

FGPA-BASED SUDOKU SOLVER WITH BACKTRACKING LOGIC

GROUP PA-19

ANGGOTA KELOMPOK

Wilman Saragih Sitio 2306161776
Ammar Fattan R 2306266981
Nelson Laurensius 2306161845
Aisya Rivelia Azzahra 2306161864

INTRODUCTION

BACKGROUND

Dalam era digital, pemecahan masalah logis dan algoritmik semakin dibutuhkan. Sudoku, teka-teki angka yang mengasah keterampilan analisis pola dan strategi, menjadi tantangan populer dalam pengembangan algoritma. Penyelesaian manual Sudoku, terutama dengan tingkat kesulitan tinggi, memakan waktu, sehingga dibutuhkan **algoritma pemecah Sudoku** yang efisien. Proyek ini bertujuan mengembangkan **Sudoku Solver** menggunakan algoritma **backtracking** untuk menyelesaikan teka-teki secara otomatis dan valid, serta memverifikasi solusinya. Selain memberikan solusi praktis, proyek ini juga bertujuan untuk mendalami pemrograman logis, algoritma pencarian, dan struktur data yang lebih efisien.



PROJECT DESCRIPTION

Sistem berbasis **VHDL** untuk menyelesaikan teka-teki **Sudoku 9x9** secara otomatis.

Menerima input **koordinat dan angka** melalui sinyal *i*, *j*, dan *puzzle_buffer* untuk mengisi grid awal.

Deteksi posisi sel kosong dan penerapan **validasi aturan** Sudoku (angka 1-9 unik di baris, kolom, dan kotak 3x3).

Backtracking: jika angka tidak valid, menghapus angka terakhir dan coba angka lain hingga solusi ditemukan.

Sinyal ready sebagai indikator bahwa Sudoku telah terisi dengan angka valid.

Disimulasikan di **ModelSim** untuk memastikan proses otomatis berjalan dengan benar dan sesuai aturan.

IMPLEMENTATION

EQUIPMENT



**Visual Studio Code
(IDE)**

Intel Quartus Prime

ModelSim

Github

IMPLEMENTATION

Entitas Input Handler bertugas untuk memasukkan nilai-nilai awal ke dalam grid Sudoku. Nilai ini diinputkan dalam bentuk koordinat (i, j) dan nilai sel. Proses ini menggunakan sinyal **puzzle_buffer** untuk mengisi grid Sudoku dengan angka dari 1 hingga 9, sementara angka 0 digunakan untuk mewakili sel yang kosong.



IMPLEMENTATION

```
begin
    case state_present is
        when idle =>
            if (start = '1') then
                state_next <= next_empty_cell;
            else
                state_next <= idle;
            end if;

        when next_empty_cell =>
            if (next_cell_found = '1') then
                state_next <= guess;
            elsif (all_cell_filled = '1') then
                state_next <= solve;
            else
                state_next <= next_empty_cell;
            end if;

        when guess =>
            if (error = '1') then
                state_next <= backtrack;
            elsif (valid = '1') then
                state_next <= next_empty_cell;
            else
                state_next <= guess;
            end if;

        when backtrack =>
            if (restored_last_valid_fill = '1') then
                state_next <= guess;
            else
                state_next <= backtrack;
            end if;

        when solve =>
            state_next <= idle;
    end case;
end process state_machine;
```

Entitas State Machine digunakan untuk mengatur alur logika penyelesaian teka-teki Sudoku. State utama yang digunakan meliputi:

IDLE: Menginisialisasi sistem dan menunggu sinyal start.

NEXT_EMPTY_CELL: Mencari sel kosong untuk diisi.

GUESS: Mencoba mengisi sel kosong dengan nilai yang valid.

BACKTRACK: Mengembalikan kondisi sebelumnya jika terjadi kesalahan (invalid guess).

SOLVE: Menandakan proses penyelesaian selesai dan memberikan hasil.

IMPLEMENTATION

Beberapa sinyal utama digunakan dalam implementasi solver ini, seperti:

- **selected_row** dan **selected_col**: Menyimpan posisi sel kosong yang sedang diproses.

```
signal selected_row      : integer range 0 to 9 := 0;  
signal selected_col      : integer range 0 to 9 := 0;
```

- **valid**: Menandakan apakah nilai yang dicoba valid.

```
signal valid           : std_logic := '0';
```

- **error**: Sinyal yang digunakan untuk memicu proses backtracking.

```
signal error           : std_logic := '0';
```

- **all_cell_filled**: Menandakan bahwa semua sel telah diisi.

IMPLEMENTATION

```
when guess =>
    -- Try placing a number
    symbol_var := conv_integer(unsigned(puzzle_buffer));

    if is_valid_placement(puzzle, selected_row, selected_col, symbol_var) then
        puzzle(selected_row, selected_col) <= symbol_var;
        valid <= '1';

        -- Save backtracking information
        stack_row(pointer) <= selected_row;
        stack_col(pointer) <= selected_col;
        pointer <= pointer - 1;
    else
        error <= '1';
    end if;
```

Bagian ini memuat algoritma inti yang bertugas memecahkan teka-teki Sudoku. Jika ditemukan sel kosong, algoritma akan mencoba memasukkan angka 1–9 secara berurutan. Fungsi `is_valid_placement` digunakan untuk memeriksa apakah angka yang diinput valid berdasarkan aturan Sudoku, yaitu **tidak boleh ada angka yang sama** pada baris, kolom, dan blok 3x3.

Jika semua angka tidak valid, sistem akan berpindah ke BACKTRACK untuk mengembalikan kondisi sebelumnya dan mencoba angka lain. Fitur Backtracking memastikan bahwa semua kemungkinan angka dicoba hingga solusi ditemukan.

TESTING AND ANALYSIS

TESTING

Untuk menganalisis kinerja dan fungsi dari sistem Sudoku Solver yang telah dibuat, dilakukan pengujian menggunakan testbench.

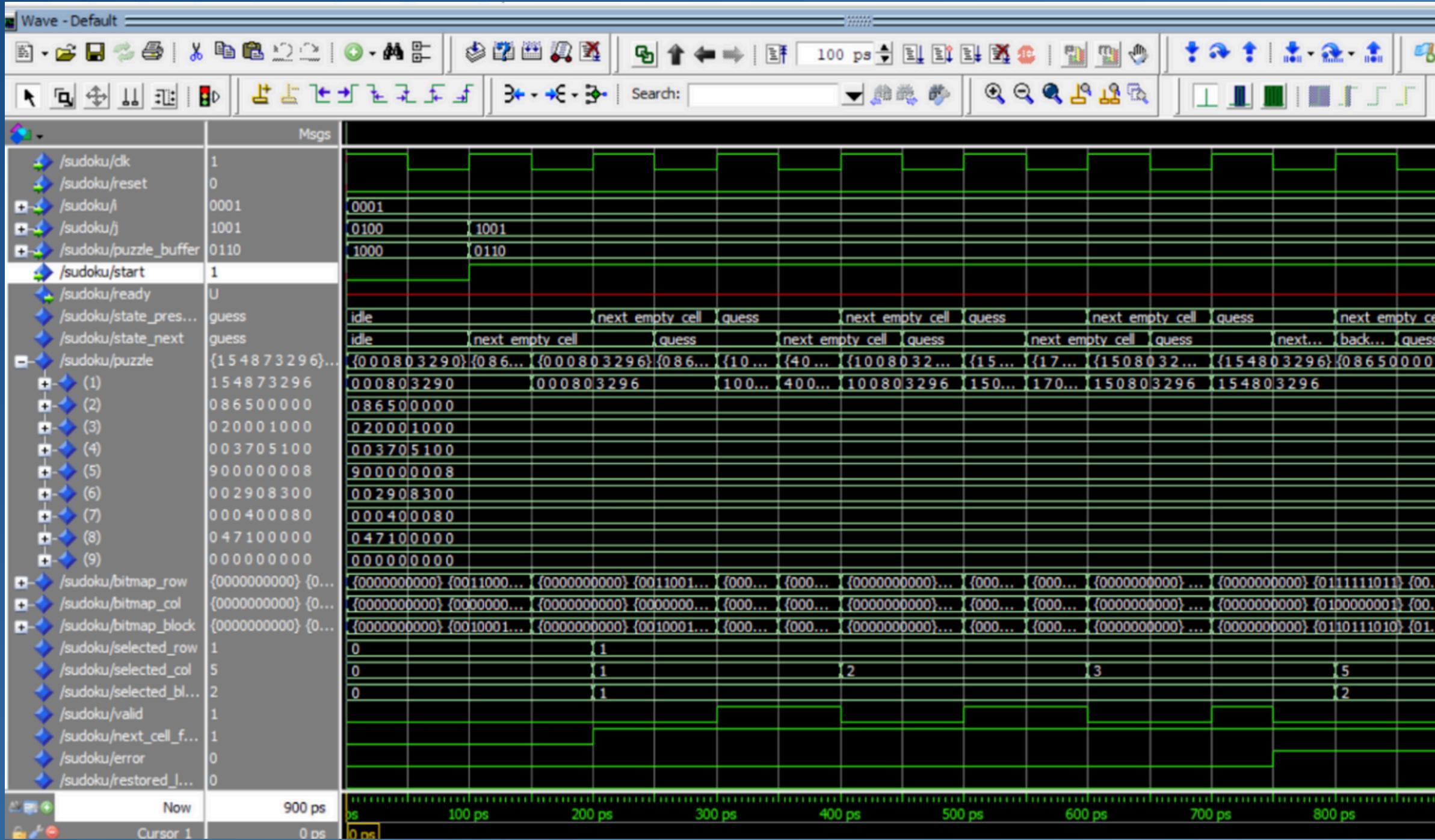
- **Proses Utama Testbench:**

1. Generasi Clock Cycle: Periode 10 ns (low & high masing-masing 5 ns) untuk sinkronisasi sistem.
2. Pemberian Input:
 - Input berupa koordinat sel (i, j) dan nilai angka (0 untuk sel kosong).
 - Diisi berurutan dari baris dan kolom pertama hingga terakhir.
 - Sinyal start memulai proses pemecahan.

- **Pemantauan & Validasi:**

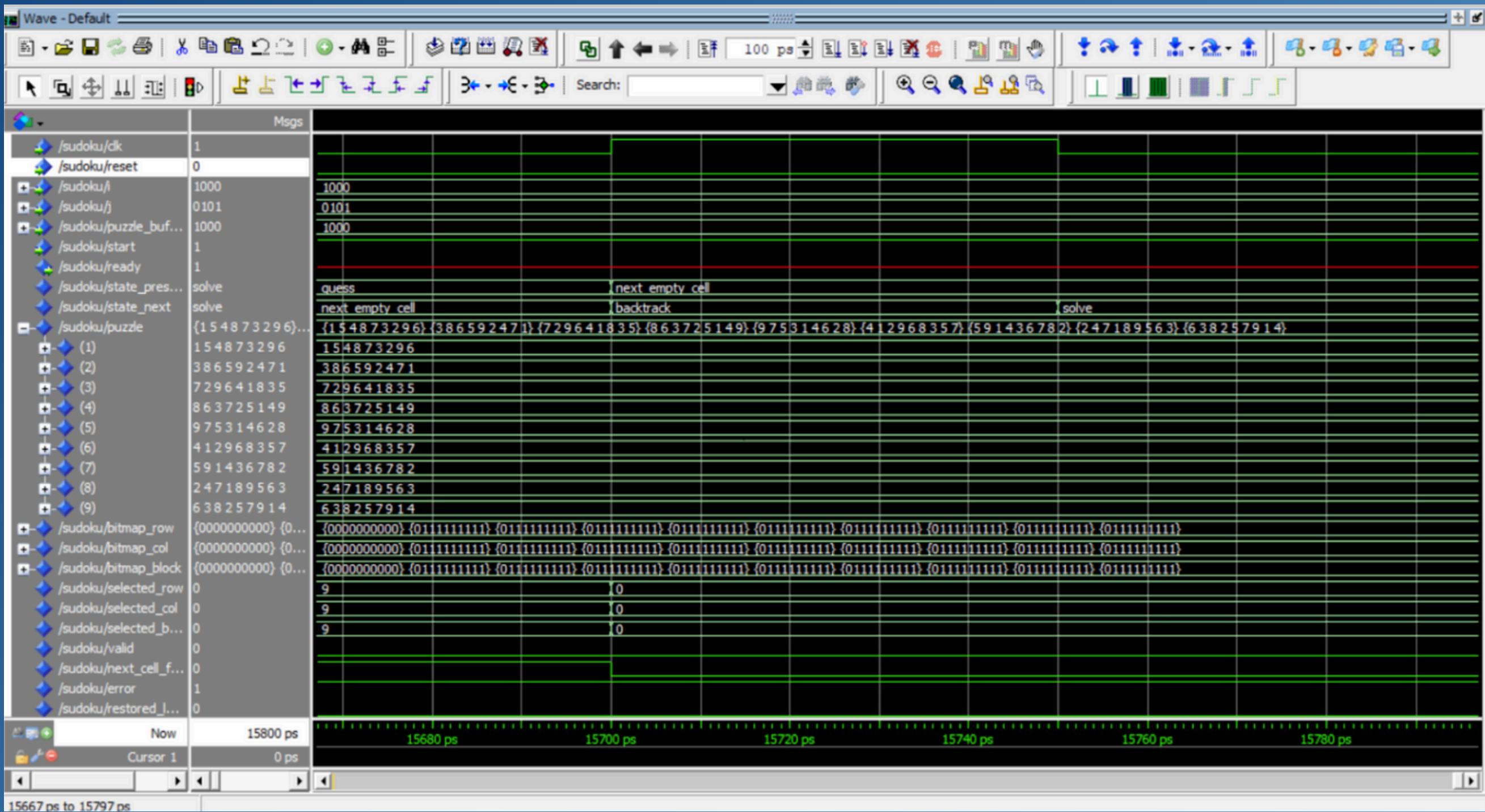
- Sinyal ready: Indikator proses selesai.
- Grid diperiksa berdasarkan aturan unik angka dalam baris, kolom, dan kotak 3x3.
- Hasil Simulasi: Ditampilkan sebagai waveform untuk analisis visual; keberhasilan ditandai oleh sinyal ready aktif dan grid terisi penuh.

RESULT



Testing Start dari TopLevel sudoku.vhd pada MODELSim

RESULT



Testing Result dari TopLevel sudoku.vhd pada MODELSim

ANALYSIS

Inisialisasi dan Pemberian Input Awal

- **Aktivasi Sinyal:** Mengaktifkan sinyal start (nilai 1) dan mengatur reset ke 0.
- **State Awal:** Memulai pada idle state dengan perubahan status pada rising edge clock
- **Input Grid:** Memasukkan nilai awal melalui koordinat baris (i), kolom (j), dan nilai angka (puzzle_buffer), seperti baris 1 kolom 4 (8) dan kolom 9 (6).

Proses Pemecahan dan Transisi State

- **Mencari Cell Kosong:** Beralih ke next_empty_cell untuk menemukan cell kosong di grid.
- **Menebak Angka:** Masuk ke state guess untuk mencoba angka yang valid pada koordinat yang ditemukan.
- **Validasi Aturan Sudoku:** Memastikan angka tidak berulang di baris, kolom, atau blok 3x3 sesuai aturan Sudoku.

Verifikasi dan Penyelesaian

- **Pengisian Grid:** Mengisi seluruh grid secara berurutan sesuai aturan Sudoku hingga semua cell terisi.
- **State Penyelesaian:** Setelah cell terakhir diisi, berpindah ke state solve.
- **Indikasi Selesai:** Mengaktifkan port ready (nilai 1) yang menandakan Sudoku berhasil diselesaikan.
- **Analisis Hasil:** Memverifikasi hasil melalui simulasi waveform untuk memastikan solusi sesuai ekspektasi.

CONCLUSION



CONCLUSION

Pengujian menggunakan testbench menunjukkan bahwa **sistem Sudoku Solver berbasis VHDL berhasil menyelesaikan teka-teki Sudoku 9x9 secara otomatis dan akurat**. Algoritma backtracking efektif dalam menemukan dan memvalidasi angka sesuai aturan Sudoku, memastikan tidak ada angka yang berulang di baris, kolom, atau kotak 3x3. Simulasi di ModelSim memverifikasi sinkronisasi sistem dengan clock cycle 10 ns dan hasil akhir yang sesuai, ditandai dengan sinyal ready aktif dan grid terisi penuh. Proyek ini tidak hanya berhasil mengembangkan solver Sudoku yang efisien, tetapi juga memperdalam pemahaman tentang penerapan algoritma pencarian dan desain sistem digital berbasis FPGA. Hasil ini memberikan kontribusi pada pengembangan perangkat keras dan pemrograman logika, serta dapat dijadikan referensi untuk pengembangan sistem serupa di masa depan.

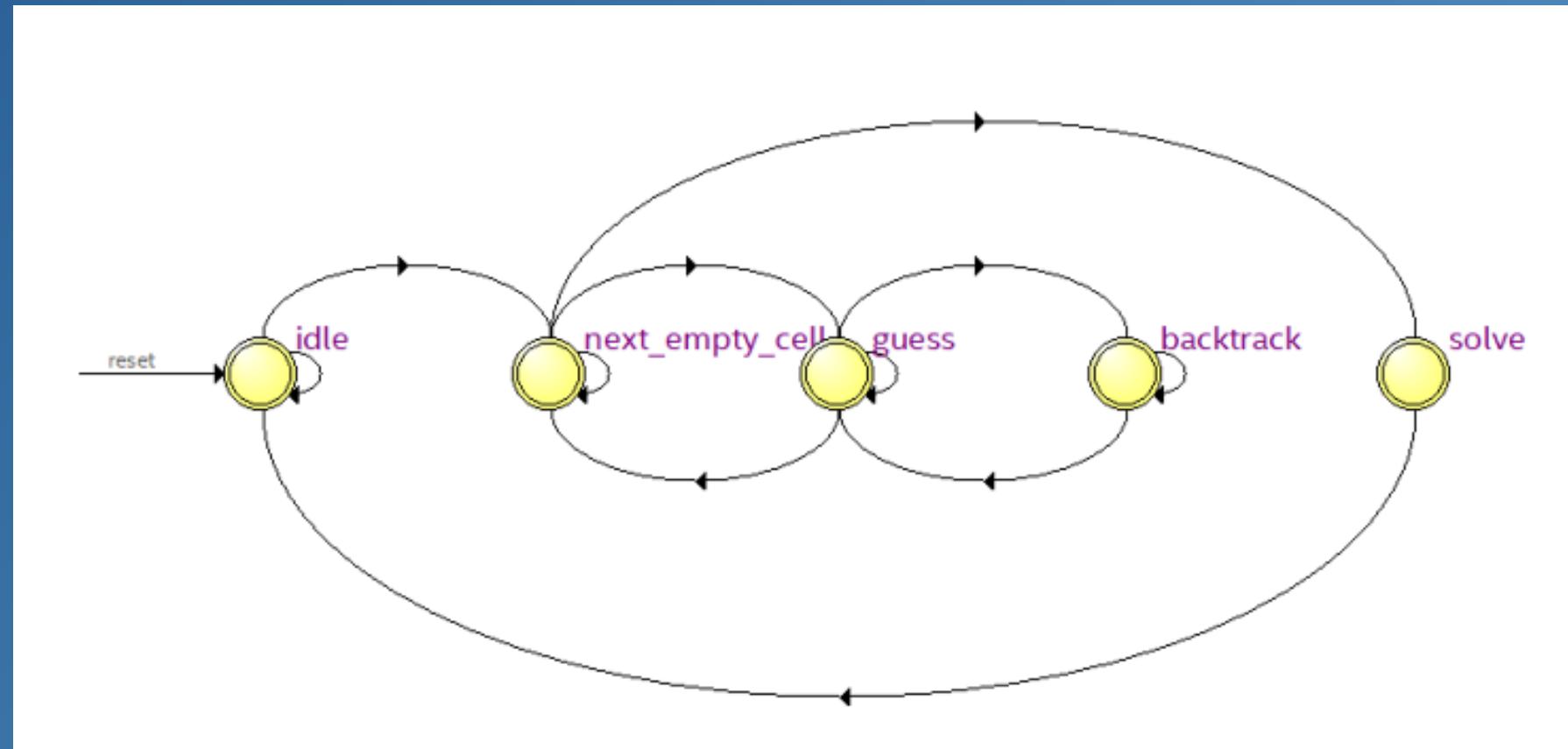
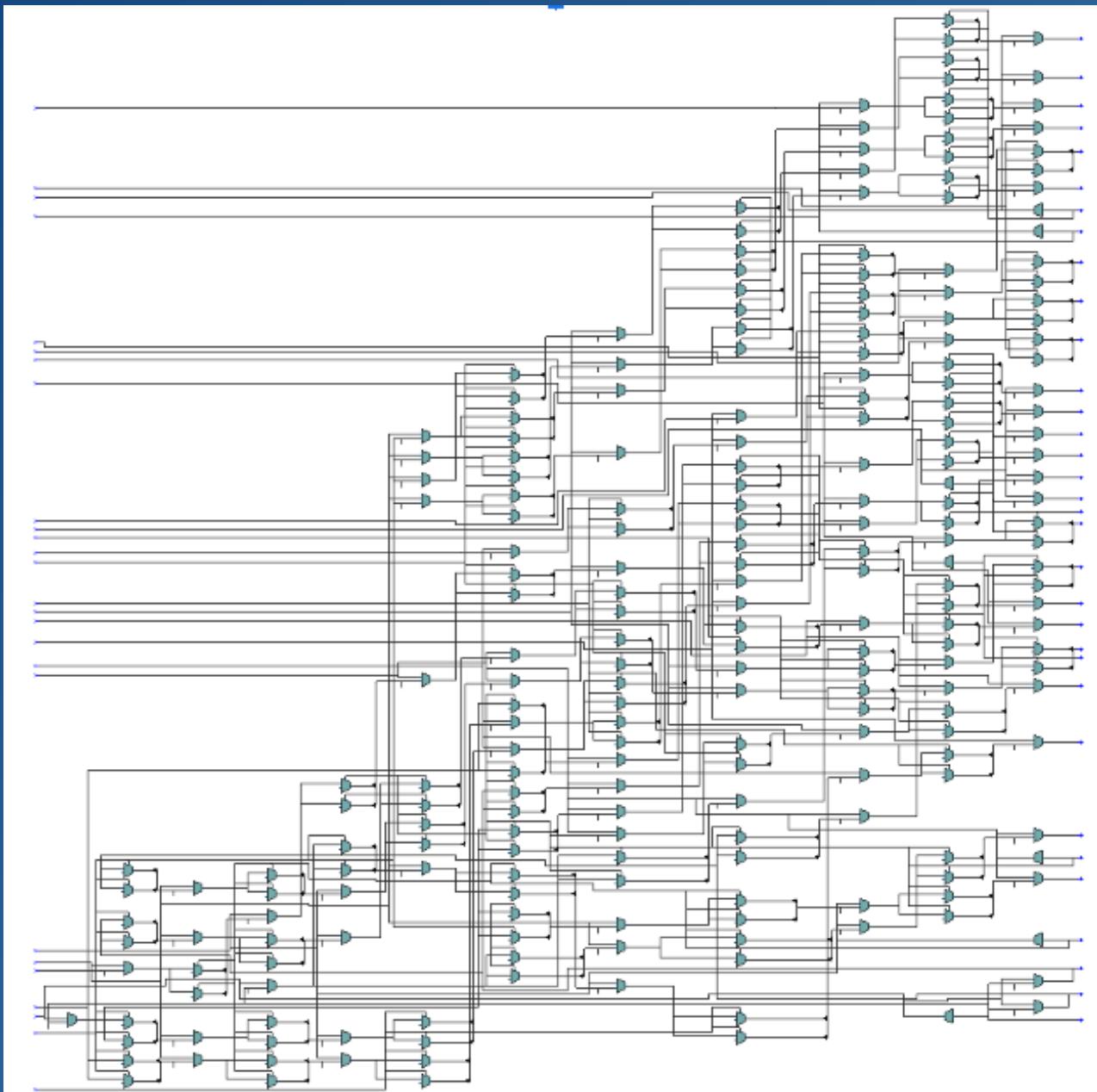
REFERENCES

REFERENCES

1. "Dataflow Style Programming in VHDL," Learn.Digilabdt. [Online]. Available: <https://learn.digilabdt.com/books/perancangan-sistem-digital/chapter/module-1-dataflow-style-programming-in-vhdl> [Accessed: Dec. 7, 2024].
2. Digilab DTE, "Structural Style Port Mapping and Generic Map," [Online]. Available: <https://learn.digilabdt.com/books/digital-system-design/page/structural-style-port-mapping-and-generic-map>. [Accessed: Dec. 7, 2024].
3. Digilab DTE, "VHDL Modularity," [Online]. Available: <https://learn.digilabdt.com/books/digital-system-design/page/vhdl-modularity>. [Accessed: Dec. 7, 2024].
4. Digilab DTE, "Array and Types in VHDL," [Online]. Available: <https://learn.digilabdt.com/books/digital-system-design/page/array-and-types-in-vhdl>. [Accessed: Dec. 7, 2024].
5. Electronics Tutorial, "Loop Statements in VHDL," [Online]. Available: <https://www.electronics-tutorial.net/VHDL/Concurrent-Statements/Loop-statements/>. [Accessed: Dec. 7, 2024].
6. FPGA Insights, "Mastering Loops in VHDL: Enhancing Flexibility and Performance in Hardware Design," [Online]. Available: <https://fpgainsights.com/fpga/mastering-loops-in-vhdl-enhancing-flexibility-and-performance-in-hardware-design/#:~:text=FOR%20Loop%20in%20VHDL,-The%20FOR%20loop&text=It%20allows%20you%20to%20specify,each%20value%20within%20that%20range.&text=end%20loop%3B,are%20executed%20for%20each%20iteration>. [Accessed: Dec. 7, 2024].
7. DigiLabDTE, "FSM Implementation Example in VHDL," Learn DigiLabDTE, [Online]. Available: <https://learn.digilabdt.com/books/digital-system-design/page/fsm-implementation-example-in-vhdl> [Accessed: Dec. 7, 2024].
8. GeeksforGeeks, "Unified Modeling Language (UML) State Diagrams," GeeksforGeeks, [Online]. Available: <https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/> [Accessed: Dec. 7, 2024].

APPENDICES

PROJECT SCHEMATIC



DOCUMENTATION



THANK YOU

