BAB III
METODE PENELITIAN

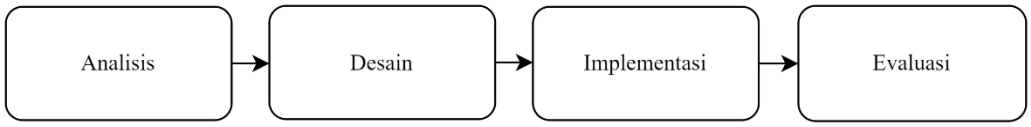
3.1 Objek Penelitian

Objek fokus pada penelitian ini adalah perancangan dan implementasi permainan Pong pada FPGA Cyclone IV dengan menggunakan *board* Daxigua Logic'3. Penelitian ini bertujuan untuk mengembangkan dan menguji implementasi permainan Pong, dengan mengeksplorasi kinerja sumber daya logika yang tersedia serta menganalisis konsumsi daya.

Permainan Pong yang diimplementasikan terdiri dari dua *paddle* dan satu bola yang dipantulkan, di mana pemain dapat mengontrol *paddle* dengan pergerakan horizontal menggunakan *push-button* untuk memantulkan bola dan mencetak skor. Penelitian ini juga berfokus pada pengolahan sinyal VGA untuk menghasilkan tampilan visual berupa elemen-elemen permainan secara *real-time* dengan resolusi 640x480 @ 60Hz. Elemen-elemen permainan tersebut adalah *paddle*, bola, dan skor, yang dirancang untuk ditampilkan pada sebuah *display* VGA dengan posisi yang telah diatur dan pergerakan yang sesuai dengan interaksi pemain. VHDL akan digunakan sebagai bahasa pemrograman *board* yang mendeskripsikan setiap elemen bergerak secara konsisten dengan logika permainan Pong, sehingga menghasilkan pengalaman bermain yang menyerupai mekanisme permainan aslinya.

3.2 Metode Penelitian

Penelitian ini mengimplementasikan metode Design and Development Research (DDR) untuk mengembangkan permainan Pong *multiplayer* berbasis FPGA sebagai alat yang dapat digunakan sebagai sarana eksplorasi. Sebagaimana dijelaskan oleh Richey dkk. (2004), DDR berfokus pada analisis, desain, pengembangan, implementasi, dan evaluasi, tahap-tahap ini digambarkan pada Gambar 3.1.



Gambar 3.1 Diagram Metode DDR

Dalam konteks penelitian ini, penelitian bertujuan untuk memberikan gambaran tentang bagaimana rancangan dan implementasi permainan Pong dapat dilakukan pada FPGA Cyclone IV, serta menunjukkan kinerja penggunaan sumber daya dan konsumsi daya *board* FPGA yang digunakan.

Tahapan penelitian ini mencakup beberapa langkah utama, yaitu:

1. Analisis. Melibatkan identifikasi kebutuhan fungsional dan non-fungsional untuk permainan Pong pada FPGA.
2. Desain. Mencakup perancangan logika dan tampilan permainan Pong, *flowchart,* struktur sistem, hingga rangkaian *pin* dan *wiring*.
3. Implementasi sistem. Setelah perancangan, desain akan diimplementasikan pada FPGA Cyclone IV menggunakan perangkat lunak Quartus Prime. Bagian ini mencakup langkah-langkah yang diperlukan untuk mengimplementasikan desain ke dalam *board* FPGA.
4. Evaluasi sistem. Hasil pengujian dianalisis untuk melihat apakah implementasi memenuhi kriteria yang telah ditentukan, seperti stabilitas logika permainan, simulasi ModelSim yang sesuai, hingga jumlah penggunaan sumber daya dan konsumsi daya FPGA.

3.3 Perancangan Sistem

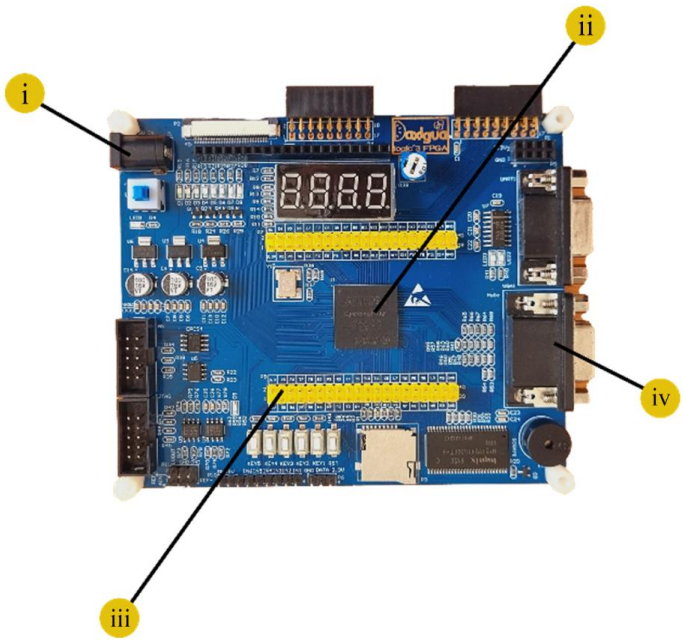
3.3.1 Analisis Kebutuhan

FPGA adalah platform yang digunakan untuk memprogram dan mengonfigurasi sirkuit digital untuk menjalankan permainan Pong secara langsung di *hardware*. *Board* yang digunakan adalah Daxigua logic'3 FPGA Cyclone IV. *Board* Daxigua Logic'3 memiliki jaringan logika Cyclone IV EP4CE6F17C8N. Tipe jaringan ini dilengkapi dengan 392 LABs/CLBs (*Logic Array Blocks/Configurable Logic Blocks*) yang memiliki total 6.272 *logic elements*. Untuk penyimpanan sementara, FPGA ini memiliki total 276.480 bit RAM. Selain

itu, FPGA ini memiliki 30 elemen *embedded multiplier* 9-bit dan 2 PLLs (*Phase-Locked Loops*).

Masing-masing sumber daya pada FPGA ini memiliki peran spesifik yang bervariasi dalam implementasi. *Logic Elements*, sebagai unit dasar untuk mengimplementasikan fungsi logika digital seperti gerbang logika dan register, merupakan bagian utama dalam perancangan permainan Pong. Adapun sumber daya lain seperti *memory bits* (untuk penyimpanan data) dan *Embedded Multiplier* (untuk operasi perkalian yang efisien) namun tidak dimanfaatkan secara signifikan dalam implementasi ini. Sementara itu, PLLs (*Phase-Locked Loops*), yang berfungsi untuk menghasilkan frekuensi *clock* yang presisi, juga tidak digunakan. Sinyal *clock* yang dibutuhkan dalam penelitian ini diperoleh melalui pembagian frekuensi langsung dari *clock* utama *board*.

Board FPGA beserta komponen-komponen utamanya yang dimanfaatkan dalam implementasi dapat dilihat pada Gambar 3.2.



Gambar 3.2 *Board* FPGA Daxigua logic'3

Penggunaan komponen ini mencakup VGA *interface* (untuk menampilkan permainan), *FPGA Core*, Port daya, serta GPIO *yellow header* yang mendukung koneksi perangkat eksternal seperti *push-button*. Berikut adalah penjelasan bagian-bagian yang relevan, disesuaikan dengan penomoran dalam Gambar 3.2.

i. Port daya menyediakan daya utama untuk mengoperasikan seluruh komponen pada *board* FPGA, termasuk logika internal dan komponen eksternal yang terhubung.
ii. *FPGA Core* adalah bagian utama dari *board* yang bertugas menjalankan semua logika permainan Pong seperti logika gerakan bola, deteksi tabrakan, dan perhitungan skor hingga sinkronisasi sinyal VGA akan diprogram di bagian *FPGA Core*.
iii. GPIO *yellow header* adalah jalur *pin input*/*output* digital pada FPGA yang digunakan untuk menghubungkan komponen eksternal seperti *push-button* di sebuah PCB untuk menggerakkan *paddle* (naik/turun) setiap pemain, start dan reset permainan.
iv. VGA *Interface* pada Cyclone IV digunakan untuk menampilkan permainan Pong pada sebuah *display* VGA. Bagian ini berfungsi mengirimkan sinyal video yang telah diolah oleh FPGA, seperti posisi *paddle*, gerakan bola, teks dan skor pemain.

Selain *board* FPGA, berikut beberapa kebutuhan yang diperlukan untuk mengembangkan penelitian ini.

1. Kode VHDL merupakan sebuah *Hardware Description Language* (HDL) yang dirancang untuk mendeskripsikan dan menyimulasikan perilaku logika digital. VHDL memungkinkan pemrograman setiap komponen permainan, seperti pergerakan bola, interaksi dengan *paddle*, deteksi tabrakan, dan perhitungan skor, tampilan teks, sebagai blok logika yang terstruktur. Selain itu, sifat VHDL yang sangat deskriptif mempermudah penambahan fitur atau modifikasi desain di masa mendatang.
2. PC atau Laptop dengan perangkat lunak Quartus Prime yang digunakan sebagai platform untuk menjalankan perangkat lunak tersebut guna menulis kode VHDL, sintesis desain, dan konfigurasi FPGA.
3. ModelSim, sebuah perangkat lunak yang digunakan untuk menguji dan memverifikasi kode VHDL sebelum diimplementasikan pada FPGA. Dengan fitur simulasi, ModelSim memungkinkan di fase pengembangan untuk menjalankan simulasi logika permainan secara *virtual* dan menganalisis

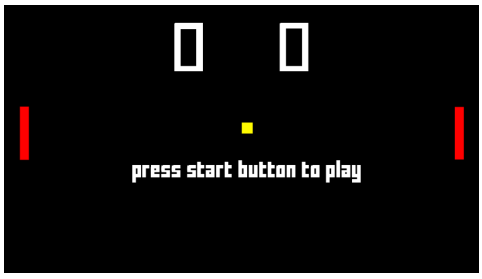
hasilnya dalam bentuk *waveform*. Ini memastikan bahwa kode yang ditulis berfungsi sesuai rancangan, tanpa harus langsung mengonfigurasi FPGA.

4. USB Blaster, sebuah alat yang digunakan untuk mengonfigurasi FPGA, dihubungkan dengan *port* JTAG pada *board* FPGA untuk mengunggah *file* konfigurasi yang dihasilkan dari perangkat lunak Quartus Prime ke FPGA. *FPGA Core* akan terprogram dengan kode VHDL yang sudah disintesis sehingga logika permainan dapat dijalankan.
5. *Display* dengan Port VGA, sebuah perangkat yang digunakan untuk menampilkan tampilan visual permainan Pong yang dihasilkan oleh FPGA.

3.3.2 Rancangan Permainan

Permainan Pong memiliki tampilan yang ikonik. Meskipun ada beberapa iterasi berbeda dari permainan Pong yang telah dibuat sepanjang sejarahnya, Pong memiliki tampilan yang sangat mudah dikenali dengan sifatnya yang sederhana, dan berwarna hitam putih. Namun, dalam penelitian ini, akan digunakan warna merah dan kuning untuk memberikan gambaran akan kemampuan pengelolaan warna sinyal VGA dalam FPGA.

Permainan terdiri dari dua *paddle* di bagian kanan dan kiri, *paddle* kiri adalah pemain 1, sedangkan *paddle* kanan adalah pemain 2. Selain itu terdapat bola di tengah bagian display dan tampilan skor pada bagian tengah atas *display.* Tampilan awal yang terjadi disebut tampilan *idle,* karena objek permainan masih dalam posisi semula. Tampilan ini dapat dilihat di Gambar 3.3.



Gambar 3.3 Rancangan Tampilan *Idle* Awal Permainan

Gambar 3.3 menunjukkan dua *paddle* di bagian kanan dan kiri, yang merepresentasikan pemain dalam Pong, dan bola pada tengah layar. Permainan akan memiliki tampilan skor di bagian tengah atas *display*. Selain itu, sebelum

permainan dimulai, permainan akan menampilkan teks ‘PRESS START BUTTON TO PLAY’, Permainan tidak akan dimulai sebelum pemain menekan tombol *start.*

Selanjutnya, setelah pemain memulai permainan dengan menekan tombol *start*, permainan masuk ke dalam sebuah *loop* berkelanjutan. Pemain bertugas menggerakkan *paddle* untuk mencegah bola melewati gawangnya. Tampilan ini dapat dilihat pada Gambar 3.4.

Gambar 3.4 Rancangan Tampilan Permainan

Gambar 3.4 menunjukkan rancangan tampilan permainan yang sedang berlangsung. Bola memiliki logika pantulan terhadap *paddle* dan batas layar, Setiap kali bola berhasil melewati *paddle* lawan, skor akan bertambah satu poin.

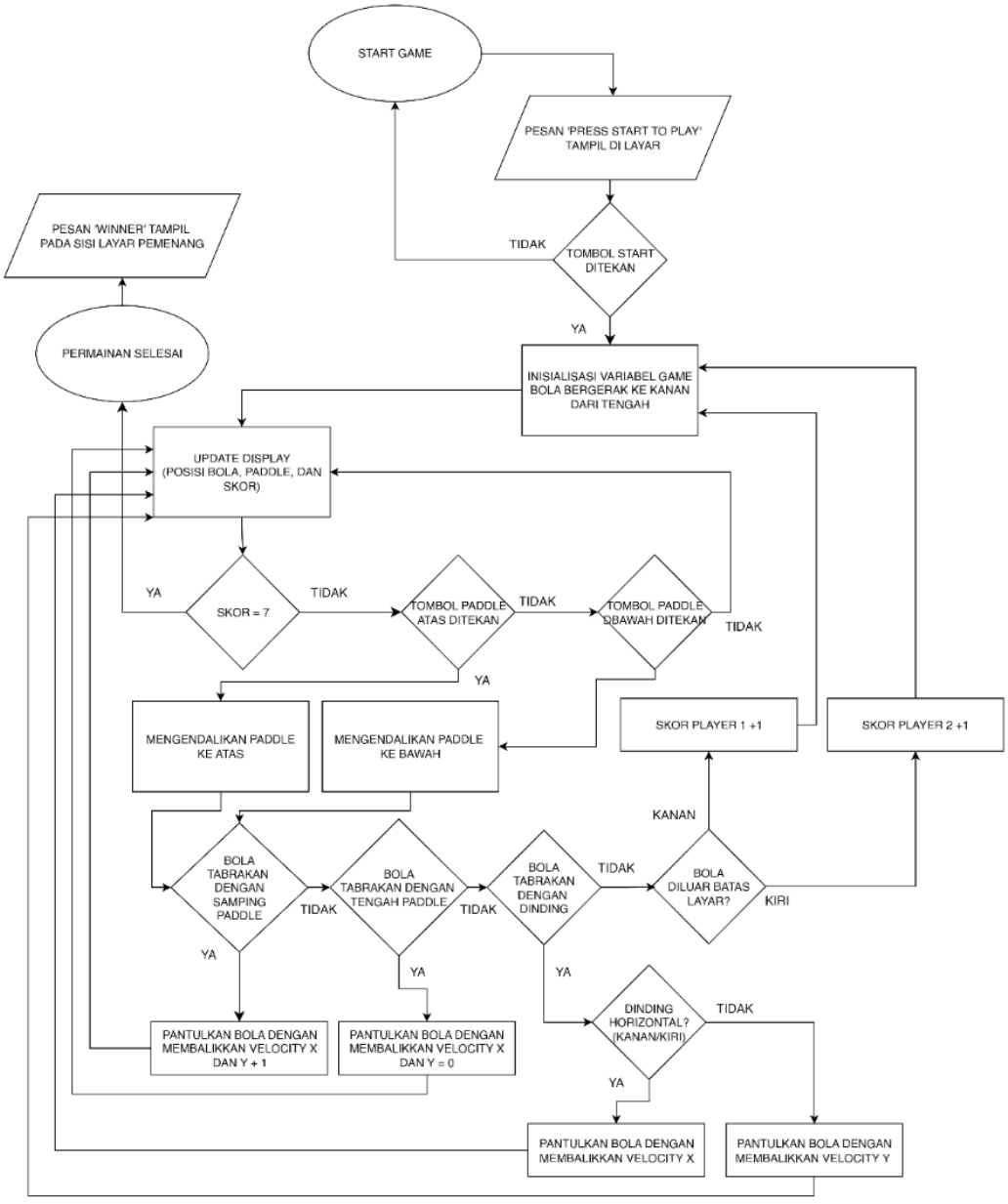
Setelah salah satu pemain mencapai skor 7, permainan akan berpindah ke tampilan *game over,* atau bagian saat permainan telah berakhir. Rancangan tampilan ini dapat dilihat pada Gambar 3.5.

Gambar 3.5 Rancangan Tampilan Permainan Pemenang

Gambar 3.5 merupakan rancangan tampilan akhir permainan dengan menampilkan teks ‘WINNER’ pada *display* yang muncul pada sisi *paddle* pemenang.

3.3.3 *Flowchart* Permainan

Logika kerja permainan digambarkan menggunakan *flowchart,* melalui setiap tahapan, mulai dari permulaan hingga berakhirnya permainan saat salah satu pemain meraih skor 7. *Flowchart* dapat dilihat pada Gambar 3.6.



Gambar 3.6 *Flowchart* Permainan

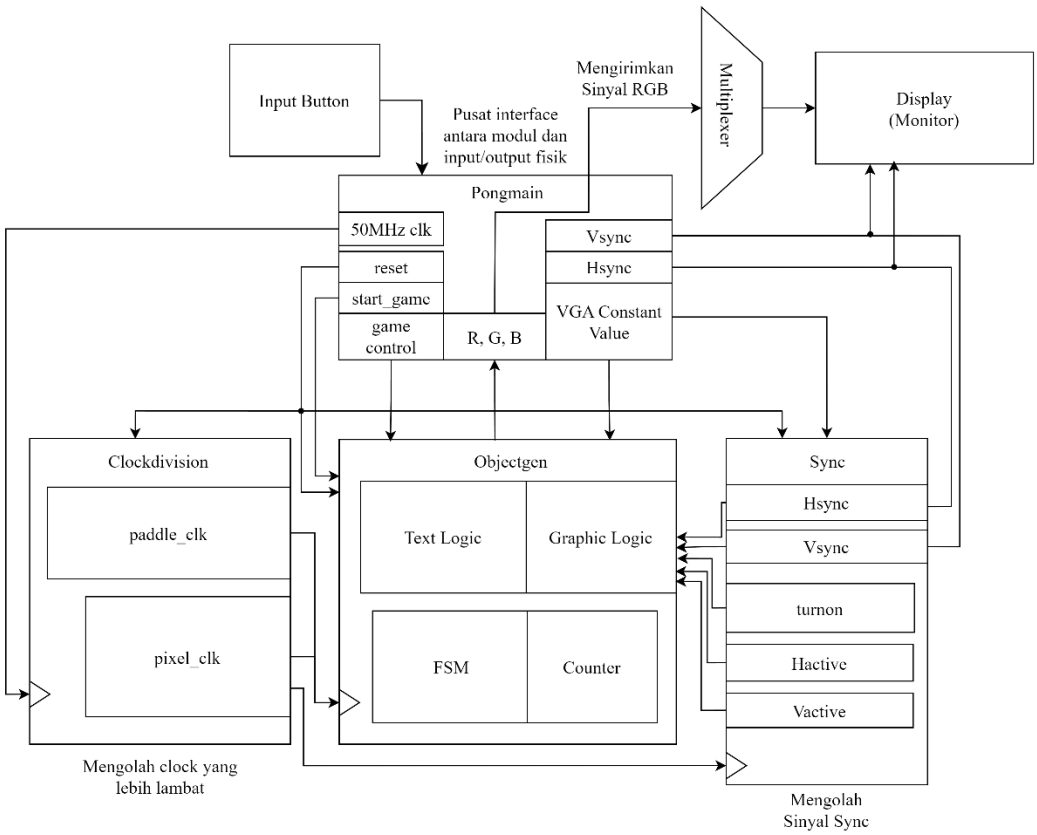
Flowchart pada Gambar 3.6 menguraikan tahapan permainan Pong pada sistem, setiap tahap memiliki kondisi logis yang harus dipenuhi sebelum dapat berlanjut. Berikut penjelasannya.

1. Permainan dimulai ketika tombol start ditekan dan variabel permainan akan diinisiasi. Berikut adalah variabel yang diinisiasi.
    a. Posisi bola diatur ke tengah *display*.
    b. Kecepatan bola diberi nilai awal (bola bergerak ke kanan).
2. Setelah permainan dimulai, permainan akan memasuki sebuah *loop*, yang terus berjalan hingga kondisi skor salah satu permain meraih angka 7. Berikut adalah hal-hal dalam *loop* yang terjadi.
    a. *Update Display*. Sistem memperbarui tampilan bola, *paddle*, dan skor pada layar sesuai data terkini.
    b. Deteksi Tabrakan (*Collision*). Dengan membandingkan koordinat bola dengan koordinat *paddle* dan batas atas/bawah *display*. Berikut adalah beberapa tipe tabrakan.
        i) Tabrakan dengan samping paddle, bola dipantulkan dengan membalikkan *velocity* x dan menambahkan percepatan vertikal (*velocity* Y ditambah).
        ii) Tabrakan dengan tengah paddle, bola dipantulkan lurus dengan membalikkan *velocity* x (hanya balik arah X).
        iii) Tabrakan dengan dinding atas/bawah, arah vertikal bola dibalik.
    c. Memeriksa *Out-of-Boundaries*, apabila bola keluar dari batas kiri atau kanan. Berikut adalah logika *Out-of-Boundaries.*
        i) Jika bola melewati batas kiri, pemain 2 akan mencetak 1 poin.
        ii) Jika bola melewati batas kanan, pemain 1 akan mencetak 1 poin.

        Setelah salah satu pemain mencetak skor, posisi dan arah bola akan reset kembali ke tengah layar, dengan kecepatan bola awal (bola bergerak ke kanan.)
3. Ketika salah satu pemain telah mencapai angka 7, permainan dihentikan dan sistem akan menampilkan pesan ‘WINNER’ di sisi pemenang.

3.3.4 Struktur Sistem Pong

Sebuah diagram digunakan untuk merepresentasikan struktur sistem Pong untuk menjelaskan bagaimana logika permainan Pong diimplementasikan pada FPGA. Struktur sistem Pong dapat dilihat pada Gambar 3.7.



Gambar 3.7 Struktur Sistem Pong

Setiap modul pada Gambar 3.7 akan dianalisis mengenai bagaimana berbagai komponen seperti *Pongmain, Clockdivision, Sync,* dan *Objectgen* bekerja sama untuk mengontrol alur permainan dan menghasilkan tampilan visual. Struktur diagram ini mencerminkan *output RTL* dari perangkat lunak Quartus Prime, sekaligus secara eksplisit merepresentasikan *Pongmain* sebagai *top-level entity* yang bertanggung jawab terhadap integrasi keseluruhan sistem. *Output RTL* dapat dilihat pada Lampiran 7.

3.3.4.1 *Input* Button

Input berupa *push-button* diimplementasikan menggunakan PCB dengan *jump wires. push-button* yang digunakan merupakan saklar mekanis yang dapat mengalami *bouncing* saat ditekan atau dilepas, menyebabkan sinyal berubah cepat sebelum stabil. Untuk mencegah deteksi *input* yang salah, digunakan metode *debouncing. Debouncing* dilakukan secara *hardware*, dengan menggunakan resistor 10k Ohm, yang dapat membantu mengurangi efek *bouncing.*

3.3.4.2 Modul *Pongmain*

Modul *Pongmain* adalah pusat kendali utama dari keseluruhan sistem permainan Pong ini. *Pongmain* bertugas sebagai jembatan antara *input*/*output* eksternal/fisik dan submodul-modul penting lainnya seperti *Clockdivision*, *Objectgen*, dan *Sync*. Modul ini menjadi jembatan *input/output* dengan mendeklarasikan *port* untuk sinyal *clock* (*clk*) utama sebesar 50Mhz, tombol *reset* (*reset*), tombol kontrol permainan (*gamecontrol*), serta tombol *start_game*. *Output* yang disalurkan melalui modul ini berupa sinyal RGB (R, G, B) yang dihasilkan modul *Objectgen*, serta sinyal sinkronisasi (*Hsync* dan *Vsync*) untuk keperluan tampilan oleh modul *Sync*.

Pongmain memiliki *Package* adalah sebuah blok kode yang berisi kumpulan nilai tetap untuk digunakan secara global dalam sistem guna menghasilkan tampilan VGA. Implementasi *package* dapat dilihat pada Gambar 3.8.

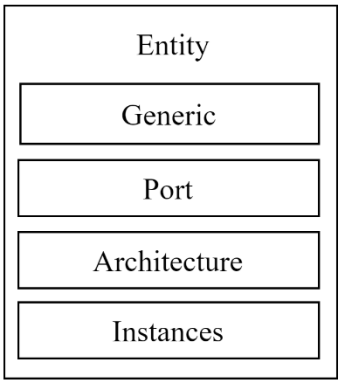
```
package VGA_constants is --VGA variables package (reusable)

    constant Hsyncpulse: integer := 96;
    constant Hstartact: integer := 144;
    constant Hendact: integer := 784;
    constant Hreset: integer := 800;
    constant Vsyncpulse: integer := 2;
    constant Vstartact: integer := 35;
    constant Vendact: integer := 515;
    constant Vreset: integer := 525;

end package VGA_constants;
```

Gambar 3.8 Cuplikan Kode *Package*

Package ini bersifat *reusable*, dapat digunakan di banyak modul lain tanpa harus mendefinisikan ulang di dalamnya. *Package* berisi beberapa nilai penting yang menentukan *timing* dan posisi aktif dari sinyal horizontal dan vertikal dalam

sistem tampilan VGA. *Hsyncpulse* mengatur rentang sinkronisasi horizontal, *Hstartact* dan *Hendact* menentukan rentang kolom *pixel* yang akan terlihat aktif di *display* serta *Hreset yang* bertanggung jawab sebagai *back porch.* Begitu pula dengan parameter vertikal seperti *Vsyncpulse, Vstartact, Vendact,* dan *Vreset*.

Setelah mendefinisikan *package*, modul akan mendeklarasi *entity* Pongmain. Di dalamnya terdapat empat bagian utama, seperti yang ditunjukkan pada Gambar 3.9 yaitu *generic, port, architecture,* dan instansi modul.



Gambar 3.9 Struktur *Entity* Pongmain

Bagian *generic* mendefinisikan parameter yang bisa diubah, seperti nilai pembagi *clock* agar kecepatan *clock* utama sebesar 50MHz dapat disesuaikan menjadi kecepatan yang dibutuhkan oleh *pixel clock* untuk mengeluarkan sinyal VGA dengan resolusi 640x480 @ 60Hz, serta *clock* yang digunakan untuk menyesuaikan kecepatan gerakan *paddle*.

Sementara itu, bagian *port* adalah tempat mendefinisikan sinyal-sinyal *input* dan *output* fisik dengan rangkaian digital yang dibuat. Beberapa sinyal tersebut adalah *clk, reset, gamecontrol, start_game, Hsync,* dan *Vsync. Clk* digunakan sebagai sinyal *clock input* dari osilator, *reset* digunakan untuk *input push-button* dalam mengatur ulang sistem, *gamecontrol* untuk mengendalikan permainan, dan *start_game* untuk memulai permainan. Sinyal *Hsync* dan *Vsync* sebagai sinyal sinkronisasi, sekaligus R, G, dan B sebagai *output* warna adalah sinyal yang digunakan untuk menghasilkan visual pada *display*.

Selanjutnya pada bagian *architecture,* berisi *component,* merupakan definisi dari blok utama dalam sistem, semacam *blueprint* yang menunjukkan bagaimana

modul *Pongmain* berinteraksi dengan modul lainnya, serta perangkat keras melalui sinyal *input* dan *output.*

Pongmain digunakan untuk inisiasi empat instansi utama, *Sync*, *Objectgen*, dan dua buah *Clockdivision* dengan fungsi berbeda.

Modul *Sync* bertanggung jawab untuk menghasilkan sinyal sinkronisasi VGA serta menentukan area aktif dari tampilan. *Port* yang digunakan seperti *Hsync*, *Vsync*, dihasilkan dari modul ini. *Output turnon* juga dihasilkan di sini, yang digunakan sebagai sinyal kendali untuk menyalakan tampilan ketika posisi *pixel* berada di area aktif oleh *Objectgen*.

Selanjutnya, modul *Objectgen* adalah bagian inti dari sistem yang mengatur objek dalam permainan seperti *paddle* dan bola. Modul ini menerima *input* dari *clock pixel* (*pixel_clk*), *clock* khusus *paddle* (*paddle_clk*), dan sinyal kontrol seperti *gamecontrol* serta *start_game*. Berdasarkan posisi aktif *display* (*turnon*) dan sinyal *Hsync*, serta *Vsync*, modul ini akan menghasilkan sinyal warna R, G, dan B yang akan ditampilan pada *display* sesuai dengan logika permainan.

Lalu ada dua instansi *Clockdivision*, Fungsinya adalah untuk memperlambat sinyal *clock* utama (*clk*) ke tingkat yang sesuai. Instansi pertama menghasilkan *pixel_clk* yang digunakan oleh sistem VGA, sementara instansi kedua menghasilkan *paddle_clk* yang lebih lambat untuk mengatur kecepatan *paddle* agar *paddle* tidak melakukan pergerakan *pixel* dalam kecepatan asli *board*.

Secara keseluruhan, bagian arsitektur ini menghubungkan semua modul utama menjadi satu sistem yang saling terkoneksi, dari pembentukan sinyal VGA, pemrosesan objek dalam permainan, hingga pembagian *clock* untuk memastikan semuanya berjalan dengan *timing* yang tepat.

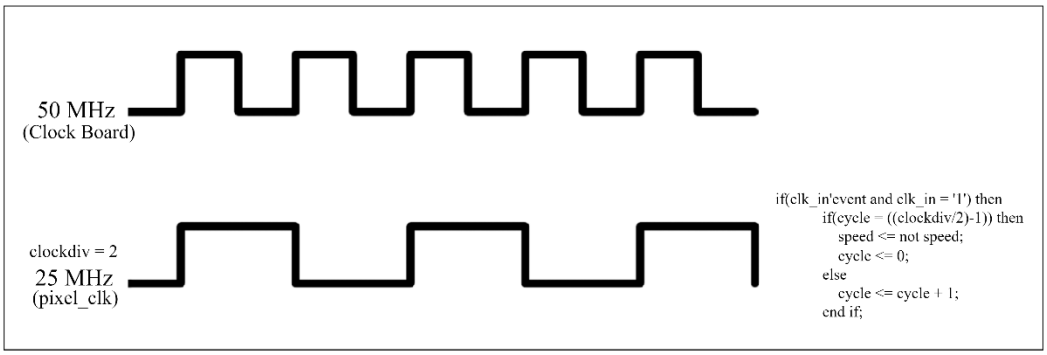
3.3.4.3 Modul *Clockdivision*

Melalui inisiasi modul *Clockdivision* dari modul *Pongmain, clockdivision* digunakan untuk memperlambat frekuensi *clock* utama (*clk*) menjadi dua jenis *clock* berbeda:

1. *pixel_clk*, digunakan untuk mengatur kecepatan *rendering pixel*, untuk resolusi 640x480 @ 60Hz, dibutuhkan *pixel clock* sebesar 25MHz. Oleh karena itu, *clock* utama FPGA sebesar 50MHz dibagi dengan 2.

2. *paddle_clk*, digunakan untuk mengatur kecepatan gerakan *paddle*. *clock* 50MHz *board* FPGA dibagi dengan 415000. Cukup untuk memperlambat pergerakan *paddle* dan bola agar dapat dilihat oleh kasat mata namun tidak memerlukan nilai yang spesifik.

Modul ini sangat penting karena kecepatan *clock* asli terlalu tinggi untuk kebutuhan grafis dan logika permainan, sehingga perlu dibagi menjadi kecepatan yang sesuai. Gambar 3.10 menunjukkan gambaran osilasi dari sebuah *clock* yang sudah diperlambat menjadi 25MHz.



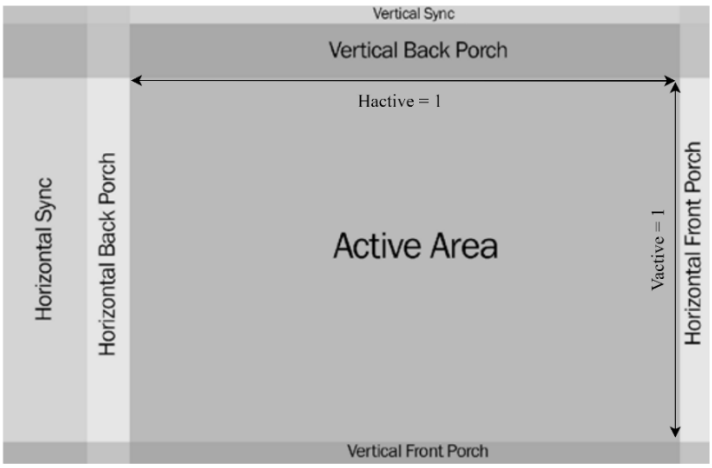
Gambar 3.10 Pembagian *Clock*

Dalam Gambar 3.10, terdapat dua sinyal internal, *cycle* dan *speed*. Sinyal *cycle* digunakan untuk menghitung jumlah siklus dari sinyal *clock input* (*clk_in*), sedangkan *speed* akan menjadi sinyal *clock output* (*clk_out)* yang telah dibagi. Proses akan terus menghitung *cycle* hingga mencapai nilai maksimum yang oleh (*(clockdiv/2)-1*) (*clockdiv* adalah jumlah pembagi). Ketika nilai ini tercapai, sinyal *speed* akan dibalik (*toggle*), dan *cycle reset* ke nol. Dengan cara ini, sinyal *speed* akan berosilasi dengan frekuensi yang lebih rendah dibandingkan *clk_in* karena proses ini memerlukan *clock* utama untuk naik menjadi '1' sebelum mengeksekusi logika.

3.3.4.4 Modul *Sync*

Modul *Sync* bertanggung jawab menghasilkan sinyal sinkronisasi VGA (*Hsync*, dan *Vsync*) yang dibutuhkan oleh monitor untuk menampilkan gambar. Proses ini dikendalikan oleh sinyal *pixel_clk*, yang merupakan *clock* hasil olahan modul *Clockdivision*.

Terdapat area aktif dalam sebuah *display,* area aktif dibagi menjadi dua, horizontal (*Hactive)* dan vertikal (*Vactive)*. *Hactive* dan *Vactive* dapat dilihat pada Gambar 3.11. Variabel *Hactive* dan *Vactive* dibutuhkan oleh variabel *turnon.*



Gambar 3.11 Area Aktif Display

Variabel penghitung horizontal (*Hcount*) digunakan untuk melacak posisi *pixel* secara horizontal, dari awal hingga akhir satu garis. Saat penghitung mencapai nilai maksimum (*Hreset*), maka *Hsync* kembali ke nol (*active low*), menandai dimulainya garis baru. Berdasarkan nilai penghitung ini, sistem menentukan kapan sinyal sinkronisasi Horizontal (*Hsync)* aktif, serta kapan area aktif horizontal (*Hactive*) dimulai dan berakhir.

Selanjutnya, menggunakan prinsip yang sama seperti horizontal, variabel penghitung vertikal (*Vcount)* digunakan untuk melacak posisi *pixel* secara vertikal, yang mengatur sinyal *Vsync* kembali ke nol (*active low*) ketika mencapai nilai maksimum (*Vreset*). Berdasarkan nilai penghitung ini, sistem menentukan kapan sinyal sinkronisasi vertikal (*Vsync*) aktif, serta kapan area aktif vertikal (*Vactive*) dimulai dan berakhir. Nilai *Vcount* akan diperbarui setiap garis horizontal selesai.

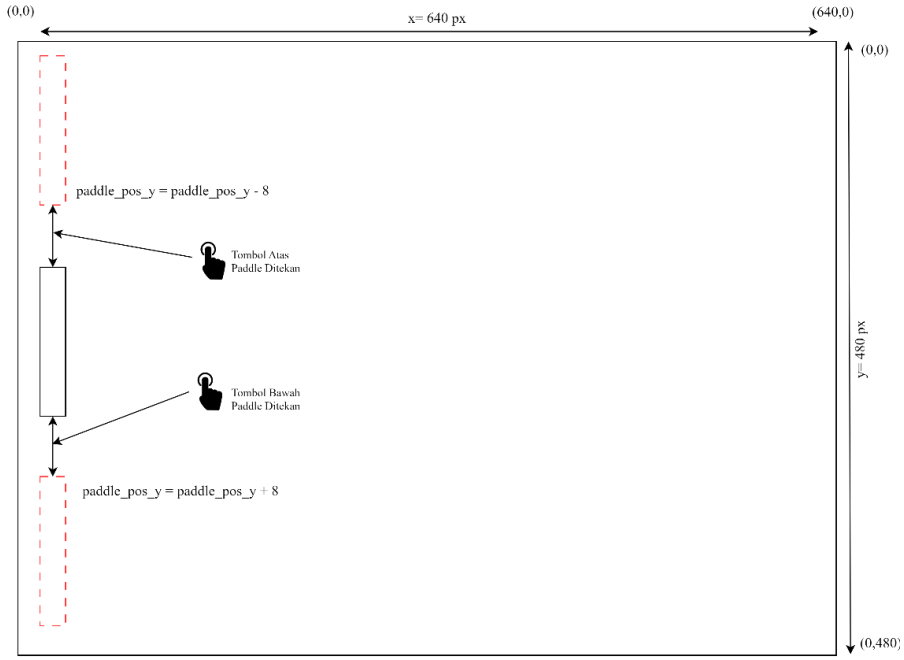
Kedua logika horizontal dan vertikal berkontribusi kepada variabel *turnon* yang akan bernilai aktif hanya ketika *Hactive* dan *Vactive* keduanya aktif, artinya posisi *pixel* berada di dalam area tampilan aktif pada *display*.

3.3.4.5 Modul *Objectgen*

Modul *Objectgen* bertanggung jawab dalam mengatur logika permainan Pong secara keseluruhan, mulai dari tampilan teks statis, objek grafis seperti bola

dan *paddle*, hingga pengaturan transisi antar *state* permainan melalui *Finite state machine* (FSM). Modul ini juga menjaga sinkronisasi antara logika permainan dan sistem tampilan monitor melalui modul *sync* dan kecepatan yang dikontrol oleh *clockdivision*.

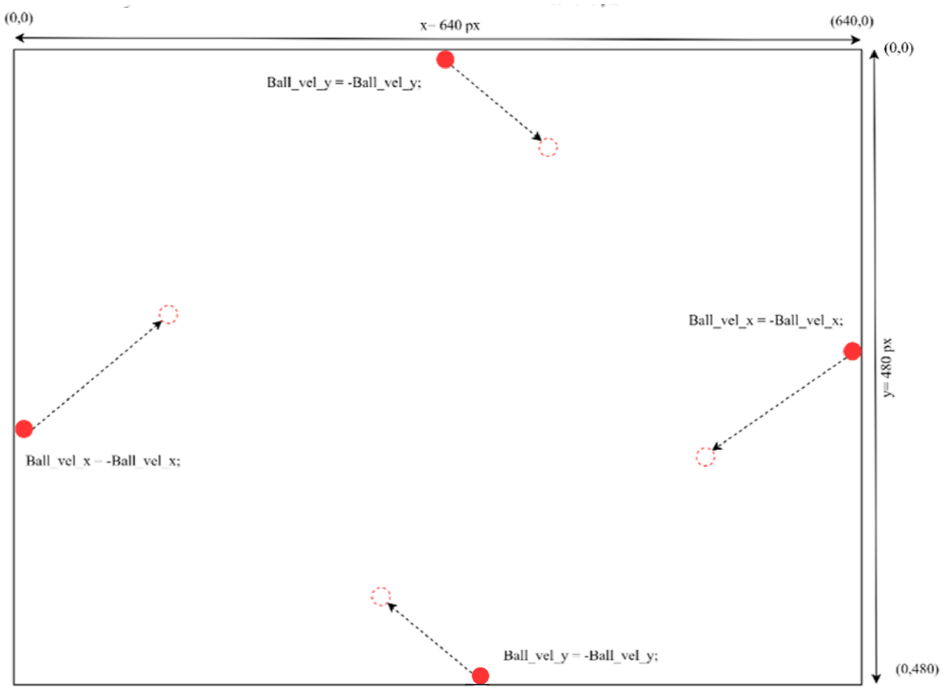
Logika permainan pergerakan *paddle* digambarkan pada Gambar 3.12. Variabel koordinat direpresentasikan menggunakan variabel *paddle_pos_y*.



Gambar 3.12 Logika Pergerakan *Paddle*

Gambar 3.12 menunjukkan *paddle* akan bergerak ketika *input* permainan (*gamecontrol*) ditekan, posisi *paddle* akan bergeser sebesar 8 *pixel* per *clock tick* sesuai dengan *input* yang diberikan menggunakan logika *paddle_pos_y + 8* untuk pergerakan *padddle* ke bawah, atau *paddle_pos_y – 8* untuk pergerakan *paddle* ke atas.

Selain memiliki logika untuk menggerakkan objek grafis seperti *paddle*, terdapat logika *collision detection* antara bola dengan *paddle* dan batas *display*. Ketika bola mencapai tepi atas/bawah atau kiri/kanan *display*, arah bola akan dibalik untuk menyimulasikan pantulan. Arah bola secara horizontal direpresentasikan dengan variabel *Ball_vel_x,* sedangkan secara vertikal dengan variabel *Ball_vel_y.* Logika pantulan dengan dinding digambarkan pada Gambar 3.13.



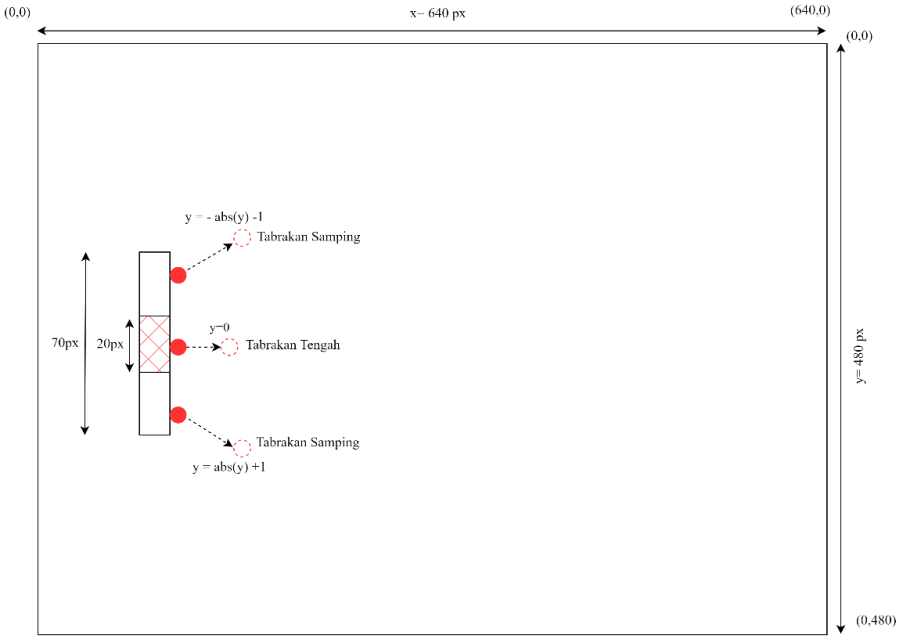
Gambar 3.13 Logika Pantulan Bola Dengan Dinding

Gambar 3.13 menggambarkan logika kolisi yang terjadi jika rentang horizontal atau vertikal bola tumpang tindih dengan rentang batas layar horizontal atau vertikal. Berikut adalah logika kolisi dinding secara horizontal dan vertikal.

1. Deteksi Kolisi Dinding Horizontal (Atas & Bawah). Berikut adalah kondisi dan respons dari logika pantulan horizontal.
    a. Kondisi: Bola diprediksi akan menyentuh atau melampaui batas atas (*next_ball_pos_y <= 0*) atau batas bawah (*next_ball_pos_y >= 480*).
    b. Respons: Kecepatan vertikal bola (*Ball_vel_y*) dibalik (*Ball_vel_y = -Ball_vel_y*), membuat bola memantul ke arah vertikal yang berlawanan.
2. Deteksi Kolisi Dinding Vertikal (Kiri & Kanan). Berikut adalah kondisi dan respons dari logika pantulan vertikal.
    a. Kondisi: Bola diprediksi akan menyentuh atau melampaui batas kiri (*next_ball_pos_x <= 0*) atau batas kanan (*next_ball_pos_x >= 640*).

b. Respons: Kecepatan horizontal bola (*Ball_vel_x*) dibalik (*Ball_vel_x* = *-Ball_vel_x*), membuat bola memantul ke arah horizontal yang berlawanan.

Selanjutnya, logika *collision,* atau tabrakan yang terjadi antara bola dengan *paddle* dapat dilihat pada Gambar 3.14. Logika pantulan dengan *paddle* sedikit berbeda dengan logika pantulan dengan batas *display*.



Gambar 3.14 Logika Pantulan Bola Dengan *Paddle*

Gambar 3.14 menunjukkan tiga kemungkinan tabrakan yang dapat terjadi pada bola, tabrakan tengah, dan dua tabrakan samping. Kolisi terjadi jika rentang horizontal bola tumpang tindih dengan rentang horizontal *paddle*, dan pada saat yang sama rentang vertikal bola tumpang tindih dengan rentang vertikal *paddle*. Berikut adalah penjelasan dari logika yang terjadi saat bola berpantul dengan paddle.

1. Kecepatan horizontal bola (*Ball_vel_x*) akan langsung dibalik (*Ball_vel_x = -Ball_vel_x*;).
2. Pantulan Lurus (Tabrakan Tengah). Jika pusat bola berada dalam jarak 10 pixel dari pusat *paddle* (*abs(next_ball_pos_y - paddleX_pos_y) < 10*). Kecepatan vertikal bola (*Ball_vel_y*) reset menjadi 0, menghasilkan pantulan horizontal lurus dari *paddle*.

3. Pantulan Diagonal (Tambarkan samping). Berikut merupakan logika bola ketika pantulan diagonal terjadi.
    a. Jika bola sebelumnya bergerak lurus (*Ball_vel_y = 0*): Bola diberi kecepatan vertikal awal 1 atau -1, tergantung apakah bola mengenai bagian atas atau bawah *paddle* (*if (next_ball_pos_y > paddleX_pos_y) then Ball_vel_y = 1; else Ball_vel_y = -1;*). Ini memulai gerakan vertikal.
    b. Jika bola sudah memiliki kecepatan vertikal, kecepatan vertikal ditingkatkan sebesar 1. Ini dilakukan dengan mengambil nilai absolut kecepatan vertikal saat ini (*abs(Ball_vel_y)*), menambahkannya dengan 1, kemudian menerapkan kembali arah yang sesuai (positif jika memantul dari bagian bawah *paddle*, negatif jika dari atas). Fungsi *abs()* memastikan bahwa kecepatan vertikal selalu bertambah (menjadi lebih cepat) setelah memantul dari sisi *paddle*, terlepas dari arah aslinya.

Logika kolisi ini diterapkan secara simetris untuk kedua *paddle* (kiri dan kanan), memastikan perilaku yang konsisten.

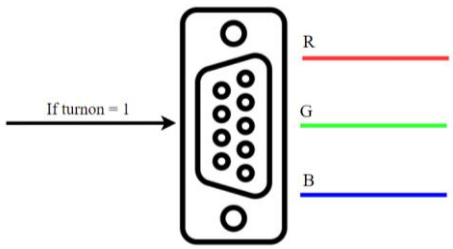
Selanjutnya, *font ROM* digunakan untuk menampilkan teks pada *display*. *Font ROM* berbentuk *array* 3 dimensi yang menyimpan representasi *bitmap* dari karakter angka dan huruf. Implementasi font ROM dapat dilihat pada Gambar 3.15.

```
type fontrom is array (0 to 30, 0 to 9, 0 to 5) of std_logic;
constant font_rom : fontrom := (
    -- Index 0: Digit 0
    (
        "111110",
        "100010",
        "100010",
        "100010",
        "100010",
        "100010",
        "100010",
        "111110",
        "000000",
        "000000"
    ),
    -- Index 1: Digit 1
    (
        "000100",
        "000100",
        "000100",
        "000100",
        "000100",
        "000100",
        "000100",
        "000100",
        "000000",
        "000000"
    ),
-- dan seterusnya
```

Gambar 3.15 Cuplikan Kode Font ROM

Font ROM adalah sebuah tempat penyimpanan data tetap yang berisi pola *pixel* untuk setiap angka. Pola ini tersusun dalam bentuk kolom dan baris, di mana setiap '1' mewakili *pixel* yang menyala dan setiap '0' mewakili *pixel* yang mati, membentuk tampilan visual dari digit tersebut. Data ini kemudian digunakan untuk menampilkan skor, dan teks seperti ‘PRESS START TO PLAY’, ‘WINNER’, dan *watermark* bertuliskan ‘SULTHAN FAIRUZANDY TEKNIK KOMPUTER’.

Selanjutnya, dengan menggunakan variabel *turnon*, yang dihasilkan dari logika modul *sync*, setiap variabel *turnon* berubah menjadi ‘1’, maka logika untuk menampilkan teks dan skor akan menghasilkan sinyal RGB seperti yang digambarkan pada Gambar 3.16, Variabel ini memastikan bahwa karakter ditampilkan sesuai *timing* dari monitor VGA.



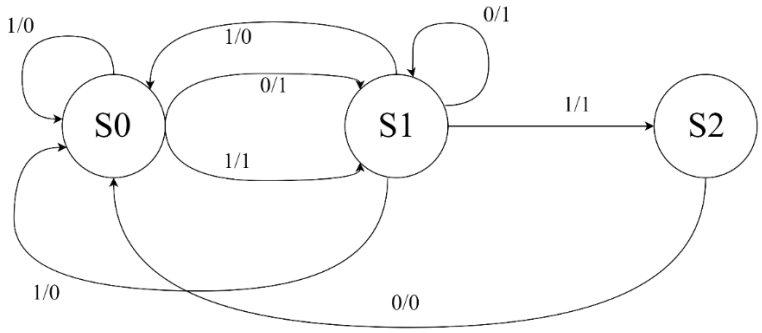
Gambar 3.16 Logika Pengiriman Sinyal RGB

Dalam penelitian ini, digunakan *output* RGB 3 bit, yang memungkinkan untuk mengeluarkan delapan warna pada *display*. Kombinasi nilai RGB dan *output* warna yang dapat dikeluarkan dapat dilihat pada Tabel 3.1.

Tabel 3.1 *Output* Warna 3 bit

R	G	B	*Output* Warna
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Berikutnya, FSM (*finite state machine*) adalah bagian yang bertanggung jawab atas transisi antar tiga *state* utama permainan yaitu, S0 (*Idle*), S1 (*Playing*), dan S2 (*Game Over*). Transisi dari sistem FSM digambarkan menggunakan diagram pada Gambar 3.17.



Gambar 3.17 Diagram Finite State Machine

FSM mengelola alur logika permainan secara keseluruhan, merefleksikan kondisi nyata permainan melalui *input* seperti tombol *start*, serta *output* seperti pergerakan bola (*move*). Transisi antar *state* disertai panah dengan label biner *input/output*. Berikut adalah penjelasan transisi antar *state*:

1. *State S0* atau *state idle,* adalah keadaan awal permainan di mana sistem berada dalam posisi diam, menunggu *input start_game* untuk memulai. Berikut adalah transisi yang dapat terjadi pada *State S0.*
    a. Transisi ke S1 (0/1):
        i) Kondisi (*Input* 0): Tombol *start_game* ditekan (*active low*) dan skor kedua pemain masih nol.
        ii) Aksi (*Output* 1): Permainan dimulai, dan sinyal *move* diaktifkan (*active high*) agar bola dan *paddle* dapat bergerak.
    b. Transisi ke S1 (1/1):
        i) Kondisi (*Input* 1): Tombol *start_game* tidak ditekan, namun permainan telah dimulai sebelumnya (salah satu atau kedua skor tidak nol).
        ii) Aksi (*Output* 1): Sistem langsung masuk ke S1 dan permainan berlanjut. Sinyal *move* tetap aktif.

c. *Loop* di S0 (1/0):
   i) Kondisi (*Input* 1): Tombol *start_game* tidak ditekan dan skor kedua pemain masih nol.
   ii) Aksi (*Output* 0): Sistem tetap berada di S0; sinyal *move* tetap tidak aktif.

2. *State S1* atau *state playing*, adalah keadaan saat logika inti permainan Pong dijalankan, meliputi pergerakan *paddle,* bola dan deteksi tabrakan (*collission*). Berikut adalah transisi yang dapat terjadi pada *State S1.*
   a. *Loop* di S1 (0/1):
      i) Kondisi (*Input* 0): Bola tidak keluar dari batas samping layar (tidak ada skor yang bertambah) dan belum ada pemain yang mencapai skor 7.
      ii) Aksi (*Output* 1): Permainan terus berlangsung; sinyal *move* tetap aktif.
   b. Transisi ke S0 (1/0):
      i) Kondisi (*Input* 1): Bola keluar dari batas kiri atau kanan layar (pemain mencetak skor).
      ii) Aksi (*Output* 0): Sistem kembali ke S0 untuk reset posisi bola dan menambah skor pemain lawan. Sinyal *move* dimatikan sementara.
   c. Transisi ke S2 (1/1):
      i) Kondisi (*Input* 1): Salah satu pemain mencapai skor 7.
      ii) Aksi (*Output* 1): Sistem berpindah ke S2 sebagai kondisi akhir permainan. Sinyal *move* tetap aktif .
3. *State S2* atau *state* akhir permainan, adalah keadaan yang menampilkan pemenang dan mengelola proses transisi kembali ke *state idle*. Berikut adalah transisi yang dapat terjadi pada *State S2.*
   a. Transisi ke S0 (0/0):
      i) Kondisi (*Input* 0): *Delay* selama lima detik setelah permainan selesai telah berakhir.
      ii) Aksi (*Output* 0): Sistem secara otomatis kembali ke S0. Skor pemain reset, dan sistem siap untuk permainan baru

Delay yang terdapat pada *State S2* diimplementasikan dengan sinyal *restart_counter*, yaitu *counter* integer yang berjalan berdasarkan *clock* mencapai batas 250.000.000 *ticks* untuk memberikan efek *delay* selama sekitar 5 detik dikarenakan *board* FPGA yang digunakan memiliki *clock* 50MHz (50.000.000 *ticks*).

Keseluruhan modul ini mengatur bagaimana *state* permainan berlangsung hingga bagaimana logika permainan divisualisasikan melalui sinyal warna RGB yang dihasilkan menggunakan logika permainan dan *multiplexer* yang kemudian disalurkan ke *display* melalui koneksi VGA.

3.3.4.6 *Display* (Monitor)

Bagian *display* atau monitor merupakan *output* utama dari keseluruhan sistem permainan Pong pada FPGA ini. Semua logika yang sudah diatur mulai dari posisi bola, pergerakan *paddle*, skor pemain, hingga teks akan divisualisasikan melalui sinyal VGA ke monitor. Kabel VGA dibutuhkan untuk menghubungkan FPGA ke monitor.

3.3.5 Blok Diagram dan Diagram *Wiring*

Hubungan utama antar komponen pada sistem permainan Pong digambarkan menggunakan blok diagram pada Gambar 3.18.

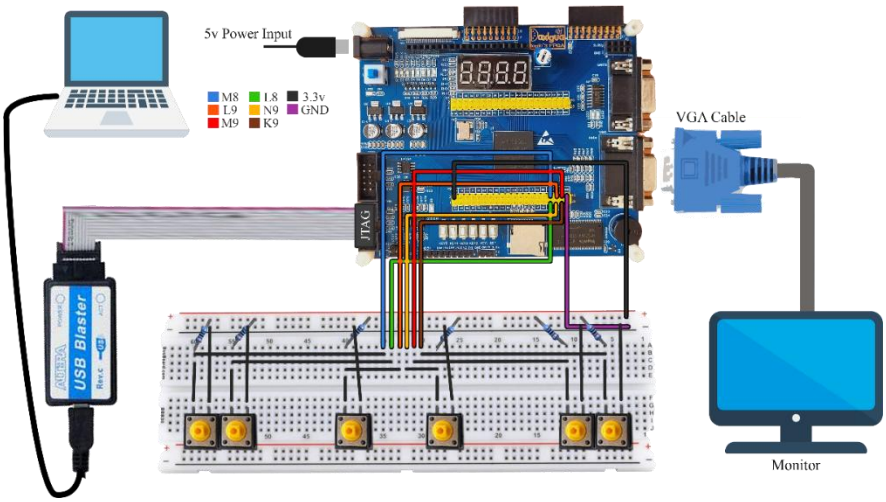
Gambar 3.18 Blok Diagram

Diagram pada Gambar 3.18 memberikan gambaran arsitektur sistem secara menyeluruh. Selanjutnya, akan digunakan diagram *wiring* (rangkaian) untuk menunjukkan koneksi *pin* dan perangkat keras secara lebih rinci.

Dalam tahap implementasi alat sistem, konfigurasi *pin* menjadi bagian penting yang harus diperhatikan agar setiap sinyal yang dihasilkan dari modul-modul logika dalam FPGA dapat terhubung dengan perangkat keras, seperti tombol *input, clock,* dan *pin* yang bertanggung jawab untuk menghubungkan masing-masing sinyal warna R, G, B dan sinkronisasi *display*. Proses ini disebut sebagai *pin assignment* dan dilakukan melalui perangkat lunak Quartus Prime melalui menu

pin planner, untuk menentukan koneksi antara sinyal logika dalam desain dengan *pin* pada *board* FPGA.

Konfigurasi *wiring* dapat dilihat pada Gambar 3.19, menunjukkan sebuah dokumentasi secara visual, dilengkapi dengan legenda warna yang menunjukkan hubungan *pin* yang terlihat langsung pada *yellow header* dengan *push-button.*



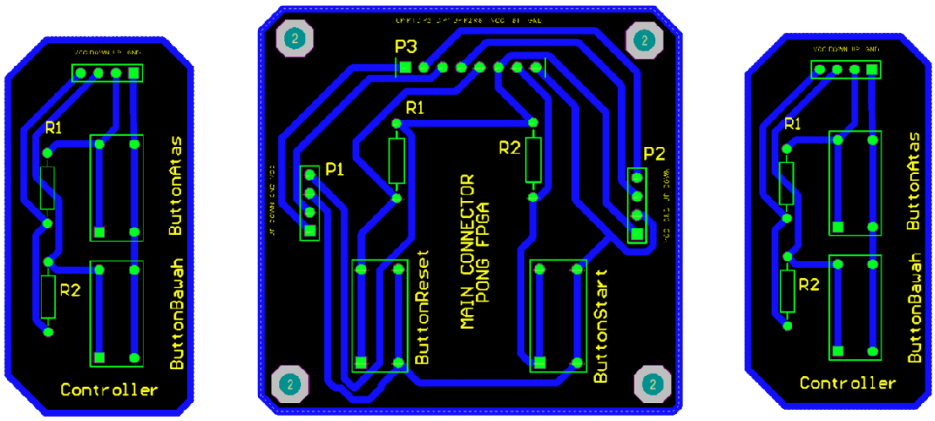
Gambar 3.19 Diagram Wiring Dan Pin Assignment Header

Selain *pin-pin* yang terlihat secara fisik pada *header board* seperti pada Gambar 3.19, terdapat juga *pin* yang terhubung secara internal pada *chip* FPGA, yang tidak terlihat seperti *pin* yang ada di *yellow header*. *Pin-pin* ini tetap dipetakan dalam pengaturan *pin* untuk kebutuhan logika sistem internal seperti sinyal *clock* (clk), warna (R, G, B), dan sinkronisasi VGA (*Hsync*, *Vsync*).

Dalam proses *wiring input* untuk kontrol permainan, digunakan komponen tambahan berupa *push-button*, resistor 10kΩ pada PCB untuk menghubungkan tombol ke FPGA. Konfigurasi ini menyesuaikan karakteristik *active low* pada *input* digital FPGA Cyclone IV, sehingga memerlukan *pull-up* resistor agar *input* dapat membaca logika tinggi ('1') saat tombol tidak ditekan.

Secara fisik, satu sisi tombol dihubungkan langsung ke *ground*, sementara sisi lainnya terhubung ke dua arah secara paralel, satu sisi ke *pin input* FPGA, dan sisi lainnya ke resistor 10kΩ yang dihubungkan ke tegangan 3.3V. Dengan konfigurasi ini, saat tombol tidak ditekan, arus mengalir melalui resistor ke *input* FPGA, sehingga terbaca sebagai logika tinggi ('1'). Namun saat tombol ditekan, jalur *input* terhubung langsung ke *ground*, menghasilkan logika rendah ('0').

Konfigurasi *wiring breadboard* pada Gambar 3.19, dalam penelitian ini akan diimplementasikan ke dalam sebuah PCB untuk menghasilkan desain yang lebih ringkas dan stabil. Desain PCB dapat dilihat pada Gambar 3.20.



Gambar 3.20 Desain PCB

Perancangan PCB Gambar 3.20 melibatkan dua desain modular utama. Modul konektor utama berfungsi sebagai penghubung *pin* dan daya serta *input* untuk memulai dan reset permainan antara *board* FPGA dan PCB. Sementara itu, Modul *controller* berperan sebagai perangkat *input* untuk mengendalikan *paddle* pada permainan, terdapat dua modul *controller* untuk mengendalikan *paddle* masing-masing pemain. Modul *controller* terhubung dengan konektor utama.

Penelitian ini menggunakan *board* FPGA Daxigua Logic'3 dengan *chip* Cyclone IV EP4CE6F17C8. Setiap *pin* yang digunakan dalam desain telah disesuaikan dengan dokumentasi *hardware* dari *board* yang digunakan, terutama untuk *pin* yang berhubungan langsung dengan monitor VGA melalui *VGA Port.*.

Konfigurasi *pin* akan diperlihatkan dalam sebuah tabel pada Tabel 3.2, meliputi keseluruhan komponen yang digunakan, mencakup fungsi masing-masing sinyal, *input*/*output*, serta *pin* FPGA yang ditetapkan.

Tabel 3.2 Konfigurasi *Pin* FPGA

No	***Input*/*Output***	Fungsi	Pin FPGA
1	*clk*	Sinyal *clock* utama sistem	PIN_E1
2	*reset*	*Reset* sistem ke kondisi awal	PIN_N9
3	*start_game*	Sinyal untuk memulai permainan	PIN_L9
4	*gamecontrol[0]*	Kontrol *paddle* pemain 1 ke atas	PIN_L8
5	*gamecontrol[1]*	Kontrol *paddle* pemain 1 ke bawah	PIN_M8
6	*gamecontrol[2]*	Kontrol *paddle* pemain 2 ke atas	PIN_M9
7	*gamecontrol[3]*	Kontrol *paddle* pemain 2 ke bawah	PIN_K9
8	R	*Output* warna merah ke monitor VGA	PIN_B16
9	G	*Output* warna hijau ke monitor VGA	PIN_J14
10	B	*Output* warna biru ke monitor VGA	PIN_J11
11	*Hsync*	Sinyal sinkronisasi horizontal VGA	PIN_C15
12	*Vsync*	Sinyal sinkronisasi vertikal VGA	PIN_C16

Konfigurasi *pin* yang tepat berperan penting dalam memastikan alat dapat bekerja sesuai dengan desain.

3.4 Tahap Implementasi

Dalam implementasi desain perangkat keras berbasis FPGA, terdapat tiga tahapan utama yang harus dilalui sebelum suatu desain dapat dijalankan pada perangkat fisik, yaitu *synthesis, fitter*, serta *assembler*. Proses ini memungkinkan deskripsi perangkat keras yang ditulis dalam bahasa VHDL untuk dikonversi menjadi konfigurasi yang dapat dieksekusi oleh FPGA. Sebelum tiga proses tersebut dapat dijalankan, ada beberapa langkah yang perlu dilakukan.

3.4.1. Analisis Desain

Analisis desain merupakan tahap awal yang diperlukan untuk menuliskan deskripsi dari sistem digital yang ingin dibuat menggunakan HDL, seperti VHDL, yang sudah disebutkan sebelumnya pada struktur sistem Pong. Pada tahap ini, struktur dasar sistem seperti modul, entitas, arsitektur, dan sinyal-sinyal utama didefinisikan dengan jelas.

File VHDL yang ditulis akan mencerminkan logika dan perilaku dari sistem digital, desain ini bisa terdiri dari satu *file* utama atau terbagi menjadi beberapa *file* modul yang saling terhubung secara hierarkis. Dalam penelitian ini, pendekatan modular digunakan, di mana *file* utama bertindak sebagai *top level entity* berupa *Pongmain* yang mengatur beberapa modul terpisah seperti modul *Sync*, *Objectgen*, dan *Clockdivision*. Cara ini memudahkan proses *debugging* serta meningkatkan keterbacaan dan fleksibilitas desain. Kode lengkap implementasi VHDL dapat dilihat pada Lampiran 2.

3.4.2 *Compilation*

Disebutkan sebelumnya, bahwa *synthesis, fitter*, serta *assembler* merupakan tahapan utama agar desain dapat dijalankan pada *board* FPGA. Tahapan tersebut terdapat pada langkah *Compilation.* Tampilan kompilasi beserta waktu yang dibutuhkan untung masing-masing tahap dapat dilihat pada Gambar 3.21.

Gambar 3.21 Menu Tahap Kompilasi

Berikut adalah penjelasan dari setiap tahapan seperti pada Gambar 3.21.

1. *Analysis and Synthesis*. Pada tahap ini, perangkat lunak Quartus Prime melakukan analisis *syntax* dan semantik terhadap kode VHDL untuk memastikan tidak ada kesalahan. Setelah itu, dilakukan proses *synthesis*, di mana desain VHDL dikonversi menjadi *netlist*, sebuah representasi logika dari desain dalam bentuk gerbang logika dan blok fungsional lainnya.
2. *Fitter*. Setelah *netlist* terbentuk, proses *fitting* menentukan bagaimana *logic elements* yang telah disintesis akan ditempatkan secara fisik di dalam *chip* FPGA. *Fitter* juga mempertimbangkan penempatan *pin I/O*, penggunaan *resource* internal seperti LUT (*Look-Up Table*), *flip-flop*, dan *clock*.
3. *Assembler.* Proses ini akan mengambil hasil tahapan *fitter* dan menggabungkannya menjadi *file* konfigurasi akhir, seperti .sof (*SRAM Object File*), yang digunakan untuk memprogram FPGA. *File* ini menjadi representasi biner dari seluruh desain sistem.
4. *Timing Analysis*. Setiap kali proses kompilasi dilakukan di Quartus Prime, Kode akan melewati tahap *Timing Analysis* untuk memastikan bahwa seluruh sinyal dalam desain dapat berjalan sesuai dengan batas waktu yang dibutuhkan oleh sistem. Analisis ini memeriksa akan pelanggaran waktu, seperti *Setup time violation*, atau data berubah terlalu cepat setelah sinyal *clock* naik.
5. *EDA Netlist Writer.* Langkah ini merupakan salah satu proses tambahan dalam tahapan kompilasi, berfungsi untuk mengekspor desain dalam bentuk *netlist* dengan format standar industri yang dapat digunakan ketika desain perlu disimulasikan atau dianalisis lebih lanjut menggunakan perangkat lunak pihak ketiga, serta saat integrasi sistem berskala besar. Namun, dalam penelitian ini,

hanya digunakan simulasi fungsional (*behavioral simulation*) menggunakan ModelSim untuk memverifikasi sinyal sinkronisasi horizontal dan vertikal pada modul *sync*, oleh karena itu ekspor desain *netlist* tidak diperlukan.

3.4.3 Simulasi

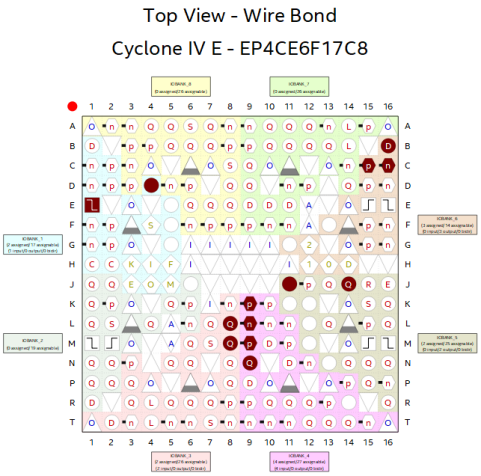
Sebelum desain diimplementasikan secara fisik, proses simulasi dilakukan untuk memverifikasi fungsionalitas logika sistem menggunakan ModelSim. Sebuah *file testbench* ditulis untuk memberikan *input* simulasi, merekayasa lingkungan FPGA apabila kode seolah-olah dijalankan langsung oleh *board* ke desain utama.

Simulasi dilakukan untuk menguji logika sinkronisasi VGA. Langkah ini penting, karena kesalahan pada *timing* sinyal tidak selalu tampak secara visual saat diuji langsung pada layar. Dengan menggunakan ModelSim, validasi terhadap kestabilan sinyal *Hsync* dan *Vsync* dapat dilakukan lebih awal untuk memastikan kesesuaian desain.

Namun, karena sistem ini bersifat interaktif dan berbasis visual, ModelSim tidak digunakan untuk menguji logika permainan secara keseluruhan. Pergerakan objek seperti *paddle* dan bola lebih efektif diverifikasi melalui pengujian langsung pada FPGA.

3.4.4 *Pin Assignment*

Setiap sinyal yang ada dalam desain harus dihubungkan dengan *pin* fisik pada *board* FPGA. Seperti yang sudah disebutkan sebelumnya, proses ini dilakukan menggunakan *tool Pin Planner* di Quartus Prime, di mana pengguna menentukan koneksi sinyal ke *pin* berdasarkan dokumentasi *board* yang digunakan. Tampilan *pin planner* beserta simbol spesifikasi masing-masing *pin* dapat dilihat pada Gambar 3.22.



Gambar 3.22 *Layout* Pin Cyclone IV EP4CE6F17C8 Dalam Menu *Pin Planner*

Selanjutnya, terdapat tabel konfigurasi yang digunakan untuk mengonfigurasi masing-masing *input* dan *output* ke *pin* yang sesuai, dapat dilihat pada Gambar 3.23.

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
B	Output	PIN_J11	5	B5_N0	PIN_J11	2.5 V		8mA (default)	2 (default)		
clk	Input	PIN_E1	1	B1_N0	PIN_E1	2.5 V		8mA (default)			
G	Output	PIN_J14	5	B5_N0	PIN_J14	2.5 V		8mA (default)	2 (default)		
gamecontrol[3]	Input	PIN_M9	4	B4_N0	PIN_M9	2.5 V		8mA (default)			
gamecontrol[2]	Input	PIN_K9	4	B4_N0	PIN_K9	2.5 V		8mA (default)			
gamecontrol[1]	Input	PIN_M8	3	B3_N0	PIN_M8	2.5 V		8mA (default)			
gamecontrol[0]	Input	PIN_L8	3	B3_N0	PIN_L8	2.5 V		8mA (default)			
Hsync	Output	PIN_C15	6	B6_N0	PIN_C15	2.5 V		8mA (default)	2 (default)		
R	Output	PIN_B16	6	B6_N0	PIN_B16	2.5 V		8mA (default)	2 (default)		
reset	Input	PIN_N9	4	B4_N0	PIN_N9	2.5 V		8mA (default)			
start_game	Input	PIN_L9	4	B4_N0	PIN_L9	2.5 V		8mA (default)			
Vsync	Output	PIN_C16	6	B6_N0	PIN_C16	2.5 V		8mA (default)	2 (default)		

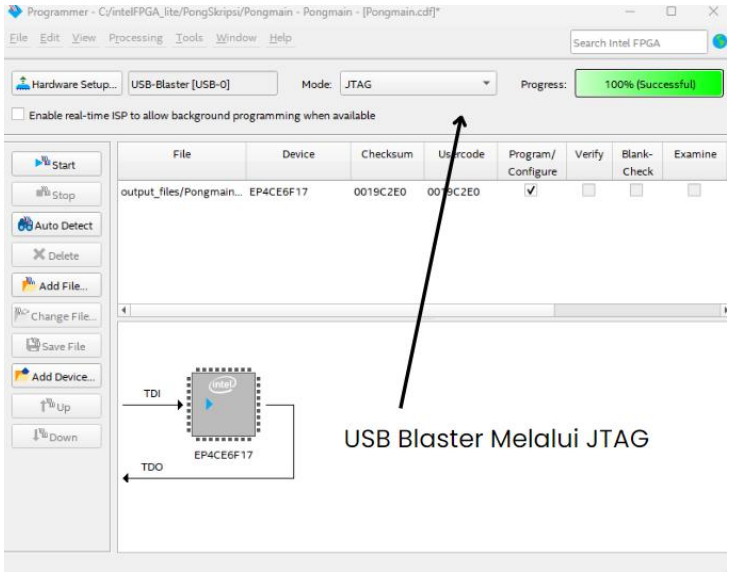
Gambar 3.23 Tabel Konfigurasi Pin Dalam Menu *Pin Planner*

Penentuan *pin* yang tepat memastikan koneksi ke perangkat keras seperti LED, tombol, VGA, atau modul lainnya dapat berjalan sesuai fungsi yang diinginkan. Dalam penelitian ini, *pin Clock*, GPIO, RGB, VGA, serta *Hsync*, dan *Vsync* disesuaikan dengan desain dan dokumentasi manufaktur *board* yang diberikan.

3.4.5 Program *board* FPGA

Langkah terakhir adalah mengunggah desain ke FPGA. *File* .sof, hasil dari proses *assembler* digunakan untuk mengonfigurasi *chip* FPGA melalui kabel USB Blaster melalui JTAG. Proses ini dilakukan melalui perangkat lunak Quartus Prime dengan memilih *tool Programmer* dan memilih USB Blaster sebagai *hardware*.

Menu *programmer* digunakan untuk mengonfigurasi FPGA dengan memilih metode yang akan digunakan untuk mengunggah data ke *board*. Tampilan dapat dilihat pada Gambar 3.24.



Gambar 3.24 Tampilan Menu *Programmer*

Setelah berhasil diprogram, FPGA akan mulai menjalankan desain secara langsung. Hasilnya dapat diamati melalui *input* dan *output* fisik, dengan tampilan di monitor, dan *button* yang dihubungkan melalui GPIO di PCB sudah dapat digunakan untuk menerima *input*.

3.5 Tahap Pengujian

Tahap pengujian adalah bagaimana data yang diperoleh akan dianalisis untuk mengevaluasi kinerja desain pada FPGA seperti yang dilakukan oleh Saha dkk. (2023). Analisis ini mencakup penggunaan sumber daya FPGA seperti *logic elements* dan konsumsi daya dalam implementasi permainan Pong. Tujuan utama analisis ini adalah untuk memastikan bahwa desain yang diimplementasikan efektif dan berjalan dengan baik dalam batasan yang ada pada *board* FPGA.

3.5.1 Rancangan Pengujian Sistem

Untuk pengujian sistem, dibagi menjadi dua bagian. Pengujian melalui ModelSim, dan pengujian melalui pengetesan alat secara langsung.

Pengujian melalui ModelSim dilakukan menggunakan *testbench* yang memberikan *input* seolah kode VHDL dijalankan langsung pada *board* FPGA. Dalam penelitian ini, pengujian ModelSim digunakan untuk menyimulasikan sinkronisasi VGA dalam modul *sync*.

Pengujian dilakukan berdasarkan *timing* VGA dengan resolusi yang digunakan berupa 640×480 @ 60Hz. *Output* sinyal *turnon, Hsync,* dan *Vsync,* akan diperhatikan untuk dibandingkan dengan *timing* VGA yang sesuai. Sinyal *turnon* mengatur kapan sinyal R, G, B, dapat dikirimkan, yaitu di saat rentang sinyal *Hcount* 144-784, *turnon* akan menjadi high (‘1’). *Hsync* dan *Vsync* berbeda dengan *turnon,* karena sinyal ini aktif ketika nilai menunjukkan low (‘0’). *Hsync* akan terjadi di saat penghitung horizontal atau *Hcount* berada pada rentang 0 hingga 95, sementara *Vsync* terjadi disaat penghitung vertikal atau *vcount* berada pada rentang 0 hingga 1. Jika hasil simulasi sinkronisasi VGA menunjukkan nilai yang diharapkan, maka pengujian dinyatakan berhasil.

Namun, pengujian alat secara langsung seperti respons tombol dilakukan dengan melihat hasil *output* secara *real-time* pada *display*. *Debugging* dapat dilakukan lebih cepat dibandingkan melalui simulasi berbasis *waveform* melalui ModelSim. Pendekatan ini mengarah pada kelancaran logika tanpa gangguan atau *glitch* selama permainan berlangsung. Pengujian *black-box* digunakan untuk menganalisa fungsionalitas sistem. Tabel 3.3 menunjukkan berbagai *event* yang akan diuji.

Tabel 3.3 Rencana Pengujian Perilaku Permainan

No	*Event*	*Output* Yang Diharapkan
1	Tombol *reset* ditekan	Sistem kembali ke kondisi awal (posisi & skor)
2	Tombol start ditekan	Permainan dimulai, bola mulai bergerak
3	Tombol atas *paddle* kiri ditekan	*Paddle* kiri naik
4	Tombol bawah *paddle* kiri ditekan	*Paddle* kiri turun
3	Tombol atas *paddle* kanan ditekan	*Paddle* kanan naik
4	Tombol bawah *paddle* kanan ditekan	*Paddle* kanan turun
7	Bola menyentuh batas layar bawah	Bola memantul ke atas
8	Bola menyentuh batas layar atas	Bola memantul ke bawah
9	Bola keluar dari batas kiri	Poin untuk pemain kanan bertambah
10	Bola keluar dari batas kanan	Poin untuk pemain kiri bertambah
11	Bola mengenai tengah *paddle*	Bola memantul lurus secara horizontal
12	Bola mengenai sisi atas/bawah *paddle*	Bola memantul dengan kecepatan vertikal yang meningkat
13	Salah satu pemain mencapai skor 7	Sistem menampilkan *display* kemenangan
14	Sistem berada di *display* kemenangan	Setelah 5 detik, sistem kembali ke tampilan awal (S0)

3.5.2 Rancangan Pengujian Penggunaan Sumber daya dan Konsumsi Daya

Pengujian penggunaan sumber daya dilakukan menggunakan fitur laporan *Compilation Report* pada Quartus Prime setelah proses sintesis kode. Informasi yang diperoleh meliputi *Logic elements, Registers, Combinational LUTs, dan Maximum Fan-out.*

Sementara itu, pengujian konsumsi daya dilakukan menggunakan fitur *Power Analyzer* yang terdapat dalam perangkat lunak Quartus Prime. Pengukuran ini memberikan estimasi konsumsi daya dari berbagai komponen sistem.

Rancangan analisis hasil pengujian data sumber daya dan konsumsi daya yang diperoleh dari hasil pengujian akan dianalisis secara deskriptif untuk mengevaluasi kinerja dalam implementasi sistem terhadap kapasitas FPGA. Data yang telah didapat akan menyoroti proporsi penggunaan terhadap kapasitas total *board* FPGA yang digunakan.

Logic Elements akan dianalisis sebagai indikator utama kompleksitas sistem, dengan mempertimbangkan persentase penggunaannya terhadap kapasitas total *chip*. *Combinational LUTs* akan diamati untuk menilai seberapa besar porsi logika kombinasi yang digunakan dibandingkan dengan logika yang dimiliki. Penggunaan register juga akan ditinjau untuk memahami karakteristik desain, apabila penggunaan register tinggi, maka sistem lebih cenderung ke dalam sistem sekuensial, dan sebaliknya cenderung kombinasional. Sementara itu, nilai *Maximum Fan-out* akan digunakan untuk mengidentifikasi potensi masalah *delay* atau pelanggaran waktu (*timing violation*).

Analisis penggunaan konsumsi daya akan difokuskan pada dua komponen utama, *Core Static Thermal Power* dan *I/O Thermal Power*, dengan total konsumsi daya sebagai indikator efisiensi sistem. Nilai *Core Dynamic Thermal Power* yang tercatat nol akan ditinjau dalam konteks penggunaan *Power Analyzer*. *Dynamic power* sangat dipengaruhi oleh pola *switching* dari *input real-time*, di mana dalam penelitian ini tidak dapat disimulasikan secara akurat. Oleh karena itu, fokus evaluasi akan diberikan pada nilai statis dan I/O sebagai dasar pertimbangan kinerja penggunaan daya secara keseluruhan.