PERANCANGAN RANGKAIAN DIGITAL MAZE PUZZLE

Ivan Giovanni (13218006)
Asisten: Mikael Wahyu Diantama (13216014)
Tanggal Percobaan: 25/11/2019
EL2102-Sistem Digital
Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

Pada proyek ini, dibuat sebuah permainan labirin digital bernama "Maze Puzzle". Permainan ini dibuat menggunakan VHDL dan diimplementasikan pada FPGA dengan input berupa switch, serta output berupa layar monitor. Program ini dirancang sedemikian rupa agar menampilkan antarmuka yang menarik bagi pemain. Untuk menambah kompleksitas permainan, pemain harus menghindari rintangan yang ditemui sepanjang permainan dengan berbekal 3 nyawa untuk sampai ke garis finish.

Kata kunci: Maze Puzzle, VHDL, FPGA

1. PENDAHULUAN

FPGA tidak hanya dimanfaatkan sebagai sarana percobaan, namun dapat digunakan sebagai alat pemrosesan dengan fungsi tertentu, salah satunya adalah pemrosesan permainan. Dengan memanfaatkan konsep rangkaian sekuensial menggunakan VHDL, dapat dibuat suatu permainan digital yang menarik pada FPGA.

Pada percobaan ini, dibuat suatu permainan labirin dengan menggunakan VHDL. Program yang dibuat akan diimplementasikan pada FPGA dan dihubungkan dengan *output* layar monitor. Pada proyek ini, dibuat pula tampilan antarmuka pengguna yang menarik agar pengguna dapat menikmati permainan dengan baik.

Secara umum, proyek ini bertujuan untuk membuat permainan labirin yang dapat menghibur pengguna. Permainan ini diharapkan dapat dimainkan oleh semua lapisan usia dan bermanfaat dalam mengembangkan strategi serta alur berpikir manusia.

Secara khusus, pembuatan program permainan bertujuan sebagai sarana untuk mengenal dan mempelajari perancangan sistem digital, khususnya melalui pemrograman VHDL, VGA interface, dan implementasi pada FPGA.

2. STUDI PUSTAKA

2.1 *FIELD-PROGRAMMABLE GATE ARRAY (FPGA)*

Field-Programmable Gate Array (FPGA) merupakan suatu perangkat logika yang dapat diprogram dan mendukung implementasi sirkuit logika dalam skala besar, yakni hingga lebih dari 1 juta gerbang logika. [1]

2.2 *FINITE STATE MACHINE (FSM)*

Finite State Machine (FSM) merupakan suatu model yang mensimulasikan logika sekuensial dengan sejumlah kondisi (*state*), kejadian (*input*) dan aksi (*ouput*) yang saling berkaitan. Keluaran rangkaian dapat bergantung pada *state* dan *input* (Mealy) ataupun hanya bergantung pada *state*-nya saja (Moore). [2]

2.3 *VIDEO GRAPHICS ARRAY (VGA)*

Video Graphics Array (VGA) merupakan sebuah standar tampilan yang diperkenalkan oleh IBM pada 1987 [3]. Hingga saat ini, VGA masih cukup populer dan dapat dijumpai pada banyak perangkat, salah satunya terdapat pada layar monitor LCD.

Interface VGA menggunakan 2 jenis sinyal utama, yakni sinyal warna (merah, hijau, dan biru) dan sinyal sinkron (horizontal dan vertikal). Berikut adalah penjelasan beberapa sinyal yang digunakan:

a. *Horizontal Sync (TTL level)*
   Sinyal ini aktif pada *range* piksel kolom 0 sampai dengan 639. Ketika sinyal tidak aktif, terjadi pergantian baris.

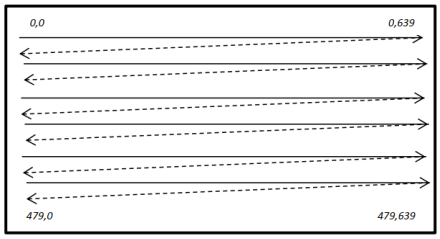
b. *Vertical Sync (TTL level)*
   Sinyal ini aktif pada *range* piksel baris 0 sampai dengan 479. Ketika sinyal tidak aktif, maka terjadi pergantian layar atau kembali ke baris pertama.

c. Sinyal warna RGB (Analog 3 pin: 0,7 – 1 V)

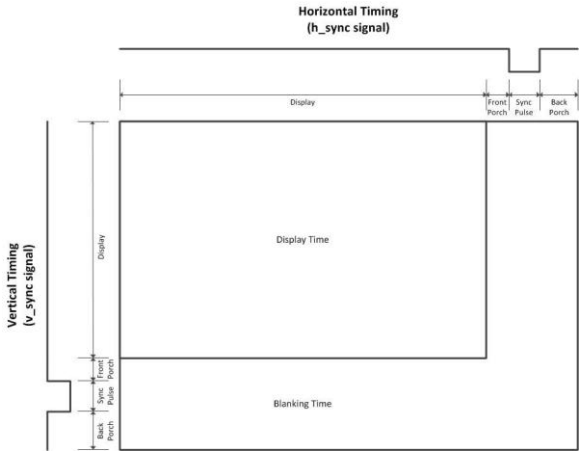
   Sinyal ini merepresentasikan intensitas untuk masing-masing komponen warna (merah, hijau, dan biru) untuk setiap piksel yang saat itu aktif.

Sehingga ketiga sinyal ini berubah-ubah sesuai piksel yang sedang aktif dalam proses *scanning* (dari kiri ke kanan untuk setiap baris, selanjutnya dari baris paling atas sampai baris paling bawah).[4]

Pada percobaan kali ini kita menggunakan resolusi 640x480 px dan menggunakan *refresh rate* lebih dari 60 Hz. *Refresh rate* ini digunakan karena pada rentang kurang dari 30-60 Hz manusia dapat melihat adanya *flicker*. Selain itu *refresh rate* ini juga umum digunakan pada monitor LCD. LCD modern memiliki fitur *multirate*, sehingga kita tidak harus tepat membuat *refresh rate*-nya 60 Hz. Proses *scanning* berawal dari kiri atas ke kanan lalu ke kiri bawah dan kembali ke kiri atas ketika sudah mencapai piksel terakhir.[5]



Gambar 2-1-1 Razor Scan pada Layar LCD [6]

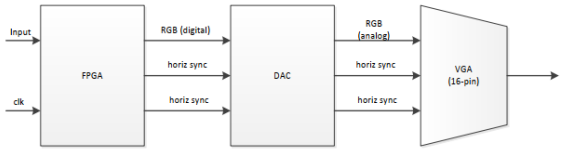


Gambar 2-1-2 *Timing Signal* untuk VGA 640x480 px [7]

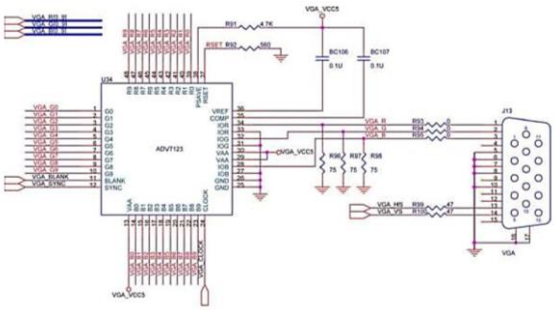
Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

Tabel 2-1-1 Nilai Parameter VGA *Signal Timing* [8]

Gambar 3 menunjukkan blok diagram dari FPGA hingga ke LCD monitor. Chip DAC mengubah sinyal digital ke analog. Dalam kasus ini, data RGB digital diubah ke analog, begitu pula untuk sinyal sinkronisasinya. Pada gambar 4 ditampilkan skematik dari *display* VGA pada board DE1. Board DE1 menyediakan 16-pin konektor untuk output VGA dan Analog Devices ADV7123 10-bit high speed video DAC. DAC ini mendapatkan sinyal sinkronisasi dari FPGA.



Gambar 2-1-3 Diagram Blok VGA Display [9]



Gambar 2-1-4 Skematik VGA Display [10]

3. METODOLOGI

3.1 KOMPONEN DAN ALAT

- PC yang telah ter-*install* Quartus II dan GIMP
- Board FPGA tipe Altera DE1
- Monitor LCD
- Catu daya, kabel, dan konektor tambahan
- Kabel *downloader* USB-Blaster

3.2 SPESIFIKASI PERMAINAN

- Menggunakan *input* 4 buah *switch* untuk menggerakkan objek (SW6 – SW9)
- Menggunakan *input* 1 *switch* untuk mengatur kecepatan gerak objek (SW0)
- Menggunakan *input* 1 *switch* untuk *reset* permainan
- Menggunakan *output* layar LCD

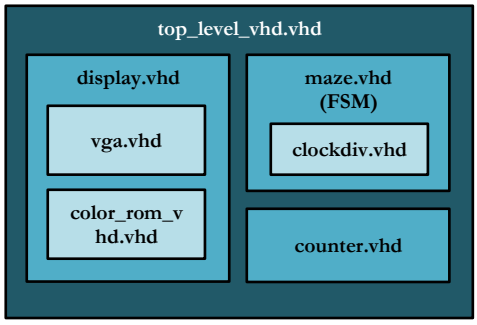
3.3 FITUR PERMAINAN

- Pergerakan objek permainan dapat diatur ke 4 arah
- Kecepatan objek permainan yang dapat diubah
- Terdapat 3 nyawa
- Terdapat *obstacle* diam dan *obstacle* bergerak yang harus dihindari
- Terdapat tampilan khusus ketika akan memulai permainan, finish, ataupun ketika gagal (kalah)

3.4 LANGKAH PERCOBAAN

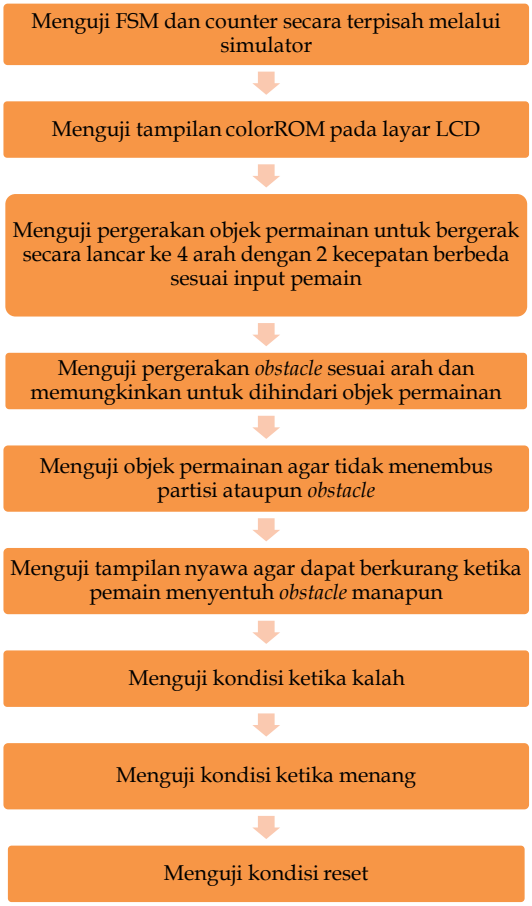
3.4.1 PROSES PEMBUATAN DESAIN PERMAINAN MAZE PUZZLE

Gambar 3-2-1 Diagram Proses Pembuatan Desain Maze Puzzle



Gambar 4-1-2 Susunan Hierarkis Program Maze Puzzle

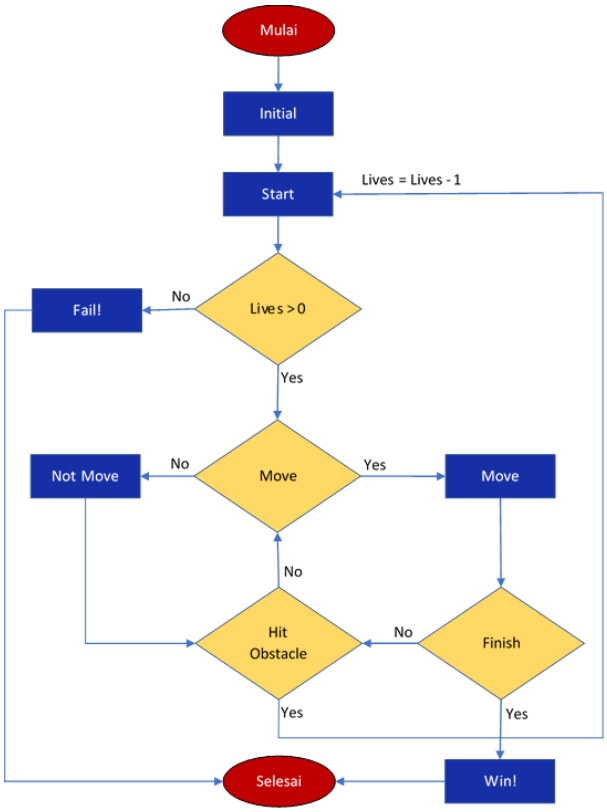
3.4.2 Proses Pengujian Desain Permainan Maze Puzzle



Gambar 3-2-2 Diagram Proses Pengujian Desain Maze Puzzle

4. Hasil dan Analisis

4.1 Cara Kerja Sistem



Gambar 4-2-1 *Flow Chart* Sistem

Sistem permainan Maze Puzzle bekerja dengan *input* dari 6 buah *switch*, yakni SW0 untuk mengatur kecepatan gerak objek permainan, SW1 untuk me-*reset* permainan, dan SW6-SW9 untuk mengatur arah gerak objek permainan. *Output* permainan akan ditampilkan pada layar monitor.

Pada kondisi initial (sebelum memulai permainan), saklar reset harus diatur bernilai 1, akan ditampilkan judul permainan. Selanjutnya ketika reset bernilai 0, maka sistem akan berada pada kondisi start. Pada kondisi start, permainan dimulai dan layout permainan mulai ditampilkan.

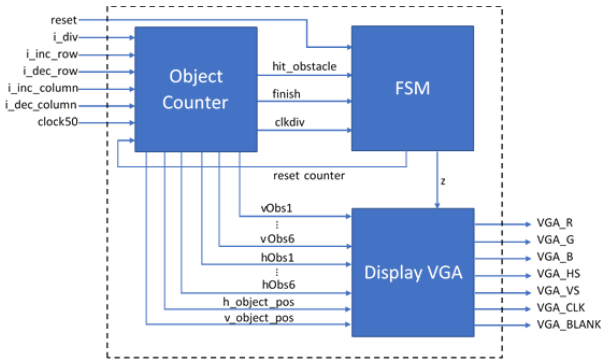
Sebelum pemain menggerakkan objek, dilakukan pengecekan terhadap jumlah nyawa pemain. Pada awalnya pemain akan diberikan 3 nyawa. Apabila nyawa telah habis, maka sistem akan berpindah ke posisi fail dan akan ditampilkan keterangan bahwa pemain gagal menyelesaikan permainan. Apabila pemain masih memiliki nyawa, maka permain boleh menggerakkan objek permainan.

Ketika objek permainan dalam kondisi bergerak ataupun diam, sistem akan dilakukan pengecekan apakah pemain menyentuh obstacle atau tidak. Jika tidak maka pemain bisa melanjutkan pergerakan objek. Apabila pemain bersentuhan dengan

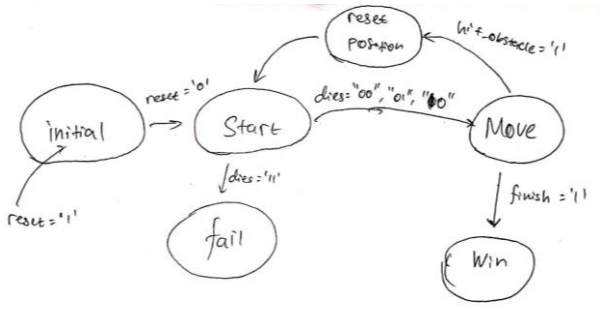
obstacle, maka pemain akan kembali ke posisi start dan nyawa pemain akan berkurang 1.

Apabila pemain telah mencapai garis finish maka sistem akan beralih ke kondisi win dan ditampilkan keterangan bahwa pemain berhasil menyelesaikan permainan. Dengan demikian permainan berakhir. Permainan dapat diulang kembali dengan menyalakan saklar reset.

4.2 STRUKTUR DAN ALGORITMA PROGRAM



Gambar 4-2-1 *Data Path* Program Maze Puzzle



Gambar 4-2-2 Diagram FSM Program Maze Puzzle

Dari gambar 4-2-1, *data path* pada sistem permainan Maze Puzzle terdiri atas 3 blok utama, yakni blok *object counter*, FSM, dan *display* VGA.

Pada blok *object counter* (counter.vhd), terdapat algoritma process dengan label "player_object" yang berfungsi untuk memberikan aturan untuk batasan-batasan gerak dari objek permainan. Selain itu, terdapat process dengan label "obstacle_3", "obstacle_4", "obstacle_5", "obstacle_6" yang berfungsi untuk mengatur gerak obstacle 3, obstacle 4, obstacle 5, obstacle 6. Terdapat pula algoritma process "hit_condition" yang berfungsi untuk mengatur output sinyal hit_condition apakah objek menabrak obstacle atau tidak. Setelah itu ada process "finish_condition" yang berfungsi untuk mengatur sinyal output finish apakah objek telah sampai ke finish atau belum. Pada algoritma tersebut juga terdapat sinyal posisi objek dan setiap obstacle yang di-assign dengan nilai posisi awal, sinyal clock untuk objek dan obstacle, sinyal untuk pergerakan atas atau bawah objek 4 dan 3.

Pada blok *display* VGA, akan diatur tampilan keluaran dalam antarmuka VGA. Salah satu komponen dari blok *display* adalah colorrom.vhd. Pada komponen color rom, terdapat 4 tampilan yang akan ditunjukkan ke layar LCD yang ditentukan dari state pada saat ini. Tampilan pertama adalah tulisan "Maze Puzzle" yang akan ditampilkan ketika state berada pada state initial. Ketika state berada pada state start atau state move, yang ditampilkan adalah layar interface game dan nyawa yang ditampilkan tergantung dari berapa banyak jumlah kematian yang telah terjadi. Kemudian tampilan "You Win" akan ditampilkan pada state win dan tampilan "NOOB!" akan ditampilkan pada state fail.

Pada blok FSM, terdapat 6 state yaitu "initial", "start", "move", "win", "fail", "reset". Alur state machine dari algoritma tersebut adalah permainan dimulai dari state initial, setelah itu ke state start untuk dicek apakah pemain sudah mati sebanyak 3 kali atau belum. Jika sudah, maka state selanjutkan akan ke state fail, namun jika belum, akan ke state move. Pada state move ini pemain akan menggerakkan objek sampai akhirnya mencapai finish atau menabrak obstacle. Jika objek menabrak obstacle, state selanjutnya adalah state reset yang berfungsi untuk mereset semua posisi objek dan pada state reset pula jumlah mati ditambah 1. Jika objek mencapai finish, state selanjutnya adalah state win. State win hanya dapat berpindah ke state initial ketika diberikan input reset = '1'.

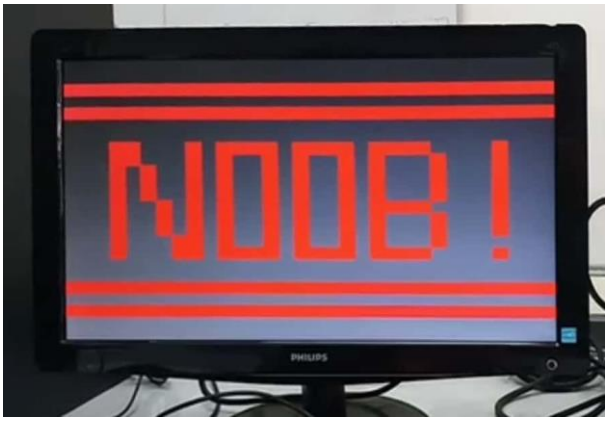
4.3 HASIL PENGUJIAN

No.	Aspek Pengujian	Hasil Pengujian
1	Simulasi FSM dan Counter	Berhasil
2	Tampilan ColorROM pada layar LCD	Berhasil
3	Pergerakan objek permainan ke 4 arah	Berhasil
4	Kecepatan objek permainan yang dapat diubah	Berhasil
5	Pergerakan *obstacle* sesuai arah dan memungkinkan untuk dihindari objek permainan	Berhasil
6	Objek permainan tidak menembus partisi ataupun *obstacle*	Berhasil
7	Tampilan nyawa berkurang ketika pemain menyentuh *obstacle* manapun	Berhasil
8	Kondisi menang	Berhasil
9	Kondisi kalah	Berhasil
10	Kondisi reset	Berhasil

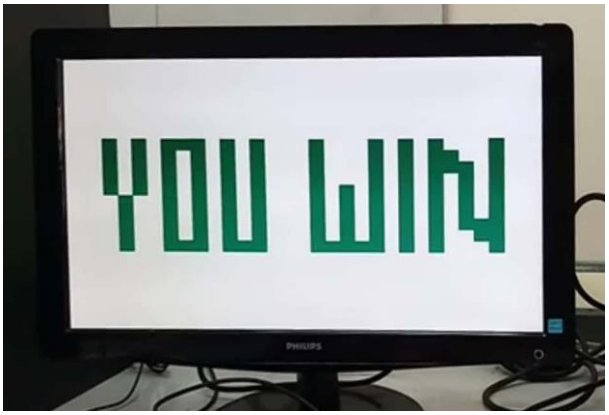
Tabel 4-3-1 Hasil Pengujian Desain Maze Puzzle

Gambar 4-3-1 Tampilan Judul Permainan

Gambar 4-3-2 Tampilan *Layout* Permainan pada Layar LCD



Gambar 4-3-3 Tampilan Apabila Pemain Kalah dalam Permainan



Gambar 4-3-4 Tampilan Apabila Pemain Berhasil Memenangkan Permainan

Pada awal pengujian, beberapa hal kembali ditinjau dan disempurnakan, baik dari *layout* hingga metode menggerakkan objek.

Untuk menggerakkan objek permainan, pada awalnya, kami menggunakan 4 buah *push button*. Namun dari hasil pengujian ditemukan bahwa objek cenderung sulit untuk digerakkan dan kurang responsif karena kondisi *push button* yang kurang baik. Solusinya adalah mengganti *input* dengan menggunakan *switch*. Dengan mengganti input pergerakan objek ke *switch*, masalah berhasil diselesaikan, meskipun harus mengurangi aspek *user experience*. Untuk pengembangan selanjutnya, diharapkan dapat menggunakan *joystick* untuk memudahkan pemain menggerakkan objek.

Selain itu, ditemukan masalah lainnya yang membuat permainan terhenti ketika *layout* program dibuat semakin rumit. Hal tersebut mungkin disebabkan keterbatasan pemrosesan pada FPGA sehingga terjadi *overload* ketika program dibuat semakin rumit. Oleh karena itu, dilakukan optimalisasi dalam program sehingga tingkat kerumitan program dapat dikurangi tanpa mempengaruhi *output* program secara signifikan.

Pada pengujian akhir, diketahui bahwa permainan dapat berjalan dengan baik dan sesuai dengan spesifikasi yang direncanakan. Seluruh fitur permainan baik tampilan, nyawa, rintangan, maupun pergerakan objek dapat bekerja dengan baik.

5. KESIMPULAN

Dari percobaan tersebut, dapat disimpulkan bahwa:

1. Permainan Maze Puzzle berhasil dirancang dengan baik dan berhasil diimplementasikan pada FPGA dengan output layar LCD.
2. Pembuatan program permainan Maze Puzzle berhasil menjadi sarana untuk mengenal dan mempelajari perancangan sistem digital, khususnya melalui pemrograman VHDL, VGA interface, dan implementasi pada FPGA.

DAFTAR PUSTAKA

[1] Stephen Brown, Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design 3rd edition, Mc. GrawHill, New York, 2009.

[2] Mervin T. Hutabarat, dkk. *Petunjuk Praktikum Sistem Digital,* ITB, Bandung, 2019.

[3] https://www.computerhope.com/jargon/v/vga.htm, diakses pada 25 November 2019 pukul 22.30.

[4] Mervin T. Hutabarat, dkk. *Petunjuk Praktikum Sistem Digital,* ITB, Bandung, 2019.

[5] Ibid.

[6] Ibid.

[7] https://www.digikey.com/eewiki/pages/viewpage.action?pageId=15925278, diakses pada 25 November 2019 pukul 23.10.

[8] http://martin.hinner.info/vga/timing.html, diakses pada 25 November 2019 pukul 23.20.

[9] Mervin T. Hutabarat, dkk. *Petunjuk Praktikum Sistem Digital,* ITB, Bandung, 2019.

[10] Ibid.

LAMPIRAN

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
USE  IEEE.STD_LOGIC_ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY top_level_vhd  IS
     PORT(
     i_div          : IN STD_LOGIC;
     reset          : IN STD_LOGIC;
     CLOCK_50       : IN STD_LOGIC;
     SW             : IN STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     PB             : IN STD_LOGIC_VECTOR( 3 DOWNTO 0 );
     i_inc_row : IN STD_LOGIC;
     i_dec_row : IN STD_LOGIC;
     i_inc_column: IN STD_LOGIC;
     i_dec_column: IN STD_LOGIC;
     VGA_R          : OUT STD_LOGIC_VECTOR( 5 DOWNTO 0 );
     VGA_G          : OUT STD_LOGIC_VECTOR( 5 DOWNTO 0 );
     VGA_B          : OUT STD_LOGIC_VECTOR( 5 DOWNTO 0 );
     VGA_HS         : OUT STD_LOGIC;
     VGA_VS         : OUT STD_LOGIC;
     VGA_CLK        : OUT STD_LOGIC;
     VGA_BLANK      : OUT STD_LOGIC;
     GPIO_0         : OUT STD_LOGIC_VECTOR( 35 DOWNTO 0 );
     LEDR           : OUT STD_LOGIC_VECTOR( 9 DOWNTO 0 ));
END top_level_vhd;

ARCHITECTURE behavioral OF top_level_vhd  IS


SIGNAL  hPos : STD_LOGIC_VECTOR ( 9 DOWNTO 0 );
SIGNAL  vPos : STD_LOGIC_VECTOR ( 9 DOWNTO 0 );
SIGNAL
v Obs1,v Obs2,v Obs3,v Obs4,v Obs5,v Obs6,h Obs1,h_Obs2,h_Obs3,h_Obs4,
h Obs5,h Obs6 : STD LOGIC VECTOR (9 DOWNTO 0);
SIGNAL  z signal : STD LOGIC VECTOR ( 4 DOWNTO 0 );
SIGNAL finish_signal, hit_obstacle_signal, resetcounter_signal,clk :
STD_LOGIC;


COMPONENT display_vhd  IS
     PORT(
     i clk          : IN STD LOGIC;
     reset            : IN STD LOGIC;
     i hPos           : IN STD LOGIC VECTOR ( 9 DOWNTO 0 );
     i vPos           : IN STD LOGIC VECTOR ( 9 DOWNTO 0 );
     z                : IN STD_LOGIC_VECTOR( 4 DOWNTO 0 );
     vObs1                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs1                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs2                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs2                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs3                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs3                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs4                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs4                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs5                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs5                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs6                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs6                 :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     VGA_R          : OUT STD_LOGIC_VECTOR( 5 DOWNTO 0 );
     VGA_G          : OUT STD_LOGIC_VECTOR( 5 DOWNTO 0 );
     VGA_B          : OUT STD_LOGIC_VECTOR( 5 DOWNTO 0 );
     VGA_HS         : OUT STD_LOGIC;
     VGA_VS         : OUT STD_LOGIC;
     VGA_CLK        : OUT STD_LOGIC;
     VGA_BLANK      : OUT STD_LOGIC);
END COMPONENT;

COMPONENT counter IS
     PORT(
     i clk                  :  IN   STD LOGIC;
     i div                  :  IN   STD LOGIC;
     reset                  :  IN   STD LOGIC;
     i inc row              :  IN   STD LOGIC;
     i_dec_row              :  IN   STD_LOGIC;
     i_inc_column       :  IN   STD_LOGIC;
     i_dec_column       :  IN   STD_LOGIC;
     o obj row          :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
     o obj column       :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
     vObs1                  :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
     hObs1                  :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs2                  :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs2                  :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs3                  :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs3                  :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs4                  :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs4                  :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
     vObs5                  :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
     hObs5                  :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
     vObs6                  :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
     hObs6                  :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     o_clk                  :  OUT  STD_LOGIC;
     finish                 :  OUT  STD_LOGIC;
     hit_obstacle       :  OUT  STD_LOGIC);
END COMPONENT;

COMPONENT maze IS
     PORT(
     i clk              : IN  STD LOGIC;
     reset              : IN  STD LOGIC;
     finish             : IN  STD_LOGIC;
     hit_obstacle   : IN STD_LOGIC;
     z                  : OUT STD_LOGIC_VECTOR( 4 DOWNTO 0 );
     resetcounter   : OUT STD_LOGIC);
END COMPONENT;


BEGIN

module_vga : display_vhd
     PORT MAP (
     i_clk              => CLOCK_50,
     reset              => reset,
     i hPos             => hPos,
     i vPos             => vPos,
     z                  => z_signal,
     vObs1              => v Obs1,
     hObs1              => h_Obs1,
     vObs2              => v_Obs2,
     hObs2              => h_Obs2,
     vObs3              => v_Obs3,
     hObs3              => h Obs3,
     vObs4              => v Obs4,
     hObs4              => h Obs4,
     vObs5              => v Obs5,
     hObs5              => h_Obs5,
     vObs6              => v_Obs6,
     hObs6              => h_Obs6,
     VGA_R              => VGA_R,
     VGA_G              => VGA_G,
     VGA_B              => VGA_B,
     VGA_HS             => VGA_HS,
     VGA_VS             => VGA_VS,
     VGA_CLK            => VGA_CLK,
     VGA_BLANK      => VGA_BLANK);

pos_counter : counter
     PORT MAP (
     i_clk              => CLOCK_50,
     i_div              => i_div,
     reset              => reset,
     i inc row      => i inc row,
     i dec row      => i dec row,
     i inc column   => i inc column,
     i_dec_column   => i_dec_column,
     o_obj_row      => vPos,
     o_obj_column   => hPos,
     vObs1              => v Obs1,
     hObs1              => h Obs1,
     vObs2              => v Obs2,
     hObs2              => h Obs2,
     vObs3              => v Obs3,
     hObs3              => h Obs3,
     vObs4              => v Obs4,
     hObs4              => h_Obs4,
     vObs5              => v_Obs5,
     hObs5              => h Obs5,
     vObs6              => v Obs6,
     hObs6              => h Obs6,
     o clk              => clk,
     finish             => finish signal,
     hit_obstacle   => hit_obstacle_signal);

fsm : maze
     PORT MAP (
     i_clk              => clk,
     reset              => reset,
     finish             => finish signal,
     hit obstacle   => hit obstacle signal,
     z                  => z signal,
     resetcounter   => resetcounter_signal);

END behavioral;
```

Gambar 6-1 Kode VHDL top_level_vhd.vhd

```
LIBRARY  IEEE;
USE  IEEE.STD LOGIC 1164.ALL;
USE  IEEE.STD LOGIC ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY counter  IS
     PORT(
          i clk               :  IN   STD LOGIC;
          i div               :  IN   STD LOGIC;
          reset               :  IN   STD LOGIC;
          i_inc_row           :  IN   STD_LOGIC;
          i_dec_row           :  IN   STD_LOGIC;
          i_inc_column        :  IN   STD_LOGIC;
          i_dec_column        :  IN   STD_LOGIC;
          o_obj_row           :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          o obj column        :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
          vObs1               :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
          hObs1               :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
          vObs2               :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
          hObs2               :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          vObs3               :  OUT  STD LOGIC VECTOR( 9 DOWNTO 0 );
          hObs3               :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          vObs4               :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          hObs4               :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          vObs5               :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          hObs5               :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          vObs6               :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          hObs6               :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
          o clk               :  OUT  STD LOGIC;
          finish              :  OUT  STD_LOGIC;
          hit_obstacle        :  OUT  STD_LOGIC
          );
END counter;

ARCHITECTURE behavioral OF counter  IS

SIGNAL count row         : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0010010010"; -- 54
SIGNAL count_column      : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0010000001"; -- 10
SIGNAL obs1 row               : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0000101101"; -- 45
SIGNAL obs1 column       : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0010000001"; -- 129
SIGNAL obs2 row               : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0010000001"; -- 129
SIGNAL obs2_column       : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0011100101"; -- 229
SIGNAL obs3_row               : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0000101101"; -- 45
SIGNAL obs3 column       : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0101001001"; -- 329
SIGNAL obs4 row               : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0001110111"; -- 119
SIGNAL obs4_column       : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0110101101"; -- 429
SIGNAL obs5_row               : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0011000011"; -- 195
SIGNAL obs5_column       : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0110101101"; -- 429
SIGNAL obs6 row               : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0100101001"; -- 297
SIGNAL obs6 column       : STD_LOGIC_VECTOR( 9 DOWNTO 0 ) :=
"0010000001"; -- 129
SIGNAL obs3_atas         : STD_LOGIC := '0';
SIGNAL obs4_atas         : STD_LOGIC := '1';
SIGNAL clock_obj         : STD_LOGIC;
SIGNAL clock_obs         : STD_LOGIC;


component CLOCKDIV is port(
     CLK: IN std logic;
     IN DIV : IN std logic;
     O_DIVOUT: OUT std_logic);
end component;

BEGIN

-- Counter Objek Player --
player object : PROCESS(clock_obj, reset)
     BEGIN
     if (reset = '1') then
          count row <= "0000110110";
          count_column <= "0000001010";
     else
          IF(  clock obj'EVENT  ) AND ( clock obj  = '1'  ) then
               if (i inc row = '0' and i dec row = '1') then
                    if ((count row >= 45 and count row < 179-33) or
                         (count row >= 195 and count row < 330-33) or
                         (count row >= 345 and count row < 479-33))then
                              count_row <= count_row + 1;
                    elsif (count_row>=179-33 and count_row <= 195 and
count_column >= 505) or
                              (count_row >= 329-33 and count_row <= 345
and count_column <= 102) then
                         count_row <= count_row + 1;
                    end if;
               elsif (i dec row = '0' and i inc row = '1') then
                    if ((count row > 45 and count row <= 179-33) or
                         (count_row > 195 and count_row <= 330-33) or
                         (count_row > 345 and count_row <= 479-33))then
                         count_row <= count_row - 1;
                    elsif (count_row>= 179-33 and count_row <= 195 and
count_column >= 505) or
                              (count_row >= 329-33 and count_row <= 345
and count column <= 102) then
                         count_row <= count_row - 1;
                    end if;
               end if;
               if (i_inc_column = '0' and i_dec_column = '1') then
                    if ((count_row >= 45 and count_row <= 179-33 and
count_column < 639-33) or
                        (count_row >= 195 and count_row <= 330-33 and
count column < 639-33) or
                        (count_row >= 345 and count_row <= 479-33 and
count column < 639-33))then
                         count column <= count column + 1;
                    elsif (count row>=179-33 and count_row <= 195 and
count_column >= 505 and count_column < 639-33) or
                              (count_row >= 329-33 and count_row <= 345
and count column < 135-33) then
                         count_column <= count_column + 1;
                    end if;
```

```
elsif (i dec column = '0' and i inc column = '1') then
                    if ((count_row >= 45 and count_row <= 179-33 and
count_column > 0) or
                         (count_row >= 195 and count_row <= 330-33 and
count column > 0) or
                         (count_row >= 345 and count_row <= 479-33 and
count column > 0))then
                         count column <= count column - 1;
                    elsif (count_row >= 179-33 and count_row <= 195 and
count_column > 505) or
                          (count_row >= 329-33 and count_row <= 345 and
count_column > 0 and count_column <= 135-33) then
                         count_column <= count_column - 1;
                    end if;
               end if;
          END IF;
     end if;
END  PROCESS;

-- Counter Obstacle 3 --
obstacle_3 : PROCESS(clock_obs, reset)
     BEGIN
     if (reset = '1') then
          obs3 row <= "0010000001";
          obs3 column <= "0101001001";
          obs3_atas <= '0';
     else
          IF((  clock_obs'EVENT  ) AND ( clock_obs  = '1'  )) then
               IF (obs3_atas = '1') THEN
                    IF (obs3_row > 45) THEN
                         obs3_row <= obs3_row - 1;
                    ELSE
                         obs3_atas <= '0';
                    end IF;
               ELSIF (obs3 atas  = '0') THEN
                    IF (obs3 row < 119+27) THEN
                    obs3_row <= obs3_row + 1;
                    ELSE
                         obs3_atas <= '1';
                    end IF;
               END IF;
          END IF;
     end if;
END  PROCESS;

-- Counter Obstacle 4 --
obstacle_4 : PROCESS(clock_obs, reset)
     BEGIN
     if (reset = '1') then
          obs4 row <= "0001110111";
          obs4 column <= "0110101101";
          obs4_atas <= '1';
     else
          IF((  clock_obs'EVENT  ) AND ( clock_obs  = '1'  )) then
               IF (obs4_atas = '1') THEN
                    IF (obs4_row > 45) THEN
                         obs4_row <= obs4_row - 1;
                    ELSE
                         obs4_atas <= '0';
                    end IF;
               ELSIF (obs4 atas  = '0') THEN
                    IF (obs4 row < 119+27) THEN
                    obs4_row <= obs4_row + 1;
                    ELSE
                         obs4_atas <= '1';
                    end IF;
               END IF;
          END IF;
     end if;
END  PROCESS;

-- Counter Obstacle 5 --
obstacle_5 : PROCESS(clock_obs, reset)
     BEGIN
     if (reset = '1') then
          obs5 row <= "0011000011";
          obs5_column <= "0110101101";
     else
          IF(  clock obs'EVENT  ) AND ( clock obs  = '1'  ) then
               IF (obs5 row < 330-33 ) AND (obs5_column = 429) THEN
                    obs5 row <= obs5 row + 1;
               ELSIF ( obs5_column > 129) AND (obs5_row = 330-33) THEN
                    obs5_column <= obs5_column - 1;
               ELSIF ( obs5 row > 195 ) AND ( obs5_column = 129 ) THEN
                    obs5 row <= obs5 row - 1;
               ELSIF ( obs5 column < 429 ) AND ( obs5_row = 195 ) THEN
                    obs5_column <= obs5_column + 1;
               END IF;
          END IF;
     end if;
END  PROCESS;

-- Counter Obstacle 6 --
obstacle 6 : PROCESS(clock_obs, reset)
     BEGIN
     if (reset = '1') then
          obs6 row <= "0100101001";
          obs6_column <= "0010000001";
     else
          IF(  clock_obs'EVENT  ) AND ( clock_obs  = '1'  ) then
               IF (obs6_row < 330-33 ) AND (obs6_column = 429) THEN
                    obs6_row <= obs6_row + 1;
               ELSIF ( obs6_column > 129) AND (obs6_row = 330-33) THEN
                    obs6 column <= obs6 column - 1;
               ELSIF ( obs6 row > 195 ) AND ( obs6_column = 129 ) THEN
                    obs6 row <= obs6 row - 1;
               ELSIF ( obs6 column < 429 ) AND ( obs6_row = 195 ) THEN
                    obs6_column <= obs6_column + 1;
               END IF;
          END IF;
     end if;
END  PROCESS;

-- Mengeluarkan output posisi objek dan setiap obstacle --
vObs1 <= obs1 row;
hObs1 <= obs1 column;
vObs2 <= obs2_row;
hObs2 <= obs2_column;
vObs3 <= obs3 row;
hObs3 <= obs3 column;
vObs4 <= obs4_row;
```

```
hObs4 <= obs4_column;
vObs5 <= obs5_row;
hObs5 <= obs5_column;
vObs6 <= obs6_row;
hObs6 <= obs6_column;

o_obj_row <= count_row;
o_obj_column <= count_column;

-- Kondisi dimana objek menabrak obstacle --
hit_condition : PROCESS(count_row, count_column, obs1_row, obs2_row,
obs3_row, obs4_row, obs5_row, obs6_row,
                        obs1_column, obs2_column, obs3_column,
obs4_column, obs5_column, obs6_column)
-- Kondisi objek beririsan dengan obstacle (panjang obstacle belum
ditentukan) --
BEGIN
     IF ((obs1_row-33 <= count_row AND count_row <= obs1_row+50 AND
count_column+33 >= obs1_column AND count_column <= obs1_column+50 ) OR
          (obs2_row-33 <= count_row AND count_row <= obs2_row+50 AND
count_column+33 >= obs2_column AND count_column <= obs2_column+50 ) OR
          (obs3_row-33 <= count_row AND count_row <= obs3_row+33 AND
count_column+33 >= obs3_column AND count_column <= obs3_column+33 ) OR
          (obs4_row-33 <= count_row AND count_row <= obs4_row+33 AND
count_column+33 >= obs4_column AND count_column <= obs4_column+33) OR
          (obs5_row <= count_row+33 AND count_row <= obs5_row+33 AND
count_column+33 >= obs5_column AND count_column <= obs5_column+33 ) OR
          (obs6_row <= count_row+33 AND count_row <= obs6_row+33 AND
count_column+33 >= obs6_column AND count_column <= obs6_column+33 )
) THEN
               hit_obstacle <= '1';
     ELSE
          hit_obstacle <= '0';
     END IF;
END PROCESS;

-- Kondisi dimana objek mencapai finish (garis finish masih belum
ditentukan) --
finish_condition : PROCESS(count_row, count_column)
BEGIN
     IF (count_row = 312) THEN
          finish <= '1';
     ELSE
          finish <= '0';
     END IF;
END PROCESS;

-- Keluaran clockdiv --
obj_clock : clockdiv port map(i_clk, i_div, clock_obj);
obs_clock : clockdiv port map(i_clk, '0', clock_obs);
o_clk <= clock_obs;

END behavioral;
```

Gambar 6-2 Kode VHDL counter.vhd

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
USE  IEEE.STD_LOGIC_ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY maze IS
     PORT(
     i_clk              : IN  STD_LOGIC;
     skip          : IN  STD_LOGIC;
     reset              : IN  STD_LOGIC;
     finish             : IN  STD_LOGIC;
     hit_obstacle  : IN STD_LOGIC;
     z                  : OUT STD_LOGIC_VECTOR( 4 DOWNTO 0 );
     resetcounter  : OUT STD_LOGIC);
END maze;

ARCHITECTURE FSM OF maze IS

TYPE state_type IS (initial, start, move, win, fail);
signal state       : state_type;
signal dies             : STD_LOGIC_VECTOR (1 DOWNTO 0) := "00";

BEGIN

PROCESS(i_clk,skip,reset)
BEGIN
     IF reset = '1' THEN
          state <= initial;
     ELSIF rising_edge(i_clk) THEN
          CASE state IS
               WHEN initial =>
                    IF (skip = '1') THEN
                         state <= initial;
                    ELSE
                         state <= start;
                    END IF;
               WHEN start =>
                    IF dies < 3 THEN
                         state <= move;
                    ELSE
                         state <= fail;
                    END IF;
               WHEN move =>
                    IF (hit_obstacle = '1') THEN
                         state <= start;
                         dies <= dies + 1;
                    ELSIF (finish = '1') THEN
                         state <= win;
                    ELSE
                         state <= move;
                    END IF;
               WHEN win =>
                    state <= win;
               WHEN fail =>
                    state <= fail;
          END CASE;
     END IF;
END PROCESS;

PROCESS (state, dies)
     BEGIN
          IF state = initial THEN
               z <= "00000";
          ELSIF state = start AND dies = 0 THEN
               z <= "00001";
               resetcounter <= '1';
          ELSIF state = start AND dies = 1 THEN
               z <= "10001";
               resetcounter <= '1';
          ELSIF state = start AND dies = 2 THEN
               z <= "11010";
               resetcounter <= '1';
          ELSIF state = move AND dies = 0 THEN
               z <= "00011";
          ELSIF state = move AND dies = 1 THEN
               z <= "10011";
          ELSIF state = move AND dies = 2 THEN
               z <= "11011";
          ELSIF state = win THEN
               z <= "11111";
          ELSIF state = fail THEN
               z <= "00100";
          END IF;
     END PROCESS;
END FSM;
```

Gambar 6-3 Kode VHDL maze.vhd

```
LIBRARY  IEEE;
USE  IEEE.STD LOGIC 1164.ALL;
USE  IEEE.STD LOGIC ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY display_vhd  IS
     PORT(
     i clk           : IN STD LOGIC;
     reset              : IN STD LOGIC;
     i hPos             : IN STD LOGIC VECTOR ( 9 DOWNTO 0 );
     i_vPos             : IN STD_LOGIC_VECTOR ( 9 DOWNTO 0 );
     z                  : IN STD_LOGIC_VECTOR( 4 DOWNTO 0 );
     vObs1              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs1              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs2              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs2              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs3              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs3              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs4              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs4              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs5              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs5              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs6              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs6              :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     VGA_R           : OUT STD_LOGIC_VECTOR( 5 DOWNTO 0 );
     VGA G           : OUT STD LOGIC VECTOR( 5 DOWNTO 0 );
     VGA B           : OUT STD LOGIC VECTOR( 5 DOWNTO 0 );
     VGA HS          : OUT STD LOGIC;
     VGA VS          : OUT STD LOGIC;
     VGA_CLK         : OUT STD_LOGIC;
     VGA_BLANK       : OUT STD_LOGIC);
END display_vhd;

ARCHITECTURE behavioral OF display_vhd  IS

SIGNAL  red        : STD LOGIC VECTOR (5 DOWNTO 0);
SIGNAL  green      : STD LOGIC VECTOR (5 DOWNTO 0);
SIGNAL  blue       : STD LOGIC VECTOR (5 DOWNTO 0);
SIGNAL  red_color  : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL  green_color     : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL  blue color : STD LOGIC VECTOR (7 DOWNTO 0);
SIGNAL  pixel row  : STD LOGIC VECTOR (9 DOWNTO 0);
SIGNAL  pixel column: STD LOGIC_VECTOR (9 DOWNTO 0);
SIGNAL  red on     : STD LOGIC;
SIGNAL  green on   : STD LOGIC;
SIGNAL  blue_on         : STD_LOGIC;
SIGNAL
v_Obs1,v_Obs2,v_Obs3,v_Obs4,v_Obs5,v_Obs6,h_Obs1,h_Obs2,h_Obs3,h_Obs4,
h_Obs5,h_Obs6 : STD_LOGIC_VECTOR (9 DOWNTO 0);


COMPONENT vga  IS
     PORT(
     i clk              : IN STD LOGIC;
     i_red              : IN STD_LOGIC;
     i_green            : IN STD_LOGIC;
     i_blue             : IN STD_LOGIC;
     o_red              : OUT STD_LOGIC;
     o_green            : OUT STD_LOGIC;
     o_blue             : OUT STD_LOGIC;
     o horiz sync   : OUT STD LOGIC;
     o vert sync        : OUT STD LOGIC;
     o pixel row        : OUT STD LOGIC VECTOR( 9 DOWNTO 0 );
     o pixel column: OUT STD_LOGIC_VECTOR( 9 DOWNTO 0 ));
END COMPONENT;

COMPONENT color_rom_vhd  IS
     PORT(
     i_hPos             : IN  STD_LOGIC_VECTOR ( 9 DOWNTO 0 );
     i vPos             : IN  STD LOGIC VECTOR ( 9 DOWNTO 0 );
     i pixel column  : IN  STD LOGIC VECTOR ( 9 DOWNTO 0 );
     i pixel row     : IN  STD LOGIC VECTOR ( 9 DOWNTO 0 );
     z                  : IN  STD LOGIC VECTOR ( 4 DOWNTO 0 );
     vObs1                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs1                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs2                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs2                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs3                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs3                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs4                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs4                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs5                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs5                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     vObs6                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     hObs6                   :  IN  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
     o red           : OUT STD LOGIC VECTOR ( 7 DOWNTO 0 );
     o green         : OUT STD LOGIC VECTOR ( 7 DOWNTO 0 );
     o blue          : OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0 ));
END COMPONENT;

BEGIN

vga_driver0 : vga
     PORT MAP (
     i_clk           => i_clk,
     i red           => '1',
     i green         => '1',
     i blue          => '1',
     o red           => red on,
     o_green         => green_on,
     o_blue          => blue_on,
     o_horiz_sync    => VGA_HS,
     o_vert_sync     => VGA_VS,
     o_pixel_row     => pixel_row,
     o_pixel_column  => pixel_column);

color rom0 : color_rom_vhd
     PORT MAP (
     i hPos             => i hPos,
     i_vPos             => i_vPos,
     i_pixel_column     => pixel_column,
     i_pixel_row        => pixel_row,
     z                  => z,
     vObs1              => vObs1,
     hObs1              => hObs1,
     vObs2              => vObs2,
     hObs2              => hObs2,
     vObs3              => vObs3,
     hObs3              => hObs3,
     vObs4              => vObs4,
     hObs4              => hObs4,
     vObs5              => vObs5,
     hObs5              => hObs5,
     vObs6              => vObs6,
     hObs6              => hObs6,
     o red              => red color,
     o green            => green color,
     o_blue             => blue_color);

red   <= red color  (7 DOWNTO 2) ;
green <= green color(7 DOWNTO 2) ;
blue  <= blue_color (7 DOWNTO 2) ;


PROCESS(red_on,green_on,blue_on,red,green,blue)
BEGIN

  IF (red_on = '1'  ) THEN VGA_R <=  red;
  ELSE  VGA_R <=  "000000";
  END IF;

  IF (green on = '1'  ) THEN VGA_G <=  green;
  ELSE VGA_G <=  "000000";
  END IF;

  IF (blue_on = '1'  ) THEN VGA_B <=  blue;
  ELSE VGA_B <=  "000000";
  END IF;

END PROCESS;

END behavioral;
```

Gambar 6-4 Kode VHDL display_vhd.vhd

```
LIBRARY  IEEE;
USE  IEEE.STD LOGIC 1164.ALL;
USE  IEEE.STD LOGIC ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY color_rom_vhd  IS
     PORT(
     i hPos              : IN  STD LOGIC VECTOR (9 DOWNTO 0);
     i vPos              : IN  STD LOGIC VECTOR (9 DOWNTO 0);
     i pixel column  : IN  STD LOGIC VECTOR (9 DOWNTO 0);
     i_pixel_row     : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     z                   : IN  STD_LOGIC_VECTOR (4 DOWNTO 0);
     vObs1               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     hObs1               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     vObs2               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     hObs2               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     vObs3               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     hObs3               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     vObs4               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     hObs4               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     vObs5               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     hObs5               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     vObs6               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);
     hObs6               : IN  STD_LOGIC_VECTOR (9 DOWNTO 0);

     o red           : OUT STD LOGIC VECTOR (7 DOWNTO 0);
     o green         : OUT STD LOGIC VECTOR (7 DOWNTO 0);
     o blue          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END color_rom_vhd;

ARCHITECTURE behavioral OF color_rom_vhd  IS

SIGNAL r life1     : STD LOGIC VECTOR (7 DOWNTO 0)  := "11111111";
SIGNAL g life1     : STD LOGIC VECTOR (7 DOWNTO 0)  := "11111111";
SIGNAL b life1     : STD LOGIC VECTOR (7 DOWNTO 0)  := "11111111";
SIGNAL r life2     : STD LOGIC VECTOR (7 DOWNTO 0)  := "11111111";
SIGNAL g life2     : STD LOGIC VECTOR (7 DOWNTO 0)  := "11111111";
SIGNAL b life2     : STD LOGIC VECTOR (7 DOWNTO 0)  := "11111111";
SIGNAL r_life3     : STD_LOGIC_VECTOR (7 DOWNTO 0)  := "11111111";
SIGNAL g_life3     : STD_LOGIC_VECTOR (7 DOWNTO 0)  := "11111111";
SIGNAL b life3     : STD LOGIC VECTOR (7 DOWNTO 0)  := "11111111";
CONSTANT r panel        : STD_LOGIC_VECTOR (7 DOWNTO 0)  :=
"00000000";
CONSTANT g panel   : STD LOGIC VECTOR (7 DOWNTO 0)  := "11111111";
CONSTANT b panel        : STD_LOGIC_VECTOR (7 DOWNTO 0)  :=
"00000000";
CONSTANT r_pnshdw       : STD_LOGIC_VECTOR (7 DOWNTO 0)  :=
"00000000";
CONSTANT g_pnshdw  : STD_LOGIC_VECTOR (7 DOWNTO 0)  := "00000000";
CONSTANT b_pnshdw   : STD_LOGIC_VECTOR (7 DOWNTO 0) := "00000000";
CONSTANT r grid    : STD LOGIC VECTOR (7 DOWNTO 0)  := X"F2";
CONSTANT g grid         : STD LOGIC VECTOR (7 DOWNTO 0)  := X"F2";
CONSTANT b_grid    : STD_LOGIC_VECTOR (7 DOWNTO 0)  := X"F2";


SIGNAL grid         : STD_LOGIC;

BEGIN

PROCESS(i_pixel_row,i_pixel_column, i_vPos, i_hPos,z,
vObs1,vObs2,vObs3,vObs4,vObs5,vObs6,
hObs1,hObs2,hObs3,hObs4,hObs5,hObs6)
BEGIN



     --INITIAL
     IF z = "00000" THEN
          IF (i_pixel_row >= 80 AND i_pixel_row <= 240) AND
          --M
          ((i pixel column >= 120 AND i pixel column <= 140)
          OR (i pixel column >= 200 AND i pixel column <= 220)
          OR (i pixel column >= 160 AND i_pixel_column <= 180 AND
i_pixel_row <= 180)
          OR (i_pixel_column >= 120 AND i_pixel_column <=220 AND
i_pixel_row <= 100)

          --A
          OR (i pixel column >= 240 AND i pixel column <= 260)
          OR (i pixel column >= 300 AND i pixel column <= 320)
          OR (i_pixel_column >= 240 AND i_pixel_column <= 320 AND
i pixel row <= 100)
          OR (i pixel column >= 240 AND i pixel column <= 320 AND
i_pixel_row >= 160 AND i_pixel_row <= 180)

          --Z
          OR (i pixel_column >= 340 AND i_pixel_column <= 420 AND
i pixel row <= 100)
          OR (i pixel column >= 340 AND i pixel column <= 420 AND
i pixel row >= 160 AND i pixel row <= 180)
          OR (i_pixel_column >= 340 AND i_pixel_column <= 420 AND
i_pixel_row >= 220)
          OR (i_pixel_column >= 400 AND i_pixel_column <= 420 AND
i_pixel_row <= 180)
          OR (i_pixel_column >= 340 AND i_pixel_column <= 360 AND
i pixel row >= 160)

          --E
          OR (i pixel column >= 440 AND i pixel column <= 460)
          OR (i_pixel_column >= 460 AND i_pixel_column <= 520 AND
i_pixel_row <= 100)
          OR (i_pixel_column >= 460 AND i_pixel_column <= 510 AND
i_pixel_row >= 160 AND i_pixel_row <= 180)
          OR (i_pixel_column >= 460 AND i_pixel_column <= 520 AND
i_pixel_row >= 220))

          THEN
               o_red <= "00000000"; o_green <="00011000"; o_blue <=
"00010000";


          ELSIF (i_pixel_row >= 280 AND i_pixel_row <= 380) AND
          --P
          ((i pixel column >= 90 AND i pixel column <=110)
          OR (i pixel column >= 130 AND i pixel column <= 150 AND
i pixel row >= 300 AND i pixel row <= 320)
          OR (i_pixel_column >= 110 AND i_pixel_column <= 150 AND
i pixel row <= 300)
          OR (i_pixel_column >= 110 AND i_pixel_column <= 150 AND
i_pixel_row >= 320 AND i_pixel_row <= 340)

          --U
          OR (i pixel column >= 170 AND i pixel column <= 190)
          OR (i_pixel_column >= 210 AND i_pixel_column <= 230)
          OR (i pixel column >= 190 AND i_pixel_column <= 210 AND
i_pixel_row >= 360)

          --Z
          OR (i pixel_column >= 250 AND i_pixel_column <= 310 AND
i pixel row <= 300)
          OR (i pixel column >= 250 AND i pixel column <= 310 AND
i pixel row >= 320 AND i pixel row <= 340)
          OR (i_pixel_column >= 250 AND i_pixel_column <= 310 AND
i_pixel_row >= 360)
          OR (i_pixel_column >= 290 AND i_pixel_column <= 310 AND
i_pixel_row <= 340)
          OR (i_pixel_column >= 250 AND i_pixel_column <= 270 AND
i pixel row >= 320)

          --Z
          OR (i pixel_column >= 330 AND i_pixel_column <= 390 AND
i_pixel_row <= 300)
          OR (i_pixel_column >= 330 AND i_pixel_column <= 390 AND
i_pixel_row >= 320 AND i_pixel_row <= 340)
          OR (i_pixel_column >= 330 AND i_pixel_column <= 390 AND
i_pixel_row >= 360)
          OR (i_pixel_column >= 370 AND i_pixel_column <= 390 AND
i pixel row <= 340)
          OR (i_pixel_column >= 330 AND i_pixel_column <= 350 AND
i pixel row >= 320)

          --L
          OR (i_pixel_column >= 410 AND i_pixel_column <= 430)
          OR (i_pixel_column >= 430 AND i_pixel_column <= 470 AND
i pixel row >= 360)

          --E
          OR (i pixel column >= 490 AND i pixel column <= 510)
          OR (i pixel_column >= 510 AND i_pixel_column <= 550 AND
i pixel row <= 300)
          OR (i_pixel_column >= 510 AND i_pixel_column <= 540 AND
i_pixel_row >= 320 AND i_pixel_row <= 340)
          OR (i pixel column >= 510 AND i_pixel_column <= 550 AND
i pixel row >= 360))

          THEN
               o_red <= "00000000"; o_green <="00011000"; o_blue <=
"00010000";

          --BACKGROUND
          ELSE
               o_red <= "00000000"; o_green <="00000000"; o_blue <=
"00000000";
          END IF;

     --START/MOVE
     ELSIF (z = "00001" OR z = "10001" OR z = "11010" OR z = "00011" OR
z = "10011" OR z = "11011") THEN
          --PANEL
          IF (i_pixel_row <= 40) THEN
               IF (i_pixel_row >= 10 AND i_pixel_row <= 35) THEN
                    --LIFES
                    IF z = "00001" OR z = "00011" THEN
                         r life1 <= X"99";
                         g life1 <= X"00";
                         b life1 <= X"00";
                         r_life2 <= X"99";
                         g_life2 <= X"00";
                         b_life2 <= X"00";
                         r_life3 <= X"99";
                         g_life3 <= X"00";
                         b life3 <= X"00";
                    ELSIF z = "10001" OR z = "10011" THEN
                         r life1 <= X"99";
                         g life1 <= X"00";
                         b_life1 <= X"00";
                         r_life2 <= X"99";
                         g_life2 <= X"00";
                         b_life2 <= X"00";
                         r life3 <= X"00";
                         g life3 <= X"00";
                         b life3 <= X"00";
                    ELSIF z = "11001" OR z = "11011" THEN
                         r life1 <= X"99";
                         g life1 <= X"00";
                         b_life1 <= X"00";
                         r_life2 <= X"00";
                         g life2 <= X"00";
                         b life2 <= X"00";
                         r life3 <= X"00";
                         g life3 <= X"00";
                         b life3 <= X"00";
                    END IF;
                    --LIFES
                    IF (i_pixel_column >= 590 AND i_pixel_column <= 600)
THEN
                         o_red <= r_life1; o_green <= g_life1; o_blue <=
b life1;
                    ELSIF (i_pixel_column >= 605 AND i_pixel_column <=
615)THEN
                         o_red <= r_life2; o_green <= g_life2; o_blue <=
b_life2;
                    ELSIF (i_pixel_column >= 620 AND i_pixel_column <=
630)THEN
                         o_red <= r_life3; o_green <= g_life3; o_blue <=
b_life3;


                    --TULISAN MAZE
                    ELSIF (i_pixel_column >= 10 AND i_pixel_column <= 15)
OR
                         --M
                         ((i_pixel_column >= 15 AND i_pixel_column <= 28
AND i_pixel_row <= 15)
                         OR (i_pixel_column >= 19 AND i_pixel_column <=
24 AND i pixel row <= 30)
                         OR (i_pixel_column >= 28 AND i_pixel_column <=
33)

                         --A
                         OR (i_pixel_column >= 43 AND i_pixel_column <=
48)
                         OR (i_pixel_column >= 53 AND i_pixel_column <=
58)
```

```
                        OR (i_pixel_column >= 48 AND i_pixel_column <=
53 AND i_pixel_row <= 15)
                        OR (i_pixel_column >= 48 AND i_pixel_column <=
53 AND i pixel row >= 25 AND i_pixel_row <= 30)

                        --Z
                        OR (i_pixel_column >= 68 AND i_pixel_column <=
83 AND i pixel row <= 15)
                        OR (i_pixel_column >= 68 AND i_pixel_column <=
83 AND i_pixel_row >= 20 AND i_pixel_row <= 25)
                        OR (i_pixel_column >= 68 AND i_pixel_column <=
83 AND i_pixel_row >= 30)
                        OR (i_pixel_column >= 78 AND i_pixel_column <=
83 AND i pixel row <= 25)
                        OR (i_pixel_column >= 68 AND i_pixel_column <=
73 AND i pixel row >= 20)

                        --E
                        OR (i_pixel_column >= 93 AND i_pixel_column <=
98)
                        OR (i_pixel_column >= 98 AND i_pixel_column <=
108 AND i_pixel_row <= 15)
                        OR (i_pixel_column >= 98 AND i_pixel_column <=
108 AND i pixel row >= 20 AND i pixel row <= 25)
                        OR (i_pixel_column >= 98 AND i_pixel_column <=
108 AND i_pixel_row >= 30))


                        THEN
                        o_red <= "00000000"; o_green <="00011000";
o_blue <= "00010000";

                    ELSE
                        o_red <= r_panel; o_green <= g_panel; o_blue
<= b panel;
                    END IF;
                ELSE
                    o_red <= r_panel; o_green <= g_panel; o_blue <=
b panel;
                END IF;

            --BATAS PANEL
            ELSIF (i pixel row > 40 AND i pixel row <= 44) THEN
            o_red <= r_pnshdw; o_green <= g_pnshdw; o_blue <= b_pnshdw;

            --TEMBOK
            ELSIF (i_pixel_row >= 180 AND i_pixel_row <= 194 AND
i_pixel_column <= 504)
            OR (i pixel row >= 330 AND i_pixel_row <= 344 AND
i pixel column >= 136) THEN
            o_red <= "00000000"; o_green <= "00000000"; o_blue <=
"00000000";

            --FINISH AREA
            ELSIF i_pixel_row >= 345 THEN
                IF (i_pixel_column >= 540 AND i_pixel_column <= 580 AND
i_pixel_row <= 400) THEN
                o_red <= X"FF"; o_green <= X"FF"; o_blue <= X"99";

                ELSIF (i_pixel_row >= 360 AND i_pixel_row <= 460 AND (
                --F
                (i pixel column >= 40 AND i pixel column <= 60)
                OR (i_pixel_column >= 60 AND i_pixel_column <= 120 AND
i_pixel_row <= 380)
                OR (i_pixel_column >= 60 AND i_pixel_column <= 100 AND
i_pixel_row >= 400 AND i_pixel_row <= 420)

                --I
                OR (i_pixel_column >= 140 AND i_pixel_column <= 160)

                --N
                OR (i_pixel_column >= 200 AND i_pixel_column <= 220)
                OR (i_pixel_column >= 220 AND i_pixel_column <= 240 AND
i_pixel_row <= 400)
                OR (i_pixel_column >= 240 AND i_pixel_column <= 260 AND
i pixel row >= 380 AND i pixel row <= 440)
                OR (i_pixel_column >= 260 AND i_pixel_column <= 280 AND
i pixel row >= 420)
                OR (i_pixel_column >= 280 AND i_pixel_column <= 300)

                --I
                OR (i_pixel_column >= 340 AND i_pixel_column <= 360)

                --S
                OR (i_pixel_column >= 400 AND i_pixel_column <= 480 AND
i pixel row <= 380)
                OR (i pixel column >= 400 AND i_pixel_column <= 480 AND
i pixel row >= 400 AND i pixel row <= 420)
                OR (i_pixel_column >= 400 AND i_pixel_column <= 480 AND
i_pixel_row >= 440)
                OR (i_pixel_column >= 400 AND i_pixel_column <= 420 AND
i_pixel_row <= 420)
                OR (i_pixel_column >= 460 AND i_pixel_column <= 480 AND
i pixel row >= 400)

                --H
                OR (i pixel column >= 520 AND i pixel column <= 540)
                OR (i_pixel_column >= 540 AND i_pixel_column <= 580 AND
i_pixel_row <= 420)
                OR (i_pixel_column >= 580 AND i_pixel_column <= 600)

                )) THEN
                o_red <= X"4D"; o_green <= X"4D"; o_blue <= X"00";


                ELSE
                o red <= X"FF"; o_green <= X"FF"; o_blue <= X"99";
                END IF;
            --OBSTACLE
            ELSIF (i_pixel_row >= vObs1 AND i_pixel_row <= vObs1+50 AND
i_pixel_column >= hObs1 AND i_pixel_column <= hObs1+50)
            OR  (i pixel row >= vObs2 AND i pixel row <= vObs2+50 AND
i pixel column >= hObs2 AND i pixel column <= hObs2+50)
            OR  (i pixel row >= vObs3 AND i pixel row <= vObs3+33 AND
i pixel column >= hObs3 AND i pixel column <= hObs3+33)
            OR  (i pixel row >= vObs4 AND i pixel row <= vObs4+33 AND
i_pixel_column >= hObs4 AND i_pixel_column <= hObs4+33)
            OR  (i_pixel_row >= vObs5 AND i_pixel_row <= vObs5+33 AND
i pixel column >= hObs5 AND i pixel column <= hObs5+33)
            OR  (i pixel row >= vObs6 AND i pixel row <= vObs6+33 AND
i pixel column >= hObs6 AND i_pixel_column <= hObs6+33)
```

```
            THEN
            o_red <= "11111111"; o_green <= "00000000"; o_blue <=
"00000000";

            --OBJEK PERMAINAN
            ELSIF (i pixel row >= i vPos AND i pixel row <= (i vPos +
33) AND i_pixel_column >= i_hPos AND i_pixel_column <= (i_hPos + 33) )
THEN
                o_red <= "00000000"; o_green <= "00000000"; o_blue <=
"11111111";

            --LATAR
            ELSE
            o_red <= r_grid; o_green <= g_grid; o_blue <= b_grid;
            END IF;

        --FINISH
        ELSIF z = "11111" THEN
            IF (i_pixel_row >= 140 AND i_pixel_row <= 340) AND
            --Y
            ((i_pixel_column >= 50 AND i_pixel_column <= 70 AND
i_pixel_row <= 240)
            OR (i pixel_column >= 70 AND i_pixel_column <= 90 AND
i pixel row >= 240)
            OR (i pixel_column >= 90 AND i_pixel_column <= 110 AND
i pixel row <= 240)

            --O
            OR (i_pixel_column >= 130 AND i_pixel_column <= 150)
            OR (i pixel_column >= 150 AND i_pixel_column <= 170 AND
i pixel row <= 160)
            OR (i pixel_column >= 150 AND i_pixel_column <= 170 AND
i pixel row >= 320)
            OR (i_pixel_column >= 170 AND i_pixel_column <= 190)

            --U
            OR (i_pixel_column >= 210 AND i_pixel_column <= 230)
            OR (i pixel_column >= 250 AND i_pixel_column <= 270)
            OR (i pixel_column >= 230 AND i_pixel_column <= 250 AND
i pixel row >= 320)

            --W
            OR (i pixel column >= 330 AND i pixel column <= 350)
            OR (i_pixel_column >= 370 AND i_pixel_column <= 390 AND
i_pixel_row >= 220)
            OR (i_pixel_column >= 410 AND i_pixel_column <= 430)
            OR (i_pixel_column >= 350 AND i_pixel_column <= 410 AND
i_pixel_row >= 320)

            --I
            OR (i_pixel_column >= 450 AND i_pixel_column <= 470)

            --N
            OR (i_pixel_column >= 490 AND i_pixel_column <= 510)
            OR (i_pixel_column >= 510 AND i_pixel_column <= 530 AND
i_pixel_row >= 160 AND i_pixel_row <= 200)
            OR (i_pixel_column >= 530 AND i_pixel_column <= 550 AND
i pixel row >= 180 AND i pixel row <= 300)
            OR (i pixel column >= 550 AND i_pixel_column <= 570 AND
i pixel row >= 280 AND i pixel row <= 320)
            OR (i_pixel_column >= 570 AND i_pixel_column <= 590))
            THEN
                o_red <= "00000000"; o_green <="00011000"; o_blue <=
"00010000";
            ELSE
                o_red <= "11111111"; o_green <="11111111"; o_blue <=
"11111111";
            END IF;
        --FAIL
        ELSIF z = "00100" THEN
            IF (((i_pixel_row >= 140 AND i_pixel_row <= 340) AND
            --N
            ((i_pixel_column >= 60 AND i_pixel_column <= 80)
            OR (i_pixel_column >= 80 AND i_pixel_column <= 100 AND
i pixel row <= 180)
            OR (i pixel_column >= 100 AND i_pixel_column <= 120 AND
i pixel row >= 180 AND i pixel row <= 240)
            OR (i pixel column >= 120 AND i pixel_column <= 140 AND
i pixel row >= 240 AND i pixel row <= 300)
            OR (i pixel_column >= 140 AND i_pixel_column <= 160 AND
i_pixel_row >= 300)
            OR (i_pixel_column >= 160 AND i_pixel_column <= 180)

            --O
            OR (i pixel column >= 200 AND i pixel column <= 220)
            OR (i_pixel_column >= 220 AND i_pixel_column <= 260 AND
i pixel row <= 160)
            OR (i_pixel_column >= 220 AND i_pixel_column <= 260 AND
i_pixel_row >= 320)
            OR (i_pixel_column >= 260 AND i_pixel_column <= 280)

            --O
            OR (i pixel column >= 300 AND i pixel column <= 320)
            OR (i pixel_column >= 320 AND i_pixel_column <= 360 AND
i pixel row <= 160)
            OR (i pixel_column >= 320 AND i_pixel_column <= 360 AND
i_pixel_row >= 320)
            OR (i_pixel_column >= 360 AND i_pixel_column <= 380)

            --B
            OR (i_pixel_column >= 400 AND i_pixel_column <= 420)
            OR (i_pixel_column >= 420 AND i_pixel_column <= 480 AND
i pixel row <= 160)
            OR (i pixel column >= 420 AND i pixel_column <= 480 AND
i pixel row >= 240 AND i pixel row <= 260)
            OR (i pixel_column >= 420 AND i_pixel_column <= 480 AND
i_pixel_row >= 320)
            OR (i_pixel_column >= 480 AND i_pixel_column <= 500 AND
i_pixel_row >= 160 AND i_pixel_row <= 240)
            OR (i_pixel_column >= 480 AND i_pixel_column <= 500 AND
i_pixel_row >= 260 AND i_pixel_row <= 320)

            --!
            OR (i pixel column >= 560 AND i pixel column <= 580 AND
i pixel row <= 300)
            OR (i_pixel_column >= 560 AND i_pixel_column <= 580 AND
i_pixel_row >= 320)))

            --LINES
            OR (i pixel row >= 40 AND i pixel row <= 60)
            OR (i_pixel_row >= 80 AND i_pixel_row <= 100)
```

```
        OR (i_pixel_row >= 380 AND i_pixel_row <= 400)
        OR (i_pixel_row >= 420 AND i_pixel_row <= 440))

        THEN
            o_red <= "11111111"; o_green <="00000000"; o_blue <=
"00000000";
        ELSE
            o_red <= X"10"; o_green <=X"11"; o_blue <= X"11";
        END IF;
    END IF;

END PROCESS;

END behavioral;
```

Gambar 6-5 Kode VHDL color_rom_vhd.vhd

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
USE  IEEE.STD_LOGIC_ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY vga  IS
    PORT(
        i_clk              :  IN   STD_LOGIC;
        i_red              :  IN   STD_LOGIC;
        i_green            :  IN   STD_LOGIC;
        i_blue             :  IN   STD_LOGIC;
        o_red              :  OUT  STD_LOGIC;
        o_green            :  OUT  STD_LOGIC;
        o_blue             :  OUT  STD_LOGIC;
        o_horiz_sync       :  OUT  STD_LOGIC;
        o_vert_sync        :  OUT  STD_LOGIC;
        o_piksel_row        :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
        o_piksel_column     :  OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 ));
END vga;

ARCHITECTURE behavioral OF vga  IS

CONSTANT TH     : INTEGER := 800;
CONSTANT THB1   : INTEGER := 660;
CONSTANT THB2   : INTEGER := 756;
CONSTANT THD    : INTEGER := 640;

CONSTANT TV     : INTEGER := 525;
CONSTANT TVB1   : INTEGER := 494;
CONSTANT TVB2   : INTEGER := 495;
CONSTANT TVD    : INTEGER := 480;

SIGNAL clock_25MHz  :  STD_LOGIC;
SIGNAL horiz_sync   :  STD_LOGIC;
SIGNAL vert_sync    :  STD_LOGIC;
SIGNAL video_on     :  STD_LOGIC;
SIGNAL video_on_v   :  STD_LOGIC;
SIGNAL video_on_h   :  STD_LOGIC;
SIGNAL h_count      :  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
SIGNAL v_count      :  STD_LOGIC_VECTOR( 9 DOWNTO 0 );

BEGIN

video_on      <= video_on_h  AND video_on_v;

o_red         <= i_red AND video_on;
o_green       <= i_green AND video_on;
o_blue        <= i_blue AND video_on;

o_horiz_sync  <= horiz_sync;
o_vert_sync   <= vert_sync;

PROCESS (i_clk)
BEGIN
    IF i_clk'EVENT AND i_clk='1' THEN
        IF (clock_25MHz = '0') THEN
            clock_25MHz <= '1';
        ELSE
            clock_25MHz <= '0';
        END IF;
    END IF;
END PROCESS;

PROCESS
    BEGIN
    WAIT UNTIL(  clock_25MHz'EVENT  ) AND ( clock_25MHz  = '1'  );
    IF ( h_count  = TH-1 ) THEN
        h_count  <= (others=>'0');
    ELSE
        h_count  <= h_count  + 1;
    END IF;

    IF ( h_count  <= THB2-1 ) AND  (h_count  >= THB1-1 ) THEN
        horiz_sync  <= '0';
    ELSE
        horiz_sync  <= '1';
    END IF;

    IF ( v_count  >= TV-1 ) AND ( h_count  >= 699 ) THEN
        v_count  <= (others=>'0');
    ELSE IF ( h_count  = 699 ) THEN
        v_count  <= v_count  + 1;
        END IF;
    END IF;

    IF ( v_count  <= TVB2-1 ) AND ( v_count  >= TVB1-1 ) THEN
        vert_sync  <=  '0';
    ELSE
        vert_sync  <= '1';
    END IF;

    IF  ( h_count  <= THD-1 ) THEN
        video_on_h  <= '1';
        o_piksel_column  <= h_count;
    ELSE
        video_on_h  <= '0';
    END IF;

    IF ( v_count  <= TVD-1 ) THEN
        video_on_v  <= '1';
        o_piksel_row  <= v_count;
    ELSE
        video_on_v  <= '0';
    END  IF;

END  PROCESS;

END behavioral;
```

Gambar 6-6 Kode VHDL vga.vhd

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE  IEEE.STD_LOGIC_ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CLOCKDIV is port(
     CLK: IN std_logic;
     IN_DIV : IN std_logic;
     O_DIVOUT: OUT std_logic);
end CLOCKDIV;

architecture behavioral of CLOCKDIV is
signal DIVOUT : std_logic;
     begin
          PROCESS(CLK, IN_DIV)
               variable count: integer:=0;
               variable div: integer;
          begin
               IF (IN_DIV = '0') THEN
                    div := 200000;
               ELSIF (IN_DIV = '1') THEN
                    div := 100000;
               END IF;

               if CLK'event and CLK='1' then
                    if(count < div) then
                         count := count+1;
                         if(DIVOUT = '0') then
                              DIVOUT <= '0';
                         elsif(DIVOUT = '1') then
                              DIVOUT <= '1';
                         end if;
                    else
                         if(DIVOUT = '0') then
                              DIVOUT <= '1';
                         elsif(DIVOUT = '1') then
                              DIVOUT <= '0';
                         end if;
                    count := 0;
                    end if;

               end if;
          end process;
          O_DIVOUT <= DIVOUT;
end behavioral;
```

Gambar 6-7 Kode VHDL clockdiv.vhd

```vhdl
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY pll_vhd IS
     PORT
     (
          inclk0         : IN STD_LOGIC  := '0';
          c0        : OUT STD_LOGIC ;
          locked         : OUT STD_LOGIC
     );
END pll_vhd;


ARCHITECTURE SYN OF pll_vhd IS

     SIGNAL sub_wire0   : STD_LOGIC_VECTOR (5 DOWNTO 0);
     SIGNAL sub_wire1   : STD_LOGIC ;
     SIGNAL sub_wire2   : STD_LOGIC ;
     SIGNAL sub_wire3   : STD_LOGIC ;
     SIGNAL sub_wire4   : STD_LOGIC_VECTOR (1 DOWNTO 0);
     SIGNAL sub_wire5_bv     : BIT_VECTOR (0 DOWNTO 0);
     SIGNAL sub_wire5   : STD_LOGIC_VECTOR (0 DOWNTO 0);



     COMPONENT altpll
     GENERIC (
          clk0_divide_by     : NATURAL;
          clk0_duty_cycle         : NATURAL;
          clk0_multiply_by        : NATURAL;
          clk0_phase_shift        : STRING;
          compensate_clock        : STRING;
          gate_lock_signal        : STRING;
          inclk0_input_frequency       : NATURAL;
          intended_device_family       : STRING;
          invalid_lock_multiplier      : NATURAL;
          lpm_hint      : STRING;
          lpm_type      : STRING;
          operation_mode      : STRING;
          port_activeclock       : STRING;
          port_areset         : STRING;
          port_clkbad0        : STRING;
          port_clkbad1        : STRING;
          port_clkloss        : STRING;
          port_clkswitch      : STRING;
          port_configupdate       : STRING;
          port_fbin      : STRING;
          port_inclk0         : STRING;
          port_inclk1         : STRING;
          port_locked         : STRING;
          port_pfdena         : STRING;
          port_phasecounterselect         : STRING;
          port_phasedone      : STRING;
          port_phasestep      : STRING;
          port_phaseupdown        : STRING;
          port_pllena         : STRING;
          port_scanaclr       : STRING;
          port_scanclk        : STRING;
          port_scanclkena         : STRING;
          port_scandata       : STRING;
          port_scandataout        : STRING;
          port_scandone       : STRING;
          port_scanread       : STRING;
          port_scanwrite      : STRING;
          port_clk0      : STRING;
          port_clk1      : STRING;
          port_clk2      : STRING;
          port_clk3      : STRING;
          port_clk4      : STRING;
          port_clk5      : STRING;
          port_clkena0        : STRING;
          port_clkena1        : STRING;
          port_clkena2        : STRING;
          port_clkena3        : STRING;
          port_clkena4        : STRING;
          port_clkena5        : STRING;
          port_extclk0        : STRING;
          port_extclk1        : STRING;
          port_extclk2        : STRING;
          port_extclk3        : STRING;
          valid_lock_multiplier        : NATURAL
     );
     PORT (
               inclk     : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
               locked    : OUT STD_LOGIC ;
               clk  : OUT STD_LOGIC_VECTOR (5 DOWNTO 0)
     );
     END COMPONENT;

BEGIN
     sub_wire5_bv(0 DOWNTO 0) <= "0";
     sub_wire5    <= To_stdlogicvector(sub_wire5_bv);
     sub_wire1    <= sub_wire0(0);
     c0    <= sub_wire1;
     locked    <= sub_wire2;
     sub_wire3    <= inclk0;
     sub_wire4    <= sub_wire5(0 DOWNTO 0) & sub_wire3;

     altpll_component : altpll
     GENERIC MAP (
          clk0_divide_by => 2,
          clk0_duty_cycle => 50,
          clk0_multiply_by => 1,
          clk0_phase_shift => "0",
          compensate_clock => "CLK0",
          gate_lock_signal => "NO",
          inclk0_input_frequency => 20000,
          intended_device_family => "Cyclone II",
          invalid_lock_multiplier => 5,
          lpm_hint => "CBX_MODULE_PREFIX=pll_vhd",
          lpm_type => "altpll",
          operation_mode => "NORMAL",
          port_activeclock => "PORT_UNUSED",
          port_areset => "PORT_UNUSED",
```

```
        port clkbad0 => "PORT UNUSED",
        port clkbad1 => "PORT UNUSED",
        port_clkloss => "PORT_UNUSED",
        port_clkswitch => "PORT_UNUSED",
        port configupdate => "PORT UNUSED",
        port fbin => "PORT UNUSED",
        port inclk0 => "PORT USED",
        port inclk1 => "PORT UNUSED",
        port locked => "PORT USED",
        port_pfdena => "PORT_UNUSED",
        port_phasecounterselect => "PORT_UNUSED",
        port_phasedone => "PORT_UNUSED",
        port_phasestep => "PORT_UNUSED",
        port_phaseupdown => "PORT_UNUSED",
        port pllena => "PORT UNUSED",
        port scanaclr => "PORT UNUSED",
        port scanclk => "PORT UNUSED",
        port scanclkena => "PORT UNUSED",
        port_scandata => "PORT_UNUSED",
        port_scandataout => "PORT_UNUSED",
        port_scandone => "PORT_UNUSED",
        port_scanread => "PORT_UNUSED",
        port_scanwrite => "PORT_UNUSED",
        port_clk0 => "PORT_USED",
        port clk1 => "PORT UNUSED",
        port clk2 => "PORT UNUSED",
        port clk3 => "PORT UNUSED",
        port clk4 => "PORT UNUSED",
        port_clk5 => "PORT_UNUSED",
        port_clkena0 => "PORT_UNUSED",
        port_clkena1 => "PORT_UNUSED",
        port clkena2 => "PORT UNUSED",
        port clkena3 => "PORT UNUSED",
        port clkena4 => "PORT UNUSED",
        port clkena5 => "PORT UNUSED",
        port extclk0 => "PORT UNUSED",
        port extclk1 => "PORT UNUSED",
        port_extclk2 => "PORT_UNUSED",
        port_extclk3 => "PORT_UNUSED",
        valid_lock_multiplier => 1
    )
    PORT MAP (
        inclk => sub wire4,
        clk => sub wire0,
        locked => sub_wire2
    );

END SYN;
```

Gambar 6-8 Kode VHDL pll.vhd