FUNDAMENDTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

FGPA-BASED SUDOKU SOLVER WITH BACKTRACKING LOGIC

GROUP PA-19

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

PREFACE

Puji dan syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat, karunia, dan petunjuk-Nya sehingga proyek akhir ini dapat kami cellesaikan dengan baik. Proyek ini merupakan hasil kerja keras, dedikasi, serta kolaborasi tim Kelompok PA19 yang terdiri dari kami Nelson Laurensius, Wilman Saragih Sitio, Aisya Rivelia Azzahra, dan Ammar Fattan.

Teka-teki Sudoku yang menantang dan membutuhkan pemecahan logis telah mendorong adanya solusi berbasis teknologi yang lebih efisien. Dalam proyek akhir ini, kami melakukan percobaan dalam melakukan implementasi algoritma dan desain sistem Sudoku Solver yang dirancang untuk menyelesaikan teka-teki Sudoku secara akurat, cepat, dan optimal dengan perbandingan terhadap kinerja dari *software*. Proyek ini diharapkan dapat menjadi referensi dalam pengembangan sistem serupa di masa depan serta sebagai kontribusi dalam bidang pemrograman logika dan sistem digital.

Kami menyadari bahwa proyek ini masih memiliki kekurangan dan keterbatasan. Oleh karena itu, kami sangat terbuka terhadap kritik, saran, dan masukan yang membangun guna pengembangan lebih lanjut di masa mendatang. Semoga proyek ini dapat memberikan manfaat dan inspirasi bagi semua pihak yang tertarik dalam pengembangan sistem pemecah teka-teki logika.

Depok, December 7, 2024

Group PA-19

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Dalam era teknologi digital saat ini, pemecahan masalah dengan logis dan algoritmis semakin dibutuhkan untuk menyelesaikan berbagai permasalahan nyata. Salah satu contoh tantangan logis yang populer adalah teka-teki Sudoku. Sudoku adalah permainan teka-teki angka yang membutuhkan keterampilan logis, analisis pola, dan strategi untuk mengisi setiap cell kosong di grid 9x9 dengan angka 1 hingga 9, tanpa ada angka yang berulang pada setiap baris, kolom, dan kotak 3x3. Permainan ini tidak hanya mengasah kemampuan berpikir kritis, tetapi juga menjadi objek kajian menarik dalam dunia pemrograman dan algoritma.

Seiring dengan perkembangan teknologi, penyelesaian Sudoku secara manual seringkali memakan waktu, terutama untuk tingkat kesulitan yang lebih tinggi. Oleh karena itu, dibutuhkan algoritma pemecah Sudoku atau lebih dikenal Sudoku Solver yang mampu menyelesaikan teka-teki tersebut secara efisien. Implementasi algoritma pemecah Sudoku menjadi sebuah tantangan menarik karena melibatkan penerapan konsep pemrograman logis, struktur data, serta pendekatan algoritma pencarian dan backtracking.

Proyek ini bertujuan untuk mengembangkan sistem Sudoku Solver yang dapat menyelesaikan teka-teki Sudoku dengan cepat dan akurat menggunakan metode pemrograman logika. Dengan menggunakan pendekatan yang sistematis, program ini diharapkan mampu mengisi setiap cell kosong pada grid Sudoku, memverifikasi solusi, serta memberikan jawaban yang valid sesuai dengan aturan permainan. selain itu, proyek ini juga menjadi sarana pembelajaran dalam memahami algoritma kompleks, implementasi struktur data, dan simulasi sistem digital.

Melalui proyek akhir ini, diharapkan program Sudoku Solver dapat memberikan manfaat sebagai solusi praktis untuk pemecahan teka-teki logika, menjadi referensi dalam pengembangan sistem serupa di masa depan, serta mendorong pengembangan teknologi berbasis pemrograman logis dan algoritma yang lebih efisien.

1.2 PROJECT DESCRIPTION

Sudoku Solver adalah sistem *solver* teka-teki Sudoku yang dibuat untuk menyelesaikan teka-teki berukuran 9x9 grid secara otomatis menggunakan FPGA berbasis VHDL. Sistem ini menggunakan algoritma backtracking untuk mencari solusi yang valid dengan sesuai *rule* Sudoku, di mana angka 1 hingga 9 harus ditempatkan di setiap cell pada grid tanpa ada yang terulang di baris, kolom, atau kotak 3x3.

Sistem ini dimulai dengan menerima input yang diberikan oleh pengguna melalui tiga sinyal utama yakni i sebagai koordinat baris, j sebagai koordinat kolom, dan puzzle_buffer sebagai nilai input ke dalam grid Sudoku dalam proses penyelesaian. Input tersebut digunakan untuk mengisi grid awal dengan angka yang sudah diberikan. Proses ini dilakukan pada bagian awal untuk inisialisasi grid, di mana data Sudoku yang ada akan dipetakan dan disimpan dalam memori internal FPGA.

Setelah grid Sudoku diinisialisasi, sistem secara otomatis mengecek setiap cell untuk menemukan cell yang kosong yang ditandai nilai 0. Setelah menemukan cell kosong, sistem kemudian melakukan pencarian angka yang tepat untuk ditempatkan pada cell tersebut. Pencarian angka ini dilakukan dengan memeriksa apakah angka yang dicoba memenuhi tiga kondisi utama: angka tersebut tidak boleh berulang di baris yang sama, kolom yang sama, atau blok 3x3 yang sama.

Untuk memastikan bahwa solusi yang dihasilkan valid, terdapat mekanisme validasi angka yang memastikan bahwa setiap angka yang dimasukkan sesuai dengan aturan permainan Sudoku dan juga bantuan dari input *user*. Jika angka yang dicoba tidak valid, sistem akan backtrack, yaitu menghapus angka yang terakhir dimasukkan dan mencoba angka lain. Proses backtracking ini akan terus berlanjut hingga sistem menemukan solusi yang valid untuk semua cell kosong dalam grid.

Proses backtracking menggunakan stack yang dideklarasikan pada newtype.vhd untuk menyimpan posisi cell dan angka yang telah dicoba, sehingga jika terjadi kesalahan, sistem dapat kembali ke posisi sebelumnya dan mencoba angka yang berbeda. Proses ini berulang sampai semua cell pada grid terisi dengan angka yang valid, dan akhirnya grid Sudoku dianggap telah

selesai. Setelah Sudoku selesai diselesaikan, sistem akan memberikan sinyal output ready yang menandakan bahwa grid Sudoku telah berhasil terisi dengan angka yang valid dan sesuai dengan aturan permainan.

1.3 OBJECTIVES

The objectives of this project are as follows:

1. Membuat sistem pemecah teka-teki Sudoku berbasis VHDL yang dapat menyelesaikan grid Sudoku 9x9 secara otomatis.
2. Mengimplementasikan algoritma backtracking untuk menemukan solusi yang valid sesuai aturan permainan Sudoku.
3. Menerapkan validasi angka pada setiap baris, kolom, dan kotak 3x3 untuk memastikan solusi yang dihasilkan akurat dan optimal.
4. Mengintegrasikan seluruh modul pemrosesan, validasi, dan kontrol ke dalam satu sistem terstruktur berbasis VHDL.
5. Mensimulasikan dan menguji kinerja sistem menggunakan ModelSim untuk memastikan keakuratan dan keberhasilan penyelesaian grid Sudoku.

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Perancang konsep dasar dan dokumentasi	● Membahas dan merancang ide proyek ● Melengkapi kode dan membuat testbench	Nelson Laurensius

	• Pembuatan dokumentasi dan laporan	
Code designer dan dokuementasi laporan	• Membahas dan merancang ide proyek • Membuat base main code • Pembuatan dan dokumentasi laporan	Wilman Saragih Sitio
Pembuatan materi dan bahan presentasi	• Membahas dan merancang ide proyek • Membuat PPT • Melengkapi laporan	Aisya Rivelia Azzahra
Role 4	• Membahas dan merancang ide proyek • Membuat media powerpoint	Person 4

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

The tools that are going to be used in this project are as follows:

- Visual Studio Code (IDE)
- ModelSim
- Intel Quartus Prime
- Github

2.2 IMPLEMENTATION

Pada pengimplementasiannya digunakan backtracking untuk mencoba mengisi setiap sel kosong dengan angka yang valid, sambil memeriksa kendala-kendala yang ada dan dengan input buffer dari *user*. Bitmaps digunakan digunakan untuk memetakan koordinat tiap angka-angka yang sudah ada di baris, kolom, atau blok tertentu, sehingga kita bisa lebih efisien dalam mencari angka yang belum digunakan.

0	0	0	0	0	3	2	9	0
0	8	6	5	0	0	0	0	0
0	2	0	0	0	1	0	0	0
0	0	3	7	0	5	1	0	0
9	0	0	0	0	0	0	0	8
0	0	2	9	0	8	3	0	0
0	0	0	4	0	0	0	8	0

0	4	7	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Tabel 1. Puzzle Table

Row bitmap pada Tabel 2 menunjukkan angka-angka apa saja yang sudah ada di setiap baris. Misalnya, jika di baris pertama sudah ada angka 2 di kolom kedua, maka di peta baris akan diberi tanda di kolom pertama untuk angka 2 begitu juga sebaliknya utnuk Column bitmap yang ada pada Tabel 3.

Row	Element								
	1	2	3	4	5	6	7	8	9
1	0	1	1	0	0	0	0	0	0
2	0	0	0	0	1	1	0	1	0
3	1	1	0	0	0	0	0	0	0
4	1	0	1	0	1	0	1	0	0
5	0	0	0	0	0	0	0	1	1
6	0	1	1	0	0	0	0	1	1
7	0	0	0	1	0	0	0	1	0
8	1	0	0	1	0	0	1	0	0
9	0	0	0	0	0	0	0	0	0

Tabel 2. Row Bitmap

Col	Element								
	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	9
2	0	1	0	1	0	0	0	1	0
3	0	1	1	0	0	1	1	0	0
4	1	0	0	1	1	0	1	0	1
5	0	0	0	0	0	0	0	0	0
6	1	0	1	0	1	0	0	1	0
7	1	1	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	1
9	0	0	0	0	0	0	0	1	0

Tabel 3. Column Bitmap

Block	Element								
	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	1	0	1	9
2	1	0	1	0	1	0	0	0	0
3	0	1	0	0	0	0	0	0	1
4	0	1	1	0	0	0	0	0	1
5	0	0	0	0	1	0	1	1	1
6	1	0	1	0	0	0	0	1	0

Tabel 3. Block Bitmap

Ketiga bitmap tersebut digunakan untuk memvalidasi angka yang bisa ditempatkan di setiap sel kosong. Ketika kita menemukan sel kosong, kita akan memeriksa angka-angka yang bisa ditempatkan di sel tersebut dengan mengecek peta baris, kolom, dan blok yang terlibat. Kemudian, berdasarkan map-map ini, *user* juga dapat memilih angka yang valid untuk ditempatkan di sel tersebut.

Desain ini terdiri dari beberapa komponen utama yang bekerja secara terkoordinasi untuk menyelesaikan teka-teki Sudoku. Komponen-komponen utama tersebut meliputi:

- **Entity Sudoku**: Mengatur alur utama dari sistem, mulai dari inisialisasi puzzle hingga proses pencarian solusi.
- **Fungsi Validasi**: Memeriksa apakah angka yang dimasukkan valid pada baris, kolom, dan blok 3x3.
- **FSM**: Mengontrol alur proses backtracking dan pencarian angka yang valid.
- **Stack**: Menyimpan posisi cell dan angka yang telah dicoba, memungkinkan backtracking saat terjadi kesalahan.
- **Bitmap**: Menyimpan status dari setiap angka yang ada pada baris, kolom, dan blok 3x3, digunakan untuk validasi.

Pertama didefinisikan dulu package, ini dilakukan agar dapat memungkinkan abstraksi dari tipe data yang lebih kompleks seperti array dua dimensi atau stack. cell_type adalah subtype dari integer yang membatasi nilai antara 0 hingga 9, mewakili nilai-nilai dalam grid/map, symbol sebagai subtype std_logic untuk merepresentasikan status bit dalam validasi angka, sudoku_array adalah array dua dimensi 9x9 yang menyimpan grid Sudoku dengan tipe cell_type, sedangkan bitmap_array representasi array 10x10 yang menyimpan bitmap untuk memeriksa validitas

angka 0 - 9. Lalu ada guess_type yakni array satu dimensi yang menyimpan status validasi *temporary* dari tebakan angka, dan stack adalah array satu dimensi yang menyimpan posisi dan angka yang dicoba selama proses backtracking, membantu mengembalikan langkah sebelumnya jika tebakan tidak valid. Semua tipe ini membantu memodelkan data yang digunakan dalam proses penyelesaian teka-teki Sudoku dengan pendekatan algoritma backtracking dan validasi.

```
library ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

package newtype is
    subtype cell_type is integer range 0 to 9;
    subtype symbol is std_logic;
    type sudoku_array is array (integer range 1 to 9, integer range 1 to 9) of cell_type;
    type bitmap_array is array (integer range 0 to 9, integer range 0 to 9) of symbol;
    type guess_type is array (integer range 1 to 9) of symbol;
    type stack is array (integer range 0 to 127) of integer range 0 to 9;
end newtype;
```

Gambar 1. Package newtype.vhd

Entity sudoku mendefinisikan interface yang mencakup beberapa port untuk komunikasi dengan komponen lainnya. Port clk dan reset digunakan untuk mengontrol proses operasi dengan sinyal clock dan reset. Port i dan j masing-masing menerima input berupa koordinat baris dan kolom dalam bentuk vector 4-bit, yang memungkinkan untuk menentukan posisi sel dalam grid 9x9. Selanjutnya, ada port puzzle_buffer juga menerima input 4-bit yang berfungsi untuk membantu dalam proses guessing dan kesalahan saat backtracking dengan angka 0-9 yang akan diinputkan ke dalam posisi yang ditentukan oleh i dan j. Start adalah sinyal kontrol untuk memulai proses, sementara ready adalah output yang menunjukkan bahwa solver telah siap untuk menerima input selanjutnya.

```
entity sudoku is
    port(
        clk, reset: in std_logic;
        i : in std_logic_vector(3 downto 0);
        j : in std_logic_vector(3 downto 0);
        puzzle_buffer : in std_logic_vector(3 downto 0);
        start: in std_logic;
        ready: out std_logic
    );
end sudoku;
```

Gambar 2. Entity Sudoku.vhd

Sinyal-sinyal tersebut digunakan digunakan untuk mengontrol alur dan status dalam proses penyelesaian teka-teki Sudoku. Sinyal state_present dan state_next mencatat status dari FSM, puzzle menyimpan grid Sudoku yang sedang diproses, sedangkan bitmap_row, bitmap_col, dan bitmap_block digunakan untuk memvalidasi angka pada baris, kolom, dan blok 3x3. Sinyal valid, next_cell_found, dan error mengatur validitas *guess* dan proses pencarian *cell* kosong. Stack baik itu stack_row dan stack_col menyimpan koordinat yang telah dicoba untuk mendukung backtracking, sementara pointer mengarah pada posisi dalam stack. Sinyal symbol_variable menyimpan angka yang dites, serta update digunakan untuk memperbarui bitmap saat diperlukan.

```
signal state_present, state_next: state_type;

signal puzzle : sudoku_array := (
    (0,0,0,0,0,3,2,9,0),
    (0,8,6,5,0,0,0,0,0),
    (0,2,0,0,0,1,0,0,0),
    (0,0,3,7,0,5,1,0,0),
    (9,0,0,0,0,0,0,0,8),
    (0,0,2,9,0,8,3,0,0),
    (0,0,0,4,0,0,0,8,0),
    (0,4,7,1,0,0,0,0,0),
    (0,0,0,0,0,0,0,0,0)
);

signal bitmap_row : bitmap_array := (others => (others => ('0')));
signal bitmap_col : bitmap_array := (others => (others => ('0')));
signal bitmap_block : bitmap_array := (others => (others => ('0')));

signal selected_row: integer range 0 to 9;
signal selected_col: integer range 0 to 9;
signal selected_block: integer range 0 to 9;
signal valid: std_logic := '0';
signal next_cell_found: std_logic := '0';
signal error: std_logic := '0';
signal restored_last_valid_fill: std_logic := '0';
signal all_cell_filled: std_logic := '0';
signal stack_row : stack := (others => (0));
signal stack_col : stack := (others => (0));
signal pointer : integer range 0 to 127 := 127;
signal symbol_variable: integer range 0 to 9;
signal update: std_logic := '0';
```

Gambar 3. Signal Declaration

Pada bagian *architecture*, FSM mengatur beberapa state, yaitu idle untuk menunggu sinyal *start*, next_empty_cell untuk mencari *cell* kosong berikutnya, guess untuk mencoba angka dan memeriksa validitasnya, dan backtrack untuk kembali ke posisi sebelumnya jika tebakan

tidak valid. Sistem ini juga menggunakan stack untuk menyimpan posisi dan angka yang telah dicoba, serta bitmap untuk menyimpan status validasi angka pada baris, kolom, dan blok 3x3, memastikan bahwa setiap langkah yang diambil sesuai dengan aturan Sudoku dan efisien dalam pencarian solusi.

```
process(clk, reset, start)
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;
```

Gambar 4. FSM

Function tersebut digunakan untuk menentukan nomor blok 3x3 pada grid berdasarkan koordinat baris dan kolom. Menghitung posisi blok berdasarkan pembagian koordinat baris dan kolom dengan 3 menggunakan modulus/remainder untuk mengidentifikasi blok 3x3 yang sesuai. Setelah itu, menghitung index sebagai representasi koordinat awal *block* di grid. Berdasarkan index ini, function kemudian menggunakan case untuk me-*return* nomor blok yang sesuai, dari 1

hingga 9, yang digunakan dalam proses penyelesaian. Jika index tidak cocok dengan blok yang valid, fungsi mengembalikan nilai 0 sebagai indikasi kesalahan atau blok yang tidak valid.

```
function block_number_function ( i,j: integer) return integer is
    variable block_i : integer range 0 to 10;
    variable block_j : integer range 0 to 10;
    variable block_number: integer range 0 to 9;
    variable index : integer range 0 to 80;
begin
    -- Blok ditentukan menggunakan i-i%3 dan j-j%3, dimana i dan j a
    block_i := ((i-1) - ((i-1) mod 3)) + 1;     -- Variabel sementara
    block_j := ((j-1) - ((j-1) mod 3)) + 1;     -- Variabel sementara
    index := (block_i)*10 + (block_j);

    case (index) is  -- Menentukan nomor blok berdasarkan informasi
        when 11 => block_number := 1;
        when 14 => block_number := 2;
        when 17 => block_number := 3;
        when 41 => block_number := 4;
        when 44 => block_number := 5;
        when 47 => block_number := 6;
        when 71 => block_number := 7;
        when 74 => block_number := 8;
        when 77 => block_number := 9;
        when others => block_number := 0;
    end case;
    return block_number;
end function;
```

Gambar 5. Function

Ketika proses dimulai, semua elemen pada bitmap baris, kolom, dan blok diatur ke nilai nol 0, yang menunjukkan bahwa belum ada angka yang valid dimasukkan. Kemudian, proses ini melakukan iterasi melalui setiap elemen pada grid. Untuk setiap elemen, nilai angka pada puzzle akan diperiksa dan, berdasarkan angka, status yang sesuai pada bitmap diupdate. Jika angka tersebut valid, maka nilai pada bitmap untuk baris, kolom, dan *block* yang sesuai akan diatur menjadi 1, yang menandakan bahwa angka tersebut telah ditempatkan di lokasi yang valid. Jika angka adalah 0, status pada bitmap tetap 0, dan jika nilai di luar rentang 1-9, statusnya akan diset ke “-”. Proses ini memastikan bahwa status validitas setiap angka yang dimasukkan ke dalam puzzle tercatat dengan benar dalam bitmap untuk digunakan dalam validasi lebih lanjut.

```
process(start, puzzle, bitmap_row, bitmap_col, bitmap_block)
variable block_number: integer range 1 to 9;
variable Test_variable : integer range 0 to 9;
begin
    bitmap_row <= (others => (others => ('0')));
    bitmap_col <= (others => (others => ('0')));
    bitmap_block <= (others => (others => ('0')));

    for i in integer range 1 to 9 loop  -- baris
        for j in integer range 1 to 9 loop  -- kolom
            Test_variable := (puzzle(i, j));  -- Memuat elemen dari puzzle d
            block_number := block_number_function(i, j);
            case Test_variable is  -- Memperbarui nomor baris, kolom, dan bl
                when 0 =>
                    bitmap_row(i, Test_variable) <= '0';
                    bitmap_col(j, Test_variable) <= '0';
                    bitmap_block(block_number, Test_variable) <= '0';
                when 1 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when 2 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when 3 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when 4 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when 5 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when 6 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when 7 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when 8 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when 9 =>
                    bitmap_row(i, Test_variable) <= '1';
                    bitmap_col(j, Test_variable) <= '1';
                    bitmap_block(block_number, Test_variable) <= '1';
                when others =>
                    bitmap_row(i, j) <= '-';
                    bitmap_col(j, Test_variable) <= '-';
                    bitmap_block(block_number, Test_variable) <= '-';
            end case;
        end loop;
    end loop;
```

Gambar . Bitmap Update

Secara garis besar, proses ini adalah implementasi algoritma penyelesaian teka-teki Sudoku. Pertama, pada state idle, program akan menyalin nilai dari "puzzle_buffer" ke dalam matriks puzzle.. Kemudian pada state next_empty_cell, program akan mencari sel kosong berikutnya dan menyimpan informasi mengenai sel tersebut, seperti baris, kolom, dan blok. Saat state guess, program akan memeriksa kemungkinan angka yang dapat diisi pada sel kosong tersebut dengan mempertimbangkan aturan Sudoku. Jika ditemukan angka yang valid, angka tersebut akan dimasukkan ke dalam matriks puzzle dan proses dilanjutkan. Jika tidak ditemukan angka yang valid, program akan kembali ke state "backtrack" untuk mencoba opsi lain. Proses ini berlanjut hingga seluruh sel terisi dengan benar, yang ditandai dengan transisi ke state solve.

```
case state_present is
    when idle =>
        i_var := conv_integer(unsigned(i));
        j_var := conv_integer(unsigned(j));
        if (i = "0000") then
            i_var := 1;
        end if;
        if (j = "0000") then
            j_var := 1;
        end if;
        puzzle(i_var, j_var) <= conv_integer(unsigned(puzzle_buffer));

    when next_empty_cell =>
        valid <= '0';
        loop1: for i in integer range 1 to 9 loop  -- baris
            loop2: for j in integer range 1 to 9 loop  -- kolom
                Test_variable := (puzzle(i, j));
                if (Test_variable = 0) then
                    selected_row <= i;
                    selected_col <= j;
                    selected_block <= block_number_function(i, j);
                    next_cell_found <= '1';
                    symbol_variable <= Test_variable;
                    exit loop1 when Test_variable = 0;
                    exit loop2 when Test_variable = 0;
                elsif (i = 9 and j = 9 and Test_variable /= 0) then
                    all_cell_filled <= '1';  -- Puzzle selesai, saatnya untuk solve
                    exit loop1 when Test_variable = 0;
                    exit loop2 when Test_variable = 0;
                else
                    selected_row <= 0;
                    selected_col <= 0;
                    selected_block <= 0;
                    next_cell_found <= '0';
                    all_cell_filled <= '0';
                    symbol_variable <= 0;
                end if;
            end loop;
        end loop;
```

```
when guess =>
    error_variable := '1';
    selected_block <= block_number_function(selected_row, selected_col);
    for j in integer range 1 to 9 loop
        guess_report(j) := bitmap_row(selected_row, j) or bitmap_col(selected_col, j) or bitmap_block(selected_block, j);
        error_variable := error_variable and guess_report(j);
    end loop;
    loop_i: for i in integer range 1 to 9 loop
        if (guess_report(i) = '0' and restored_last_valid_fill = '0') then
            stack_row(pointer) <= selected_row;
            stack_col(pointer) <= selected_col;
            pointer <= pointer - 1;
            error <= '0';
            valid <= '1';  -- Masuk ke state berikutnya
            puzzle(selected_row, selected_col) <= i;  -- Memperbarui bitmap
            update <= not update;
            exit loop_i;
        elsif (error_variable = '1' and restored_last_valid_fill = '0') then
            error <= error_variable;
            valid <= '0';
            exit loop_i;
        end if;
    end loop;

    -- Jika elemen ditolak dari backtrack, pilih elemen valid berikutnya yang lebih besar
    if (restored_last_valid_fill = '1') then
        selected_block <= block_number_function(selected_row, selected_col);
        for j in integer range 1 to 9 loop
            guess_report(j) := bitmap_row(selected_row, j) or bitmap_col(selected_col, j) or bitmap_block(selected_block, j);
            error_variable := error_variable and guess_report(j);
        end loop;
        loop_2 : for i in integer range 1 to 9 loop
            if (i > variable1 and guess_report(i) = '0') then
                stack_row(pointer) <= selected_row;
                stack_col(pointer) <= selected_col;
                pointer <= pointer - 1;
                puzzle(selected_row, selected_col) <= i;
                update <= not update;
                error <= '0';
                valid <= '1';
                restored_last_valid_fill <= '0';
                exit loop_2;
            else
                restored_last_valid_fill <= '1';
            end if;
        end loop;
    end if;
```

```
when backtrack =>
    pointer <= pointer + 1;
    selected_row <= stack_row(pointer + 1);
    selected_col <= stack_col(pointer + 1);
    selected_block <= block_number_function(stack_row(pointer + 1), stack_col(pointer + 1));
    symbol_variable <= puzzle(stack_row(pointer + 1), stack_col(pointer + 1));
    variable1 := puzzle(stack_row(pointer + 1), stack_col(pointer + 1));
    puzzle(stack_row(pointer + 1), stack_col(pointer + 1)) <= 0;
    update <= not update;
    restored_last_valid_fill <= '1';

when solve =>
    ready <= '1';
```

Gambar 6 . State Controller

```
stimulus: process
begin
    reset <= '1';
    wait for CLK_PERIOD;
    reset <= '0';

    i <= "0001"; j <= "0001"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0001"; j <= "0010"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0001"; j <= "0011"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0001"; j <= "0100"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0001"; j <= "0101"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0001"; j <= "0110"; puzzle_buffer <= "0011"; wait for CLK_PERIOD;
    i <= "0001"; j <= "0111"; puzzle_buffer <= "0010"; wait for CLK_PERIOD;
    i <= "0001"; j <= "1000"; puzzle_buffer <= "1001"; wait for CLK_PERIOD;
    i <= "0001"; j <= "1001"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;

    i <= "0010"; j <= "0001"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0010"; j <= "0010"; puzzle_buffer <= "1000"; wait for CLK_PERIOD;
    i <= "0010"; j <= "0011"; puzzle_buffer <= "0110"; wait for CLK_PERIOD;
    i <= "0010"; j <= "0100"; puzzle_buffer <= "0105"; wait for CLK_PERIOD;
    i <= "0010"; j <= "0101"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0010"; j <= "0110"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0010"; j <= "0111"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0010"; j <= "1000"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0010"; j <= "1001"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;

    i <= "0011"; j <= "0001"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0011"; j <= "0010"; puzzle_buffer <= "0010"; wait for CLK_PERIOD;
    i <= "0011"; j <= "0011"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0011"; j <= "0100"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0011"; j <= "0101"; puzzle_buffer <= "0001"; wait for CLK_PERIOD;
    i <= "0011"; j <= "0110"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0011"; j <= "0111"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0011"; j <= "1000"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
    i <= "0011"; j <= "1001"; puzzle_buffer <= "0000"; wait for CLK_PERIOD;
```

Gambar 1. Kutipan kode *sudoku_tb.vhd*

CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

Untuk menganalisis kinerja dan fungsi dari sistem Sudoku Solver yang telah dibuat, dilakukan pengujian menggunakan testbench. Testbench ini dirancang untuk menguji sistem secara menyeluruh melalui pemberian input berupa koordinat baris dan kolom (i, j) beserta nilai angka pada cell tertentu. Pengujian ini bertujuan untuk memastikan bahwa algoritma dapat

bekerja dengan benar, baik dalam menemukan cell kosong, melakukan *guessing* angka, memvalidasi angka yang dimasukkan, serta menyelesaikan teka-teki hingga kondisi akhir tercapai.

Secara garis besar, testbench terdiri dari dua proses utama, yaitu clock cycle dan pemberian input puzzle secara berurutan. Clock cycle yang dihasilkan memiliki periode 10 ns, di mana kondisi low dan high masing-masing memiliki durasi 5 ns. Hal ini memastikan bahwa semua perubahan state pada sistem Sudoku Solver berjalan secara sinkron dan sesuai dengan desain.

Proses kedua dalam testbench adalah pemberian input nilai awal pada grid Sudoku. Input ini mencakup koordinat cell serta nilai angka yang akan diisi. Pada cell kosong, nilai 0 digunakan sebagai penanda bahwa cell tersebut belum memiliki angka. Input diberikan secara berurutan dari baris pertama hingga baris kesembilan, serta kolom pertama hingga kolom kesembilan. Setelah semua input dimasukkan, sinyal start diaktifkan untuk memulai proses pemecahan teka-teki Sudoku.

Selama pengujian, testbench memantau status keluaran sistem, terutama sinyal ready, yang menunjukkan bahwa proses pemecahan telah cellesai. cellain itu, sistem juga dievaluasi berdasarkan perubahan nilai pada grid Sudoku yang terjadi cellama simulasi. Nilai-nilai yang diisi pada cell kosong akan divalidasi menggunakan aturan Sudoku, yaitu setiap angka harus unik di dalam baris, kolom, dan kotak 3x3.

Hasil simulasi ditampilkan dalam bentuk waveform untuk memudahkan analisis terhadap proses pemecahan Sudoku. Pada waveform, setiap perubahan nilai i, j, puzzle_buffer, dan kondisi state system dapat diamati secara visual. Keberhasilan sistem ditandai dengan sinyal ready yang aktif serta grid Sudoku yang terisi penuh dan memenuhi aturan Sudoku.

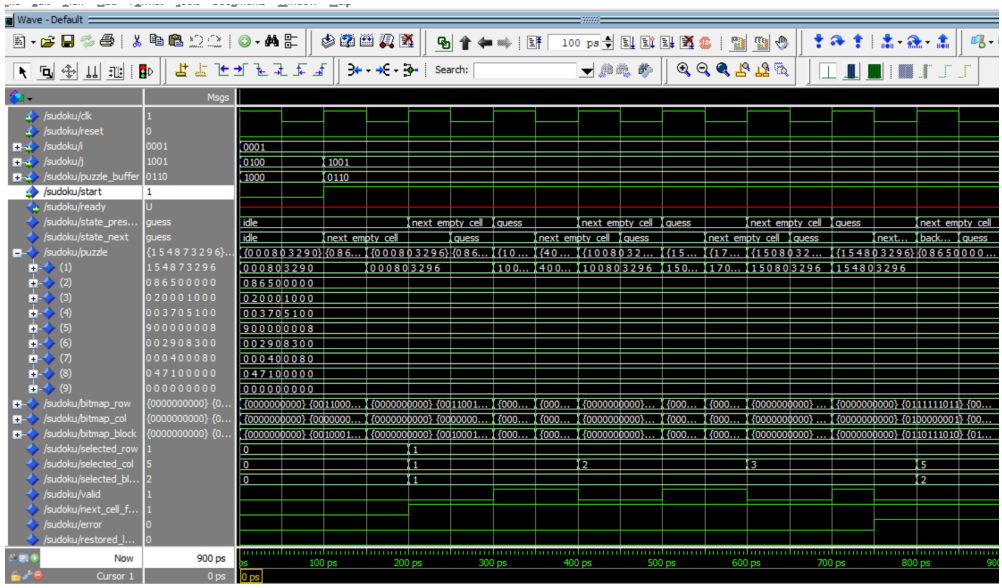
Testing juga dilakukan pada Top Level yakni sudoku.vhd, Pada tahap pertama, memulai dengan state idle, di mana proses dimulai ketika sinyal start diaktifkan. Pada tahap ini, kode memeriksa setiap *cell* dalam puzzle untuk menemukan sel yang kosong. Ketika sebuah sel kosong ditemukan, menentukan baris, kolom, dan *block* yang sesuai untuk *cell* tersebut, dan dilanjutkan untuk memilih angka yang valid untuk mengisi sel tersebut. Jika tidak ada *cell*

kosong yang ditemukan, sistem beralih ke status solve, yang menandakan bahwa puzzle telah selesai diisi.

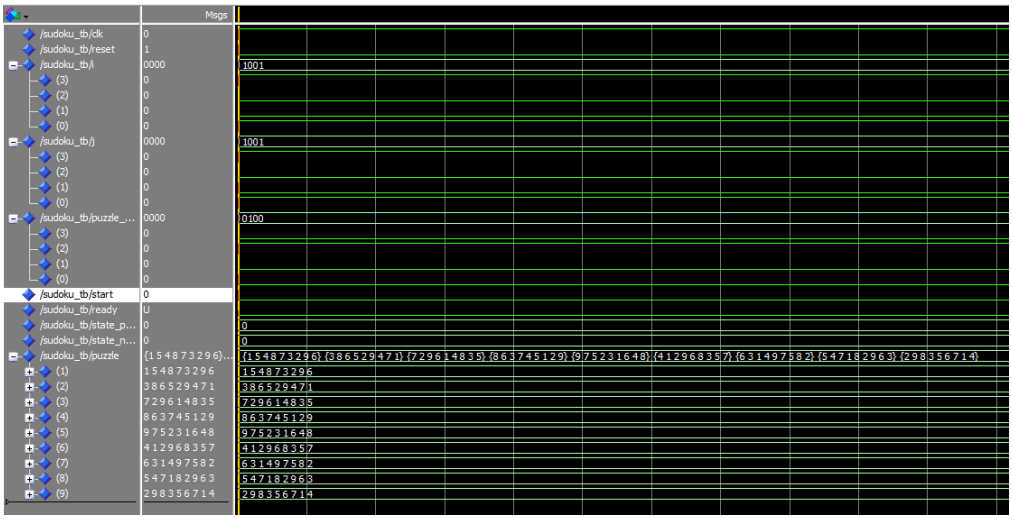
Selanjutnya, pada state guess, akan mencoba untuk menebak angka yang valid untuk *cell* yang kosong. Sebelum melakukannya, dilakuakan checking apakah angka tersebut valid berdasarkan, yaitu memastikan angka tersebut tidak sudah muncul di baris, kolom, atau blok yang sama. Jika angka yang dipilih valid, maka angka tersebut dimasukkan ke dalam puzzle, dan status akan beralih ke next_empty_cell untuk memproses *cell* berikutnya. Namun, jika tebakan tersebut tidak valid, atau jika tidak ada angka yang valid ditemukan, sistem akan beralih ke status backtrack, yang memaksa sistem untuk mundur dan mencoba pilihan lain yang lebih besar atau berbeda untuk sel yang sedang diproses.

Pada state backtrack berfungsi untuk menangani situasi di mana *guess* yang dilakukan sebelumnya salah. Sistem akan menggunakan stack untuk melacak posisi sel yang telah diproses, dan ketika kesalahan terdeteksi, sistem akan "menghapus" angka yang telah dipilih dan mencoba pilihan yang lebih tinggi. Selanjutnya, pada tahap solve, jika puzzle telah berhasil diisi dengan benar, sistem memberikan sinyal ready untuk menandakan bahwa puzzle telah selesai dan siap untuk diverifikasi atau digunakan lebih lanjut. Kode ini juga memperbarui bitmap untuk mencatat status dari setiap baris, kolom, dan blok selama proses pengisian puzzle, memastikan bahwa tidak ada angka yang duplikat dalam puzzle yang sedang diproses. Secara keseluruhan, desain ini bekerja dengan memanfaatkan metode backtracking untuk menyelesaikan puzzle Sudoku secara efisien.

3.2 RESULT



Gambar 1. Testing Start dari TopLevel sudoku.vhd pada MODELSim



Gambar 2. Testing Result dari Testbench sudoku_tb.vhd pada MODELSim

3.3 ANALYSIS

Berdasarkan hasil simulasi yang dilakukan pada top-level desain sudoku.vhd menggunakan ModelSim, dapat terlihat bahwa proses penyelesaian puzzle Sudoku berhasil dilakukan sesuai dengan arsitektur yang telah didefinisikan. Simulasi dimulai dengan

pengaktifan sinyal start (nilai 1) dan reset yang diset ke 0. Setelah itu, proses dimulai pada state idle dengan perubahan status pada rising edge dari sinyal clock.

Pada input awal, nilai-nilai i, j, dan puzzle_buffer diberikan untuk mempercepat proses guessing. Input buffer awal yakni, baris pertama kolom keempat dengan element 8 dan kolom 9 dengan element 6, saat start diaktifkan yakni dengan melakukan force value ke nilai 1. State awali dengan idle state. Setelah itu state berubah ke next_empty_cell untuk mencari cell yang masih kosong pada grid, kemudian setelah ditemukan cell kosong tersebut akan lanjut ke state guess untuk menebak angka yang ada pada koordinat tersebut.

Selanjutnya, input buffer juga dilakukan untuk elemen pada baris 8 dan kolom 5 dimulai pada nilai 8, yang kemudian menjadi titik awal untuk mengisi seluruh grid. Proses pengisian dimulai dari state next_empty_cell, yang kemudian bergerak ke state guess, di mana setiap elemen grid diuji untuk validitasnya berdasarkan aturan Sudoku baris, kolom, dan blok yang tidak saling tumpang tindih. Dalam simulasi ini, semua elemen diisi sesuai dengan urutan dan alur kerja yang telah ditentukan, mengikuti nilai-nilai yang ada dalam grid Sudoku.

Pada fase akhir, ketika proses guessing mencapai baris ke-9, kolom ke-9, dan blok ke-9, koordinat yang sedang diuji adalah elemen terakhir pada grid. Pada rising edge berikutnya, semua blok Sudoku telah terisi, dan sinyal next_cell_found berubah menjadi 0, yang menunjukkan bahwa tidak ada lagi sel kosong yang harus diisi. Kemudian, status berpindah ke state solve, menandakan bahwa seluruh puzzle telah selesai. Akhirnya, port ready berubah dari nilai 0 menjadi 1, yang menandakan bahwa Sudoku telah berhasil diselesaikan. Ini menunjukkan bahwa sistem bekerja dengan baik, dan hasilnya sesuai dengan ekspektasi.

CHAPTER 4

CONCLUSION

Pengujian menggunakan testbench menunjukkan bahwa sistem Sudoku Solver mampu bekerja sesuai dengan desain. Proses simulasi berhasil memvalidasi algoritma dalam:

1. Menemukan cell kosong, menebak angka, dan memvalidasi input berdasarkan aturan Sudoku.
2. Menyelesaikan teka-teki hingga kondisi akhir dengan grid terisi penuh.
3. Memastikan sinkronisasi sistem melalui clock cycle dengan periode 10 ns.

Hasil simulasi berupa waveform memberikan bukti visual atas keberhasilan sistem, ditandai dengan sinyal ready aktif dan grid yang memenuhi aturan unik angka pada baris, kolom, dan kotak 3x3.

Proyek ini berhasil mengembangkan Sudoku Solver berbasis VHDL yang dapat secara otomatis menyelesaikan teka-teki Sudoku berukuran 9x9 menggunakan algoritma backtracking. Sistem ini mampu mengisi grid Sudoku dengan angka yang valid berdasarkan aturan permainan, memastikan bahwa angka 1-9 tidak berulang di baris, kolom, dan kotak 3x3. Melalui implementasi algoritma backtracking, sistem dapat secara efisien mencoba berbagai kombinasi angka hingga menemukan solusi yang tepat. Pengujian menggunakan ModelSim membuktikan bahwa sistem berjalan dengan baik, dengan sinyal ready menunjukkan bahwa grid telah terisi sesuai aturan.

Dengan menggunakan FPGA dan pemrograman VHDL, proyek ini memberikan pemahaman lebih dalam tentang penerapan algoritma pencarian dan backtracking dalam desain sistem digital. Seluruh proses pemecahan teka-teki dapat dilakukan secara otomatis dan efisien, dengan hasil yang terverifikasi melalui simulasi. Hasil dari proyek ini tidak hanya berhasil menyelesaikan teka-teki Sudoku, tetapi juga memberikan kontribusi pada pengembangan perangkat keras dan pemrograman logika, serta dapat dijadikan referensi dalam pengembangan sistem serupa di masa depan.

REFERENCES

[1] "Dataflow Style Programming in VHDL," Learn.Digilabdte. [Online]. Available: https://learn.digilabdte.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.digilabdte.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.digilabdte.com/books/digital-system-design/page/vhdl-modularity. [Accessed: Dec. 7, 2024].

[4] Digilab DTE, "Array and Types in VHDL," [Online]. Available: https://learn.digilabdte.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].

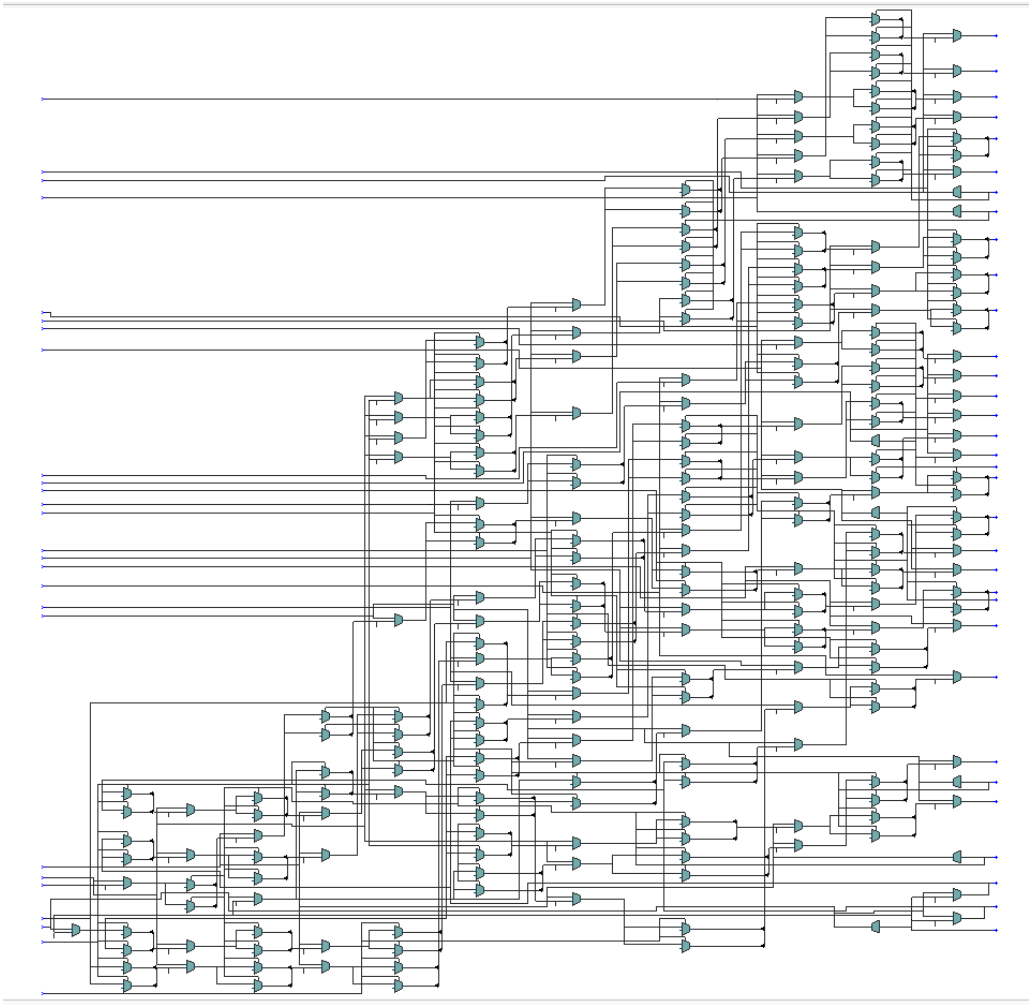
[6] FPGA Insights, "Mastering Loops in VHDL: Enhancing Flexibility and Performance in Hardware Design," [Online]. Available: https://fpgainsights.com/fpga/mastering-loops-in-vhdl-enhancing-flexibility-and-performance-in-hardware-design/#:~:text=FOR%20Loop%20in%20VHDL,-The%20FOR%20loop&text=It%20allows%20you%20to%20specify,each%20value%20within%20that%20range.&text=end%20loop%3B,are%20executed%20for%20each%20iteration. [Accessed: Dec. 7, 2024].

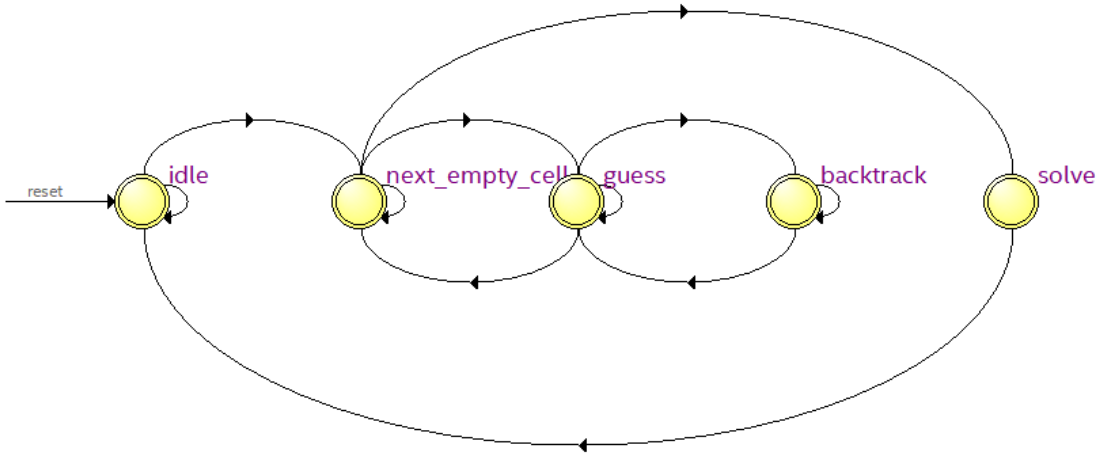
[7] DigiLabDTE, "FSM Implementation Example in VHDL," Learn DigiLabDTE, [Online]. Available: https://learn.digilabdte.com/books/digital-system-design/page/fsm-implementation-example-in-vhdl [Accessed: Dec. 7, 2024].

[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

Appendix A: Project Schematic





Appendix B: Documentation