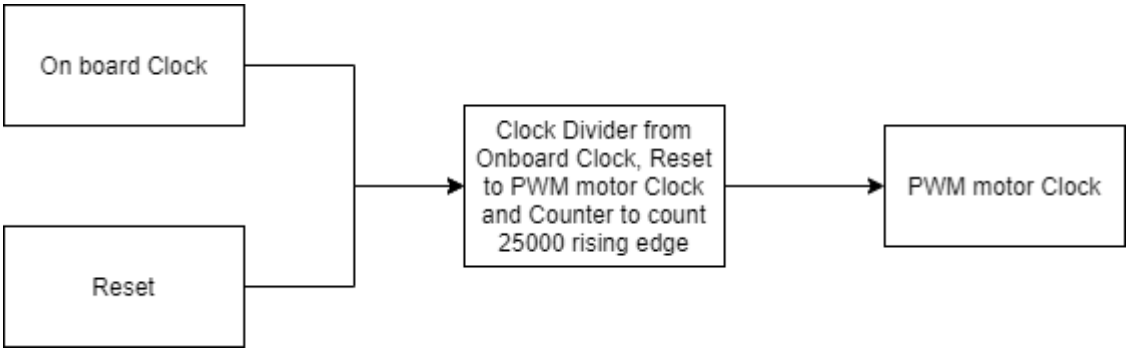
1. Penghubung *Pixel Capture* dengan VGA *Display*, dan *Thresholding Buffering* dan *Light Detection*.



Gambar 2.1 Blok diagram penghubung *Pixel Capture* dengan VGA *Display*, dan *Thresholding Buffering* dan *Light Detection* (percobaan pertama)

Pada gambar 2.1 merupakan blok diagram yang menjelaskan mengenai cara kerja dari *coding clock divider* pada percobaan pertama yang menghubungkan *Pixel Capture* dengan *VGA Display*, dan *Thresholding Buffering dan Light Detection.* Blok diagram ini akan mendapatkan input yang berupa frekuensi dari pixel capture yang memiliki frekuensi sebesar 50 MHz, dan suatu sinyal logic camera control status yang mengindikasikan bahwa Pixel Capture sudah selesai mengkonfigurasi camera OV6760. Pada blok diagram ini akan mengeluarkan output berupa clock dengan frekuensi sebesar 25 MHz yang akan digunakan oleh *VGA Display*, dan *Thresholding Buffering dan Light Detection.*

2. Pembuatan Clock untuk PWM pada Motor

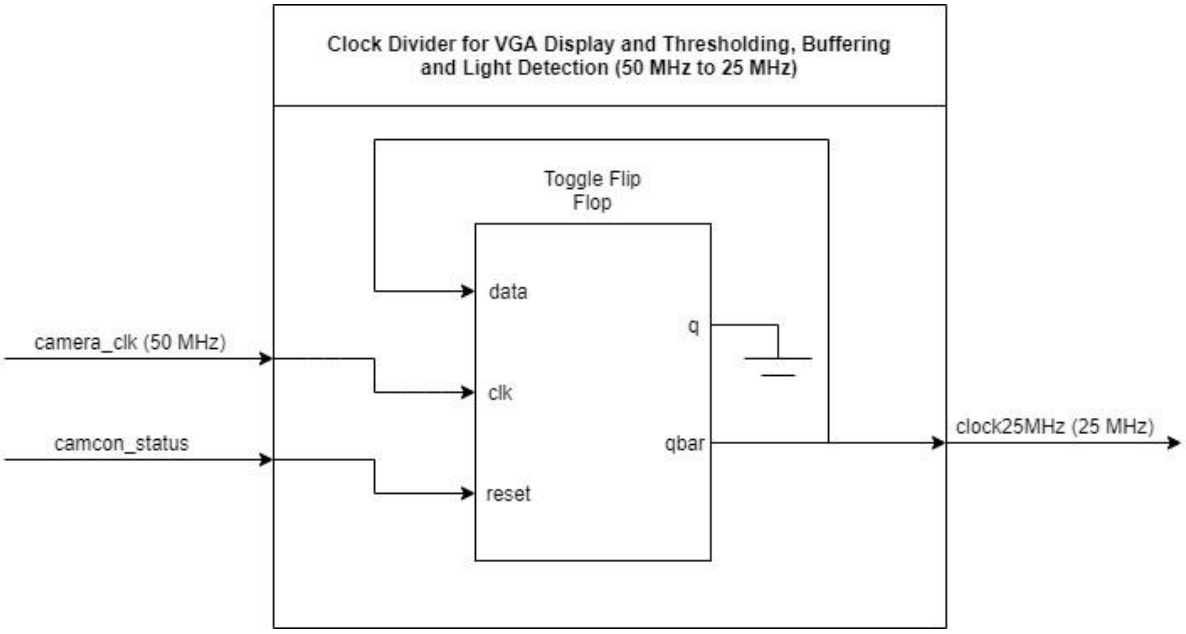


Gambar 2.2 Blok Diagram Pembuatan Clock untuk PWM pada Motor (percobaan pertama)

Pada gambar 2.2 merupakan blok diagram yang menjelaskan mengenai cara kerja dalam pembuatan *clock* untuk PWM pada *motor driver* pada percobaan pertama. Pada blok diagram ini akan didapatkan 2 buah input yaitu *on board clock*, dan *reset*, dan 1 buah output yaitu PWM *motor clock*. Cara kerjanya akan dihitung berapa banyak *rising edge* di *on board clock*, berdasarkan jumlah *rising edge* dapat dibentuk *clock* dengan frekuensi yang lebih rendah dari inputnya. Frekuensi *on board clock* yang didapatkan 100 MHz sedangkan frekuensi pada PWM motor clock yang didapatkan sebesar 2 KHz.

b) Percobaan Kedua

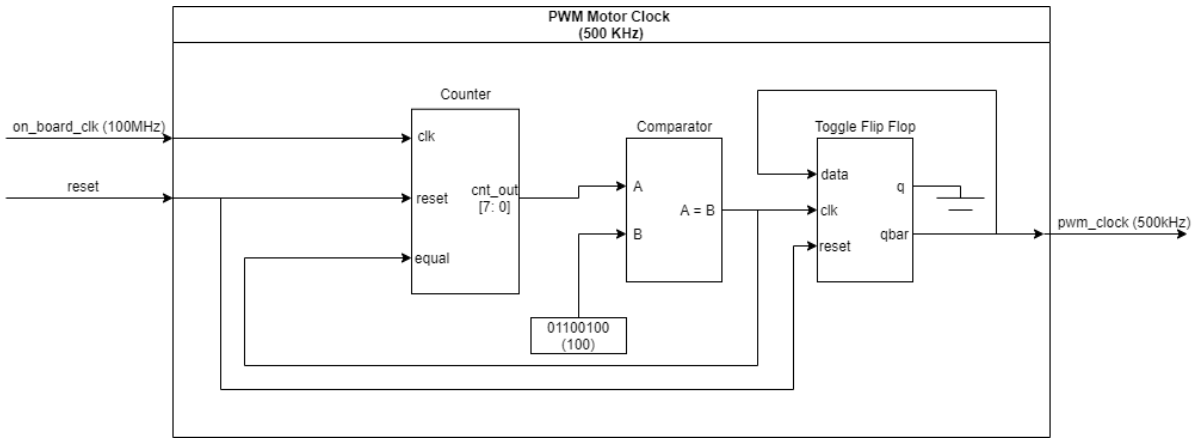
1. Penghubung *Pixel Capture* dengan VGA *Display*, dan *Thresholding Buffering* dan *Light Detection*



Gambar 2.3 Blok diagram penghubung *Pixel Capture* dengan VGA *Display*, dan *Thresholding Buffering* dan *Light Detection* (percobaan kedua)

Pada gambar 2.3 merupakan blok diagram yang menjelaskan mengenai cara kerja dari *coding clock divider* pada percobaan kedua yang menghubungkan *Pixel Capture* dengan *VGA Display*, dan *Thresholding Buffering dan Light Detection.* Blok diagram ini akan mendapatkan input yang berupa frekuensi dari pixel capture yang memiliki frekuensi sebesar 50 MHz, dan suatu sinyal logic camera control status yang mengindikasikan bahwa Pixel Capture sudah selesai mengkonfigurasi hasil settingan camera OV6760. Pada blok diagram ini akan mengeluarkan output berupa clock dengan frekuensi sebesar 25 MHz yang akan digunakan oleh *VGA Display*, dan *Thresholding Buffering dan Light Detection.* Perbedaan dari percobaan pertama dengan percobaan kedua adalah terdapat penambahan port yang dapat berfungsi dalam penggunaan alat elektronik digital (*low level*).

2. Pembuatan Clock untuk PWM pada Motor



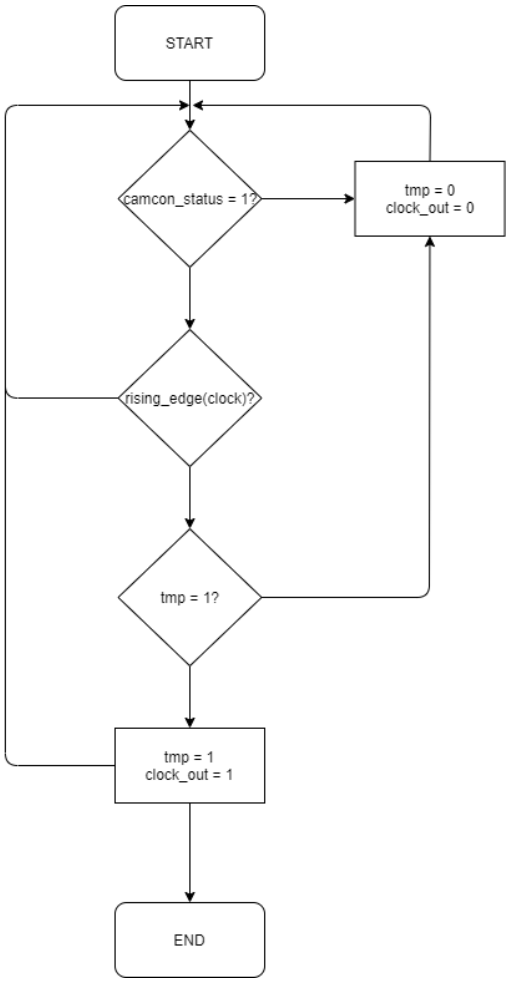
Gambar 2.4 Blok Diagram Pembuatan Clock untuk PWM pada Motor (percobaan kedua)

Pada gambar 2.4 merupakan blok diagram yang menjelaskan mengenai cara kerja dalam pembuatan *clock* untuk PWM pada *motor driver* pada percobaan kedua. Pada blok diagram ini akan didapatkan 2 buah input yaitu *on board clock*, dan *reset*, dan 1 buah output yaitu PWM *motor clock*. Cara kerjanya akan dihitung berapa banyak *rising edge* di *on board clock*, berdasarkan jumlah *rising edge* dapat dibentuk *clock* dengan frekuensi yang lebih rendah dari inputnya. Frekuensi *on board clock* yang didapatkan 100 MHz sedangkan frekuensi pada PWM motor clock yang didapatkan sebesar 2 KHz. Perbedaan pertama dan kedua dari pembuatan *clock* untuk PWM pada *motor driver* ini adalah penambahan komponen yaitu *counter*, *comparator* dan *toggle flip flop*, hal ini dilakukan agar percobaan ini dapat berfungsi dalam penggunaan alat elektronik digital (*low level*).

2.2 FLOW CHART

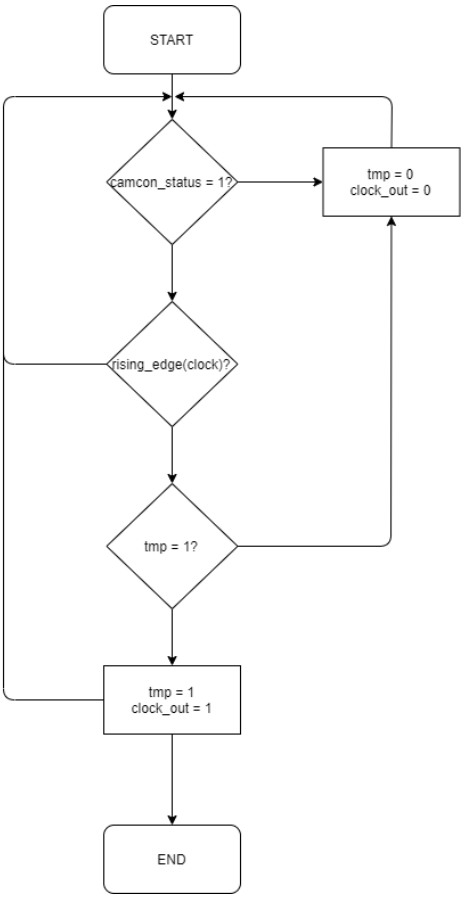
a) Percobaan Pertama

1. Penghubung *Pixel Capture* dengan VGA *Display*, dan *Thresholding Buffering* dan *Light Detection*.



Gambar 2.5 Flow Chart Penghubung *Pixel Capture* dengan VGA *Display*, dan *Thresholding Buffering* dan *Light Detection*

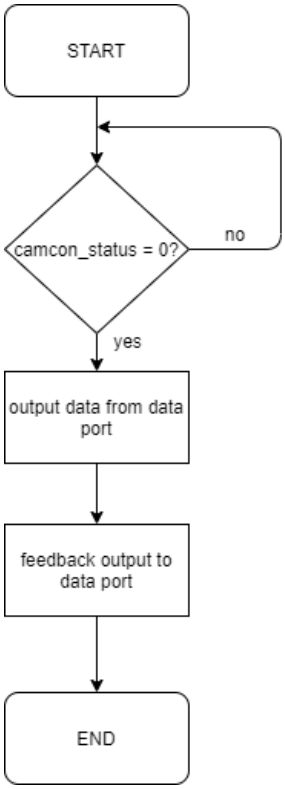
2. Pembuatan Clock untuk PWM pada Motor



Gambar 2.6 Flow Chart Pembuatan Clock untuk PWM pada Motor

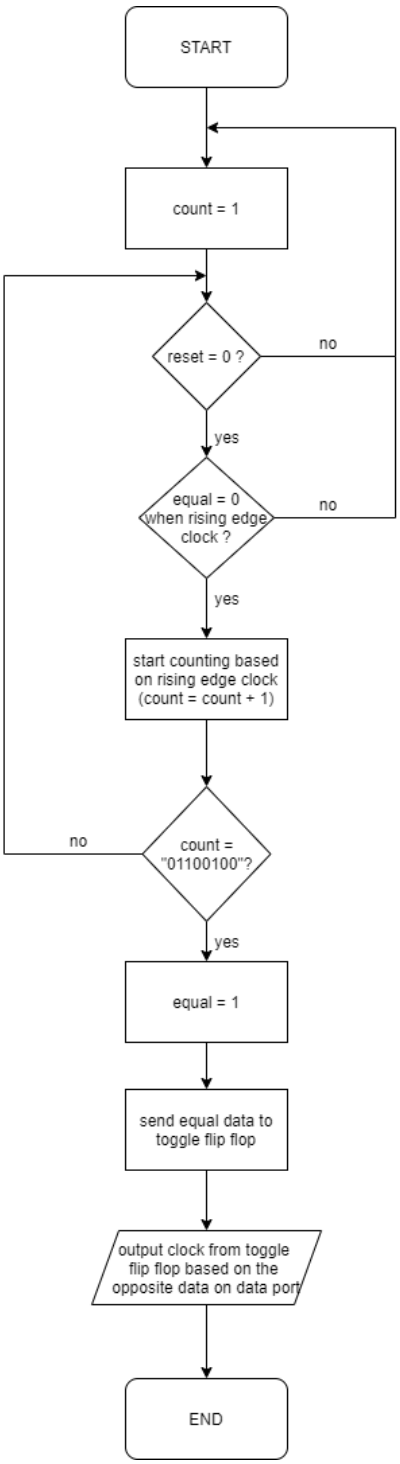
b) Percobaan Kedua

1. Penghubung *Pixel Capture* dengan VGA *Display*, dan *Thresholding Buffering* dan *Light Detection*



Gambar 2.7 Flowchart Penghubung *Pixel Capture* dengan VGA *Display*, dan *Thresholding Buffering* dan *Light Detection*

2. Pembuatan Clock untuk PWM pada Motor



Gambar 2.8 Flow Chart Pembuatan Clock untuk PWM pada Motor

BAB III
HASIL DAN ANALISA

3.1 ANALISA IMPLEMENTASI CODING

a) Hasil Percobaan Pertama

1. Clock Camera Control, VGA Display, Thresholding Buffering and Light Detection

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  -- camera control and vga and thresholding port
6  entity clockdivider is
7      Port (
8              camera_clk     : in  STD_LOGIC;
9              camcon_status  : in  STD_LOGIC;
10             clock_out      : out STD_LOGIC
11             );
12 end clockdivider;
13
14 architecture rtl_divider of clockdivider is
15    signal tmp        : std_logic := '0';
16
17
18 begin
19    process ( camera_clk, camcon_status) is
20    begin
21       -- Clock divider for vga thresholding from camera control
22       if (camcon_status = '1') then
23             if rising_edge(camera_clk) then
24                   if tmp = '0' then
25                      tmp <= '1';
26                      clock_out <= '1';
27                   elsif tmp = '1' then
28                      tmp <= '0';
29                      clock_out <= '0';
30                   end if;
31             end if;
32       elsif (camcon_status = '0') then
33             tmp <= '0';
34             clock_out <= '0';
35       end if;
36    end process;
37
38 end rtl_divider;
39
```

Gambar 3.1 Coding Camera Control, VGA Display, Thresholding Buffering and Light Detection

Pada gambar 3.1 hal pertama yang harus diketahui dalam pembuatan sesuatu sistem FPGA (VHDL) adalah menentukan *input* dan *outputnya*. Pada

gambar 3.1 karena dilakukan penghubungan antara *Pixel Capture* dengan *VGA Display*, dan *Thresholding Buffering dan Light Detection.* Cara kerja *VGA Display* dan *Thresholding Buffering dan Light Detection* tidak jauh berbeda, yaitu berhubungan dengan *Pixel* atau *Frame*. Maka, Frekuensi yang akan digunakan oleh *VGA Display* dan *Thresholding Buffering dan Light Detection* sebesar 25 MHz. Sedangkan frekuensi yang terdapat pada *Pixel Capture* sebesar 50 MHz, maka untuk memenuhi spesifikasi yang dibutuhkan *VGA Display* dan *Thresholding Buffering dan Light Detection,* frekuensi tersebut harus dibagi dua. Pada hal ini dapat diketahui *input* yang dimiliki adalah *clock* yang berasal dari *Pixel Capture* dan Camera Control Status dapat mengindikasikan bahwa camera OV6760 sudah siap untuk menangkap gambar, dan *output* yang dihasilkan adalah *clock* untuk *VGA Display* dan *Thresholding Buffering dan Light Detection.*

Oleh karena itu pada bagian *entity VHDL* terdapat camera_clk melambangkan *clock* yang didapatkan dari *pixel capture*, lalu camcon_status melambangkan *camera control status*, yang kedua *port* tersebut merupakan *input* dari *VGA Display* dan *Thresholding Buffering dan Light Detection*. Lalu terdapat clock_out melambangkan hasil clock divider, *port* ini merupakan *output* dari *VGA Display* dan *Thresholding Buffering dan Light Detection.*

Pada bahasa VHDL, *output port* tidak bisa langsung diperiksa, maka dibutuhkan sebuah sinyal internal yang melambangkan kondisi *output* pada saat itu. Sinyal internal yang digunakan adalah tmp.

Pada bagian *begin (line 18)*, terjadi proses perubahan berdasarkan *input* yang didapatkan dari camera_clk dan camcon_status. Pada bagian cara kerja *clock divider* (line 23 hingga 31) sinyal dari camera_clk diproses untuk menghasilkan suatu *output* yang baru yang dihasilkan dari *rising_edge* sinyal *camera clock* yaitu *clock out*.

Pada bagian elsif (line 32 hingga 35) merupakan *coding* untuk menghindari terjadinya *unknown* pada awal sistem ini berjalan. Pada sistem digital *unknown* tidak diperbolehkan terjadi, karena dapat menimbulkan *b*ug pada seluruh sistem.

2. Clock Penentuan PWM pada Motor

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  entity pwm_motor_clk is
6      Port (
7              -- motor clock
8              on_board_clk   : in std_logic;
9              reset          : in std_logic;
10             pwm_motor_clk  : out std_logic
11         );
12 end pwm_motor_clk;
13
14 architecture rtl_pwm_clk of pwm_motor_clk is
15    signal cnt        : integer := 1;
16    signal tmp        : std_logic := '0';
17
18 begin
19    process (on_board_clk, reset) is
20    begin
21
22       -- Clock divider for motor for pwm
23       if (reset = '1') then
24          cnt <= 1;
25          tmp <= '0';
26          pwm_motor_clk <= tmp;
27
28
29       elsif (reset = '0') then
30          if (rising_edge(on_board_clk)) then
```

Gambar 3.2 Coding Penentuan PWM pada Motor

Pada gambar 3.2 hal pertama yang harus diketahui dalam pembuatan sesuatu sistem FPGA (VHDL) adalah menentukan input dan outputnya. Pada gambar 3.2 karena dilakukan penentuan PWM pada motor. Cara kerja dari penentuan PWM pada motor tidak jauh berbeda seperti gambar 3.1, hal yang membedakan adalah *port* pada *input* dan output. *Port Input* pada penentuan PWM pada motor ini adalah on_board_clk yang melambangkan *clock* yang didapatkan dari *board* basys 2. Reset yang melambangkan proses mengulang paksa suatu progres dalam sistem menjadi kondisi awal. Lalu pada *Port Output* pada penetuan PWM pada motor ini adalah pwm_motor_clk yang

melambangkan hasil dari *output* PWM yang dibutuhkan oleh motor *driver*. Pada Penentuan PWM pada motor hasil *input* atau *output* berhubungan dengan *rising edge dan Counter*, *Counter* akan bertambah jika *rising edge* mendapatkan hasil dari *clock input*. Untuk menentukan frekuensi yang akan digunakan oleh PWM motor, dilihat pada *board* pada basys 3 diketahui frekeunsinya sebesar 100 MHz, sedangkan PWM motor membutuhkan *clock* dengan frekuensi 2 KHz. Oleh karena itu dibutuhkan *clock divider* yang dapat menyesuaikan frekuensi yang dibutuhkan pada sistem ini.

Pada hal ini dapat diketahui pembagian frekuensi *clock input* sebesar 50000 kali, maka dari itu dibutuhkan *counter* dan *comparator*. Pada bagian *architecture* pada bahasa VHDL, *output port* tidak bisa langsung diperiksa, maka dibutuhkan sebuah sinyal internal yang melambangkan kondisi *output* pada saat itu. Sinyal internal yang digunakan adalah tmp. Terdapat juga counter yang berfungsi untuk memastikan sinyal *clock* tidak selalu naik atau bernilai 1.

Pada bagian *begin (line 18)*, terjadi proses perubahan berdasarkan *input* yang didapatkan dari on_board_clk. Pada bagian cara kerja *clock divider* (line 23 hingga 26) sinyal dari *reset* diproses untuk mengembalikan *counter* kembali ke kondisi awal secara paksa, dan menghasilkan *clock* PWM *output* yang bernilai 0.

```
31              cnt <= cnt + 1;
32              if (cnt = 25000) then
33                 if tmp = '0' then
34                    tmp <= '1';
35                    pwm_motor_clk <= '1';
36                 else
37                    tmp <= '0';
38                    pwm_motor_clk <= '0';
39                 end if;
40                 cnt <= 1;
41              else
42                 pwm_motor_clk <= tmp;
43              end if;
44           else
45              pwm_motor_clk <= tmp;
46           end if;
47        end if;
48
49     end process;
50  end rtl pwm clk;
```

Gambar 3.3 Coding Penentuan PWM pada Motor

Pada gambar 3.3 merupakan sistem jika *input* reset tidak mengindikasikan *counter* untuk kembali ke kondisi awal, maka *counter* akan terus bertambah sampai ke kondisi akhir yang bernilai 25000 atau nilai maksimum. Nilai pada *clock* PWM *Output* akan berubah menjadi 1 atau sebaliknya, dan jika tmp, pwm_motor_clk bernilai 0 maka *counter* bernilai 1 yang menjelaskan bahwa sebelumnya *counter* berhasil mencapai nilai maksimum, lalu terjadi pengulangan secara otomatis.

b) Hasil Percobaan Kedua

Pada percobaan pertama, seluruh cara kerja yang digunakan bersifat high level karena tidak terdapat komponen elektronik digital yang digunakan. Oleh karena itu, seluruh cara kerja akan digunakan dalam komponen elektronik digital sehingga terdapat beberapa komponen yang akan digunakan, seperti *counter*, *comparator*, dan *toggle flip flop*. Pada bagian Clock Camera Control, VGA Display, Thresholding Buffering and Light Detection, akan digunakan Toggle Flip Flop yang akan digunakan untuk menyimpan 1 bit, sedangkan pada bagian Clock Penentuan PWM pada Motor, akan digunakan

counter, *comparator*, dan *toggle flip flop* untuk membagi suatu frekuensi input untuk jadi frekuensi yang lebih rendah.

1. **Clock Camera Control, VGA Display, Thresholding Buffering and Light Detection**

```
5   entity toggle_flip_flop is
6       Port ( data      : in  STD_LOGIC;
7              clk       : in  STD_LOGIC;
8              reset     : in  STD_LOGIC;
9              q         : out  STD_LOGIC;
10             qbar      : out  STD_LOGIC);
11  end toggle_flip_flop;
12
13  architecture rtl_toggle_flip_flop of toggle_flip_flop is
14
15  begin
16     process(clk, data) is
17     begin
18     -- setting back to default
19     if (reset = '1') then
20           q <= '1';
21           qbar <= '0';
22
23     elsif (rising_edge(clk)) then
24        -- prevent undefined from being error
25        if (data = 'U') then
26           q <= '0';
27           qbar <= '1';
28        elsif (data = '1') then
29           q <= '1';
30           qbar <= '0';
31        elsif (data = '0') then
32           q <= '0';
33           qbar <= '1';
34        end if;
```

Gambar 3.4 Coding Camera Control, VGA Display, Thresholding Buffering and Light Detection

Pada gambar 3.4 terdapat *coding camera control*, *VGA display*, *thresholding buffering* dan *light detection* pada percobaan kedua, yang membedakan dari hasil *coding* pada percobaan pertama dengan percobaan kedua adalah penambahan port-port. Pada hal ini digunakan sebuah *toggle flip flop* yang akan menyimpan 1 bit dan akan mengeluarkan 1 bit biner.

Pada hal ini terlihat apabila terdapat sinyal reset, maka *toggle flip flop* akan kembali menjadi kondisi semula. Apabila tidak terdapat sinyal reset, maka akan dilakukan pengecekan pada equal ketika terjadi rising_edge dari clock input. Output yang akan diberikan sebagai input untuk PWM motor clock adalah kebalikan dari data dan data port tersebut akan mendapatkan input berupa feedback dari qbar.

2. Clock Penentuan PWM pada Motor

a. Counter

```
1  library IEEE;
2  use IEEE.STD_LOGIC_ARITH.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity counter_clk is
7      Port ( on_board_clk       : in     STD_LOGIC;
8             reset              : in     STD_LOGIC;
9             equal              : in     STD_LOGIC;
10            cnt_out            : out    STD_LOGIC_VECTOR(7 downto 0)
11            );
12 end counter_clk;
13
14 architecture rtl_counter of counter_clk is
15    signal cnt : std_logic_vector(7 downto 0) := "00000001";
16
17 begin
18    process(on_board_clk, reset) is
19    begin
20          if (reset = '1') then
21                cnt <= "00000001";
22          elsif (rising_edge(on_board_clk)) then
23             if (equal = '1') then
24                cnt <= "00000001";
25             else
26                cnt <= cnt + "1";
27             end if;
28          end if;
29    end process;
30    cnt_out <= cnt;
31 end rtl_counter;
32
33
```

Gambar 3.5 Coding Counter untuk PWM Motor

Pada gambar 3.5 merupakan *coding counter* untuk PWM *Motor*, Pada bagian *counter* ini cara kerja yang perlu diketahui adalah ketika *rising edge*, akan dilakukan pengecekan apakah *equal* atau tidak, jika terjadi *equal* maka reset akan terjadi, jika *equal* tidak terjadi maka reset juga tidak akan terjadi dan *counter* akan mengeluarkan *output* berupa nilai *output* sebelumnya ditambah 1.

b. Comparator

```
    Port ( clk                  : in     STD_LOGIC;
           reset                : in     STD_LOGIC;
           equal                : in     STD_LOGIC;
           cnt_out              : out    STD_LOGIC_VECTOR(7 downto 0)
           );
end counter_clk;

architecture rtl_counter of counter_clk is
   signal cnt : std_logic_vector(7 downto 0) := "00000001";

begin
   process(clk, reset, equal) is
   begin
         -- reset counter to default if reset = 1
         if (reset = '1') then
               cnt <= "00000001";

         elsif (rising_edge(clk)) then
            -- reset counter to default if equal = 1
            if (equal = '1') then
               cnt <= "00000001";

            -- start counting
            else
               cnt <= cnt + "1";
            end if;
         end if;

   end process;
   cnt_out <= cnt;
```

Gambar 3.6 Coding Comparator untuk PWM Motor

Pada gambar 3.6 merupakan *coding comparator* untuk PWM *Motor*, Pada bagian *comparator* ini cara kerjanya adalah pembandingan bit per bit pada *input* A dan *input* B. Jika *input* A dan *input B* adalah sama, maka *output* E bernilai 1, dan juga sebaliknya bernilai 0 jika tidak sama.

c. Toggle flip flop

```
5   entity toggle_flip_flop is
6       Port ( data      : in  STD_LOGIC;
7              clk       : in  STD_LOGIC;
8              reset     : in  STD_LOGIC;
9              q         : out  STD_LOGIC;
10             qbar      : out  STD_LOGIC);
11  end toggle_flip_flop;
12
13  architecture rtl_toggle_flip_flop of toggle_flip_flop is
14
15  begin
16     process(clk, data) is
17     begin
18     -- setting back to default
19     if (reset = '1') then
20           q <= '1';
21           qbar <= '0';
22
23     elsif (rising_edge(clk)) then
24        -- prevent undefined from being error
25        if (data = 'U') then
26           q <= '0';
27           qbar <= '1';
28        elsif (data = '1') then
29           q <= '1';
30           qbar <= '0';
31        elsif (data = '0') then
32           q <= '0';
33           qbar <= '1';
34        end if;
```

Gambar 3.7 Coding Toggle flip flop untuk PWM Motor

Pada gambar 3.7 merupakan *coding toggle flip flop* untuk PWM *Motor*, Pada bagian *toggle flip flop* ini cara kerjanya adalah jika *reset* terjadi atau bernilai 1 maka q bernilai 1 dan qbar bernilai 0 (dikembalikan pada kondisi semula). Jika *rising edge*, maka nilai q akan mencerminkan nilai data port sedangkan nilai qbar akan selalu kebalikan dari nilai q. Jika data bernilai 0, maka nilai q akan bernilai 0 dan qbar bernilai 1 atau selalu kebalikan dari nilai q.

3.2 ANALISA IMPLEMENTASI CODING

a) Hasil Percobaan Pertama

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.numeric_std.ALL;

ENTITY clockdiv_tb IS
END clockdiv_tb;

ARCHITECTURE behavior OF clockdiv_tb IS

   -- camera control and vga and thresholding clock
    COMPONENT clockdivider
    PORT(
            camera_clk     : in  STD_LOGIC;
            camcon_status  : in  STD_LOGIC;
            clock_out      : out STD_LOGIC
            );

   END COMPONENT;

   -- motor clock
    COMPONENT pwm_motor_clk_divider
    PORT(

            on_board_clk   : in std_logic;
            reset          : in std_logic;
            pwm_motor_clk  : out std_logic

    );
    END COMPONENT;


   -- INPUT for thresholding and vga from camera control
   signal camera_clk       : std_logic := '1';
   signal camcon_status    : std_logic := '0';
```

Gambar 3.8 Coding Testbench Part 1

Pada gambar 3.8 dikarenakan terdapat 2 entity atau file maka perlu digabungkan, maka perlu dideklarasikan 2 entity tersebut menjadi 2 komponen untuk melakukan testbench. Berdasarkan *Port Input* terdapat *camera_clk, camcon_status, on_board_clk,* dan *reset*. Sedangkan pada *Port Output* tedapat *clock_out,* dan *pwm_motor_clk*. Lalu pada sinyal *input* camera_clk bernilai 1 agar tidak terjadi *unknown* pada sistem, dikarenakan sinyal *input* membutuhkan inisialisasi data, sedangkan sinyal *output* tidak perlu inisialisasi data.

```
   -- OUTPUT for thresholding and vga from camera control
   signal clock_out        : std_logic;


   -- INPUT clock divider from on board basys 3
   signal on_board_clk     : std_logic := '1';
   signal reset            : std_logic := '0';


   -- OUTPUT clock divider from on board basys 3 to motor
   signal pwm_motor_clk    : std_logic;

   -- Clock Description
   constant camcon_clk_frequency       : integer   := 50e6;
   constant camcon_clk_period          : time      := 1000 ms / camcon_clk_frequency;

   constant basys3_clk_frequency       : integer   := 100e6;
   constant basys3_clk_period          : time      := 1000 ms / basys3_clk_frequency;

BEGIN
   -- Assigning all testbench variable to all port in implementation port
   clock_div : entity work.clockdivider(rtl_divider)
   port map(
            camera_clk     => camera_clk,
            camcon_status  => camcon_status,
            clock_out      => clock_out
            );


   pwm_motor : entity work.pwm_motor_clk(rtl_pwm_clk)
   port map(
            on_board_clk   => on_board_clk,
            reset          => reset,
```

Gambar 3.9 Coding Testbench Part 2

Pada gambar 3.9 merupakan deskripsi gambar mengenai beberapa data yang dibutuhkan untuk memulai simulasi. Lalu pada *begin*, terdapat *port map* yang berfungsi untuk menghubungkan semua data yang berada di *testbench* dengan *implementation*, baik data *input* maupun *output*.

```
            pwm_motor_clk  => pwm_motor_clk
            );

   -- Simulation input clock
   camera_clk     <= not camera_clk          after camcon_clk_period / 2;
   on_board_clk   <= not on_board_clk        after basys3_clk_period / 2;


   process is
   begin
      wait for 100ns;

      camcon_status <= '1';
      reset <= '1';

      wait for 100ns;
      reset <= '0';

      wait for 400ns;
      camcon_status <= '0';

      wait;
   end process;

END;
```

Gambar 3.10 Coding Testbench Part 3

Pada gambar 3.10 karena duty cycle bernilai 50% maka periode dibagi dua, dapat diketahui berdasarkan sistem diatas, bahwa setiap setengah periode berubah camera_clk berubah menjadi not camera_clk yang berarti pada kondisi awal nilai dari camera_clk adalah 1, lalu setelah setengah periode maka camera_clk menjadi not camera_clk yang bernilai kebalikan dari 1 yaitu 0, dan siklus cara kerja ini akan berjalan secara terus menerus sampai sistem mati. Siklus cara kerja tersebut juga berlaku pada cara kerja on_board_clk, namun periode pada on_board_clk berbeda dengan periode yang digunakan pada camcon_clk period.

Pada bagian *begin* dilakukan simulasi ketika mencapai 100 ns maka camcon_status dan reset bernilai 1 yang sebelum 100 ns bernilai 0, hal ini menunjukkan bahwa sistem *clock divider VGA Display* dan *Thresholding Buffering dan Light Detection* berjalan dengan benar. Lalu setelah 100 ns reset berubah kembali menjadi 0, sehingga membuktikan bahwa sistem *clock divider* dari PWM motor *driver* mulai berjalan dengan benar. Lalu setelah 400 ns berikutnya camcon_status berubah menjadi 0, yang menandakan bahwa sistem *clock divider VGA Display* dan *Thresholding Buffering dan Light*

Detection berhenti total, tetapi sistem *clock divider* dari PWM motor *driver* masih tetap berjalan karena reset tetap atau tidak ikut berubah yang bernilai 0.

b) Hasil Percobaan Kedua

1. Clock Camera Control, VGA Display, Thresholding Buffering and Light Detection

```
5  ENTITY vga_thresh_clock_tb IS
6  END vga_thresh_clock_tb;
7
8  ARCHITECTURE behavior OF vga_thresh_clock_tb IS
9     -- camera control and vga and thresholding clock
10    COMPONENT toggle_flip_flop_clock
11    PORT(
12          -- input port for toggle flip flop
13          -- data                    : feedback from clock25Mhz to store data
14          -- camera_clk              : to start giving output
15          -- camcon_status           : acts as reset for flip flop (0 means camera already done with configuration and 1 means not done)
16          data                 : in  STD_LOGIC;
17          camera_clk           : in  STD_LOGIC;
18          camcon_status        : in  STD_LOGIC;
19
20          -- output port for toggle flip flop
21          -- q                       : reflects data from data port
22          -- clock25Mhz              : output that will be created for vga and thresholding (25MHz)
23          q                    : out  STD_LOGIC;
24          clock25Mhz           : out  STD_LOGIC
25          );
26
27    END COMPONENT;
28
29    -- INPUT
30    signal camera_clk          : std_logic := '0';
31    signal camcon_status       : std_logic := '0';
32
33    -- OUTPUT
34    signal q                   : std_logic;
35    signal clock25Mhz          : std_logic;
```

Gambar 3.11 Coding Testbench untuk VGA Display Percobaan Kedua Part 1

```
37    -- Clock Description
38    constant camcon_clk_frequency       : integer   := 50e6;
39    constant camcon_clk_period          : time      := 1000 ms / camcon_clk_frequency;
40
41  BEGIN
42    -- Assigning all testbench variable to all port in implementation port
43    toggle_flop_flop_clock  : entity work.toggle_flip_flop (rtl_toggle_flip_flop)
44    port map(
45             data           => qbar,
46             clk            => E,
47             reset          => reset,
48             q              => q,
49             qbar           => qbar
50
51    );
52    -- Simulation input clock
53    camera_clk     <= not camera_clk          after camcon_clk_period / 2;
54
55
56    process is
57    begin
58       camcon_status <= '1';
59       wait for 100ns;
60
61       camcon_status <= '0';
62       wait;
63    end process;
64
65  END;
```

Gambar 3.12 Coding Testbench untuk VGA Display Percobaan Kedua Part 2

Pada gambar 3.11 dan gambar 3.12 merupakan *coding testbench* untuk VGA *Display* percobaan kedua. Perbedaan dari percobaan pertama dengan percobaan kedua adalah penambahan beberapa port seperti data, q dan qbar. Lalu dilakukan penggabungan dengan komponen-komponen yang terdapat pada percobaan pertama, sehingga menghasilkan rangkaian *low level*. Berdasarkan hasil blok diagram pada Gambar 2.3, port *data* akan mendapatkan suatu feedback dari qbar, *clk* akan mendapatkan camera_clk, port reset akan mendapatkan camcon_status, port q akan mencerminkan nilai data atau q, dan port qbar akan berkebalikan dengan dengan q dan pada hal ini qbar akan dihubungkan dengan port clock25Mhz.

2. Clock Penentuan PWM pada Motor

```
5  ENTITY pwm_motor_tb IS
6  END pwm_motor_tb;
7
8  ARCHITECTURE behavior OF pwm_motor_tb IS
9     -- counter
10    COMPONENT counter
11    PORT (
12         -- input counter
13         -- on_board_clk for clock  : to start counting
14         -- reset                  : to force the counter back to 0 (external signal)
15         -- equal                  : to reset counter back to 0 if already pass the maximum in the comparator (internal signal from comparator)
16         on_board_clk        : in     STD_LOGIC;
17         reset               : in     STD_LOGIC;
18         equal               : in     STD_LOGIC;
19
20         -- output counter
21         -- cnt_out          : result of counting that will be passed to comparator
22         cnt_out             : out    STD_LOGIC_VECTOR(7 downto 0)
23    );
24    END COMPONENT;
25
26    -- comparator
27    COMPONENT comparator
28    PORT (
29         -- input comparator
30         -- cnt_out          : value from counter that will be compared
31         -- B                : comparison number
32         cnt_out             : in     STD_LOGIC_VECTOR (7 downto 0);
33         B                   : in     STD_LOGIC_VECTOR (7 downto 0);
34
35         -- output comparator
36         -- E                : 1 is equal, 0 is not equal
37         E                   : out    STD_LOGIC
38    );
```

Gambar 3.13 Coding Testbench pada PWM Motor Percobaan Kedua Part 1

```
39     END COMPONENT;
40
41     -- toggle flip flop
42     COMPONENT toggle_flip_flop_pwm
43     PORT (
44          -- input register
45          -- data               : feedback from qbar to store data
46          -- clk                : to start giving output
47          -- reset              : to force register back to default value
48          data                  : in  STD_LOGIC;
49          clk                   : in  STD_LOGIC;
50          reset                 : in  STD_LOGIC;
51
52          -- output register
53          -- q                  : reflects data from data port
54          -- qbar               : output that will be created for motor (500 kHz)
55          q                     : out  STD_LOGIC;
56          qbar                  : out  STD_LOGIC
57
58     );
59     END COMPONENT;
60
61    -- INPUT
62    signal on_board_clk     : std_logic := '1';
63    signal reset            : std_logic := '0';
64    signal cnt_out          : std_logic_vector (7 downto 0);
65    signal B                : std_logic_vector (7 downto 0) := "01100100";
66
67    -- OUTPUT
68    signal E                : std_logic;
69    signal q                : std_logic;
70    signal qbar             : std_logic;
```

Gambar 3.14 Coding Testbench pada PWM Motor Percobaan Kedua Part 2

```
72    -- Clock Description
73    constant basys3_clk_frequency       : integer   := 100e6;
74    constant basys3_clk_period          : time      := 1000 ms / basys3_clk_frequency;
75
76  BEGIN
77    -- Assigning all testbench variable to all port in implementation port
78    counter_clk    : entity work.counter_clk (rtl_counter)
79    port map(
80             clk            => on_board_clk,
81             reset          => reset,
82             equal          => E,
83             cnt_out        => cnt_out
84    );
85
86    comparator_clk : entity work.comparator (rtl_comparator)
87    port map(
88             A              => cnt_out,
89             B              => B,
90             E              => E
91    );
92
93    toggle_flop_flop_pwm : entity work.toggle_flip_flop (rtl_toggle_flip_flop)
94    port map(
95             data           => qbar,
96             clk            => E,
97             reset          => reset,
98             q              => q,
99             qbar           => qbar
100
101   );
102
103   -- Simulation input clock
104   on_board_clk   <= not on_board_clk        after basys3_clk_period / 2;
```

```
106     process is
107     begin
108         reset <= '1';
109         wait for 100ns;
110
111         reset <= '0';
112         wait;
113     end process;
114 END;
```

Gambar 3.15 Coding Testbench pada PWM Motor Percobaan Kedua Part 3

Pada gambar 3.13, gambar 3.14 dan gambar 3.15 merupakan *coding testbench* pada PWM *Motor* pada percobaan kedua. Perbedaan pada percobaan pertama dengan percobaan kedua adalah perubahan pada sebuah komponen dan penambahan 2 buah komponen. Perubahan pada sebuah komponen tersebut terjadi pada *counter*, terdapat 3 *input* pada komponen *counter* yaitu on_board_clk, reset, dan equal. Sedangkan *output* tetap sama yaitu cnt_out. Lalu penambahan 2 buah komponen tersebut adalah komponen *comparator* dan *toggle flip flop*. Cara kerja komponen *comparator* adalah membandingkan bit per bit pada *input* A dan *input* B. Jika *input* A dan *input B* adalah

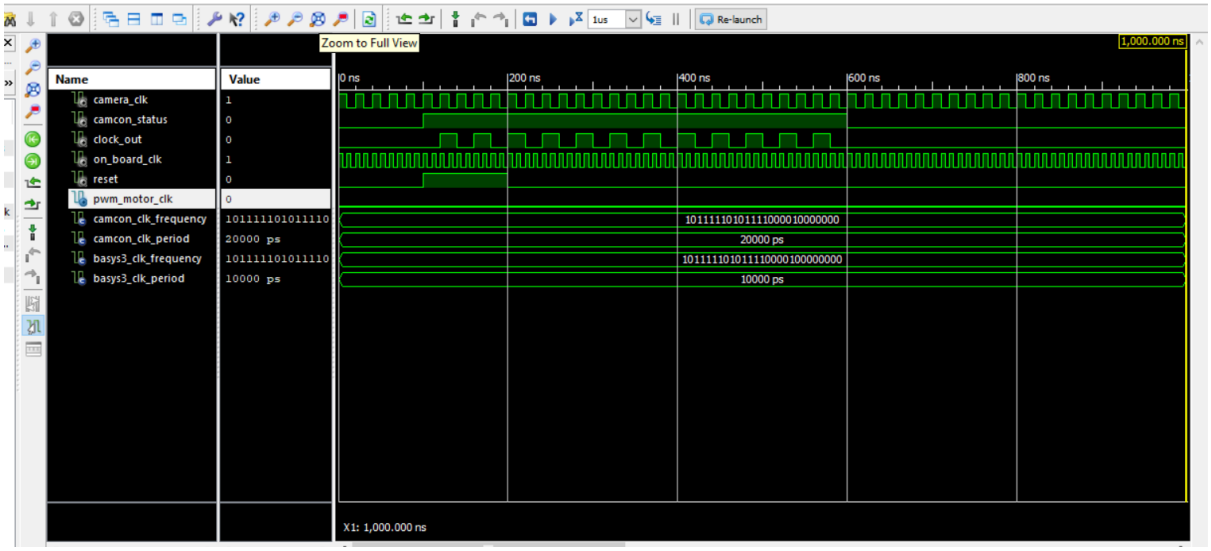
sama, maka *output* E bernilai 1, dan juga sebaliknya bernilai 0 jika tidak sama. Cara kerja komponen *toggle flip flop* adalah jika *reset* terjadi atau bernilai 1 maka q bernilai 1 dan qbar bernilai 0 (dikembalikan pada kondisi semula), dan juga berlaku kebalikannya. Jika *rising edge* maka data akan bernilai 1, maka nilai q akan bernilai 1 dan qbar bernilai 0 atau selalu kebalikan dari nilai q. Jika data bernilai 0, maka nilai q akan bernilai 0 dan qbar bernilai 1 atau selalu kebalikan dari nilai q. Pada *coding* tersebut dapat dihubungkan cara kerja tersebut dengan blok diagram pada gambar 2.3. Terdapat beberapa port yang memiliki *input* dan *output* yang berhubungan atau menghasilkan satu sama lain sesuai dengan tabel dibawah ini sebagai berikut:

Port	**Input**	**Output**
clk (Counter)	On_board_clk (100 MHz)	Cnt_out
Reset (Counter)	reset	Cnt_out
Equal (Counter)	E (comparator)	Cnt_out
Cnt_out (Counter)	Clk, reset, equal	A (comparator)
A (Comparator)	Cnt_out (counter)	E (comparator)
B (Comparator)	01100100 (100)	E (comparator)
E (Comparator)	A, B (comparator)	Clk (Toggle Flip Flop)
Data (Toggle Flip Flop)	qbar	q, qbar
Clk (Toggle Flip Flop)	E (comparator)	q, qbar
Reset (Toggle Flip Flop)	reset	q, qbar
q (Toggle Flip Flop)	Data, clk, reset	ground
qbar (Toggle Flip Flop)	Data, clk, reset	Pwm_clock (500 kHz)

Tabel 3.1 Port, input dan output terhadap PWM Motor Clock

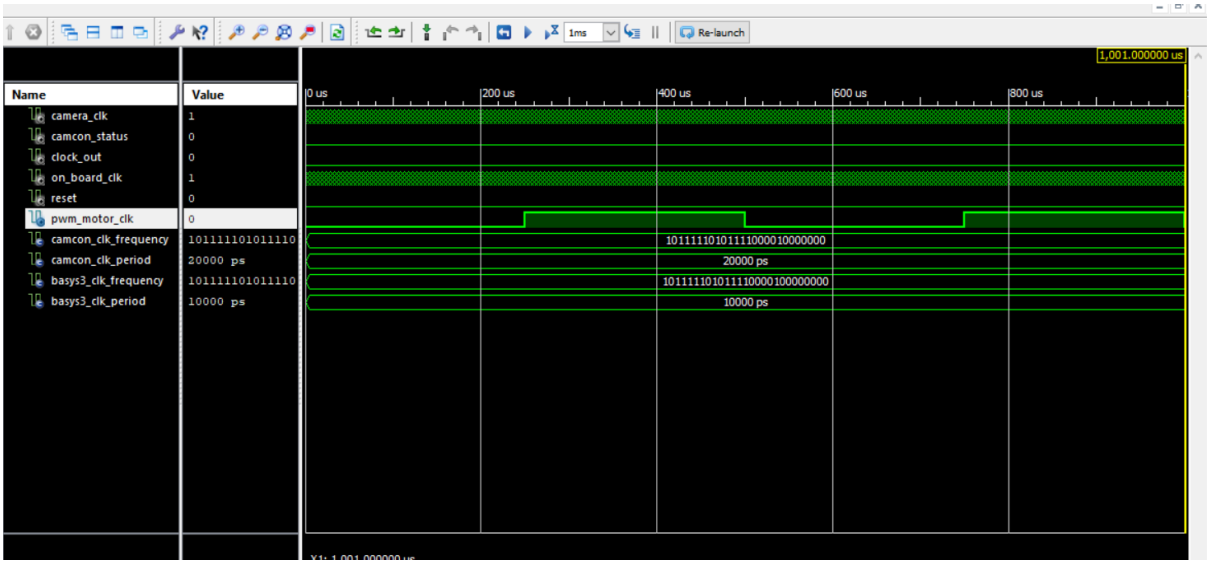
3.2. ANALISA DAN HASIL PERCOBAAN SIMULASI

a) Percobaan Pertama



Gambar 3.16 Penghubungan antara *Pixel Capture* dengan *VGA Display*, dan *Thresholding Buffering dan Light Detection.*

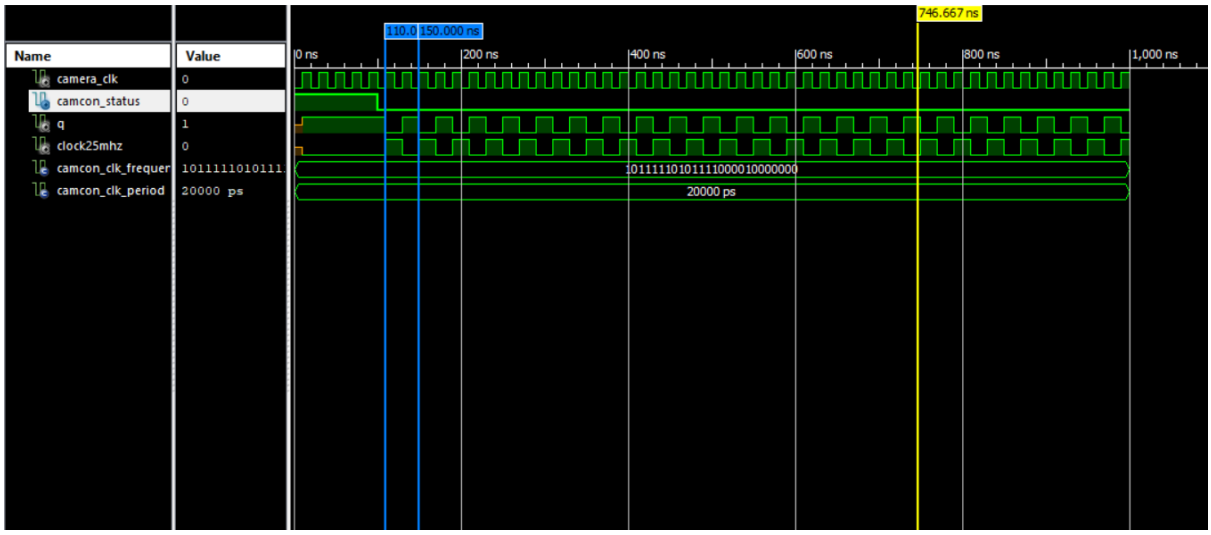
Pada gambar 3.16 diatas merupakan hasil dari penghubungan antara *Pixel Capture* dengan *VGA Display*, dan *Thresholding Buffering dan Light Detection.* Berdasarkan gambar 3.16 dapat diperhatikan bahwa clock_out akan memiliki nilai jika camera_clk dan camcon_status bernilai 1. Pada gambar 3.16 dapat dilihat bahwa setiap 200 ns terdapat banyak 10 gelombang pada camera_clk, maka dapat diketahui bahwa setiap 1 gelombang berperiode 20 ns, sehingga diketahui bahwa frekuensi pada camera_clk adalah 50 MHz. Sedangkan pada clock_out terdapat 5 gelombang saja. Berdasarkan banyak gelombang terhadap camera_clk dengan clock_out dapat dibuktikan bahwa setiap 200 ns pada clock_out menghasilkan 5 gelombang, maka dapat dibuktikan bahwa setiap 1 gelombang berperiode 40 ns, maka dapat diketahui bahwa frekuensi pada clock_out adalah 25 MHz.



Gambar 3.17 Penentuan PWM Motor Driver

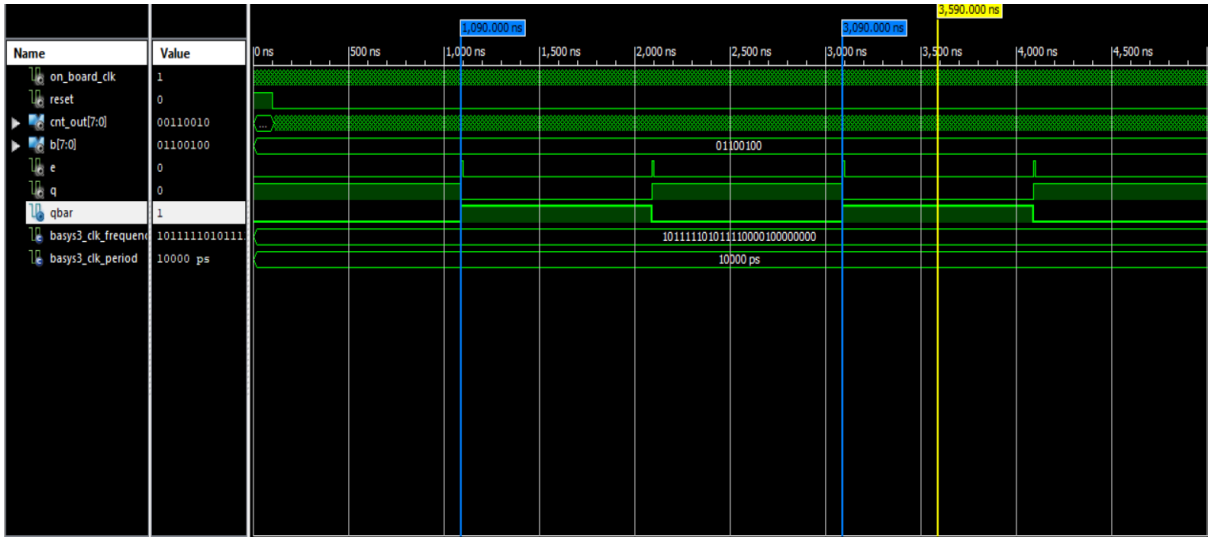
Pada gambar 3.17 diatas merupakan hasil dari penentuan PWM pada *Motor Driver*. Berdasarkan gambar 3.17 dapat diperhatikan bahwa pwm_motor_clk akan bernilai jika *reset* bernilai 0, dan *counter* sudah mencapai 25000 *rising edge* pada on_board_clk. Berdasarkan gambar 3.17 dapat diketahui bahwa on_board_clk dengan pwm_motor_clk memiliki jarak frekuensi yang sangat jauh berbeda satu sama lain. Pada on_board_clk memiliki frekuensi bernilai 100 MHz, sedangkan pwm_motor_clk hanya memiliki frekuensi bernilai 2 KHz, maka dari itu grafik data pada gambar 3.17 diatas terlihat seperti gelombang yang menyatu.

b) Percobaan Kedua



Gambar 3.18 Penghubungan antara *Pixel Capture* dengan *VGA Display*, dan *Thresholding Buffering dan Light Detection.*

Pada gambar 3.18 diatas merupakan hasil dari penghubungan antara *Pixel Capture* dengan *VGA Display*, dan *Thresholding Buffering dan Light Detection* pada percobaan kedua. Berdasarkan gambar 3.18 dapat diperhatikan bila q bernilai 1 dan qbar (clock 25MHz) bernilai 0, maka dapat dinyatakan bahwa setiap 1 gelombang berperiode 40 ns. Maka dapat diketahui bahwa frekuensi pada output_port adalah 25 MHz.



Gambar 3.19 Penentuan PWM Motor Driver

Pada gambar 3.19 diatas merupakan hasil dari penentuan PWM pada *Motor Driver*. Berdasarkan gambar 3.19 dapat diperhatikan bahwa pada saat terjadi rising edge pada on_board_clk dan nilai counter akan bertambah 1 dan menghasilkan cnt_out bernilai 100 dalam decimal, equal akan menghasilkan sinyal bila *reset* bernilai 0 karena apabila *reset* bernilai 1 maka cnt_out akan bernilai 1 dan ketika dibandingkan dengan menggunakan *comparator*, hal ini tidak akan menghasilkan suatu output. Nilai equal akan bernilai 1 selama 1 periode karena cnt_out akan kembali ke kondisi semula setelah terjadi nilai equal tersebut.

Berdasarkan gambar 3.19 dapat diketahui pula bahwa q dan qbar berubah berdasarkan sinyal dari equal. Dapat diperhatikan bila q bernilai 0 dan qbar bernilai 1, maka dapat dinyatakan bahwa setiap 1 gelombang berperiode 2ms. Maka dapat diketahui bahwa frekuensi pada output_port adalah 500 kHz. Sinyal q dan qbar akan berubah setiap rising edge yang terjadi pada equal. Hal ini dapat terjadi karena pada

bagian coding testbench kita telah menghubungkan sinyal clock yang akan digunakan dalam Toggle Flip Flop dengan sinyal equal.

References

Design, Electronic, 2016. *Clock Generators Bring Stability to Timing-Circuit Designs.* [Online]
Available at: https://www.electronicdesign.com/technologies/analog/article/21802214/clock-generators-bring-stability-to-timingcircuit-designs
[Accessed 01 June 2021].

Hub, E., 2019. *Sequential Circuits Basics.* [Online]
Available at: https://www.electronicshub.org/sequential-circuits-basics/
[Accessed 01 June 2021].

Hutahaean, A., n.d. *Digital komparator.* [Online]
Available at: https://www.academia.edu/21516089/Digital_komparator
[Accessed 1 June 2021].

t., n.d. *Digital Counters.* [Online]
Available at: https://www.tutorialspoint.com/computer_logical_organization/digital_counters.htm
[Accessed 1 June 2021].

University, S., n.d. *Rangkaian komparator adalah satu jenis penerapan.* [Online]
Available at: https://www.coursehero.com/file/p3knb2s/Rangkaian-komparator-adalah-satu-jenis-penerapan-rangkaian-kombinasional-yang/
[Accessed 1 June 2021].

Widianto, M. H., 2019. *Flip-Flop dan Jenis-jenisnya.* [Online]
Available at: https://binus.ac.id/bandung/2019/12/flip-flop-dan-jenis-jenisnya/
[Accessed 12 June 2021].

Wojciechowicz, T., 2018. *What Are Clock Signals in Digital Circuits, and How Are They Produced?.* [Online]
Available at: https://www.semiconductorstore.com/blog/2018/What-Are-Clock-Signals-in-Digital-Circuits-and-How-Are-They-Produced-Symmetry-Blog/3322/
[Accessed 01 June 2021].

Hyperphysics.phy-astr.gsu.edu. 2021. *D Flip-Flops*. [online] Available at: <http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/Dflipflop.html> [Accessed 12 June 2021].