Project Overview

Title: 21×21 Keyboard Matrix Controller using Verilog

- A keyboard controller for a 21×21 matrix keypad (441 keys)
- Scans rows and columns to detect pressed keys
- Uses Verilog HDL for FPGA implementation or simulation
- Built by expanding the concept of a 3×3 keypad controller

How a Matrix Keypad Works

- Keys arranged in rows and columns
- Columns are inputs with pull-up resistors (idle HIGH)
- Rows are outputs activated one at a time
- Pressing a key pulls the corresponding column LOW
- Active row + LOW column = key pressed

Scanning Method

1. All rows are HIGH initially
2. One row is pulled LOW
3. All columns are read
4. If any column reads LOW → key in that row is pressed
5. Move to next row and repeat for all 21 rows
6. Continuous scanning detects keys in real time

Debouncing Using Counter

- Mechanical switches cause bouncing (unstable LOW/HIGH)
- Counter delays row switching to stabilize readings
- Prevents false key detection
- Ensures clean and reliable key press capture

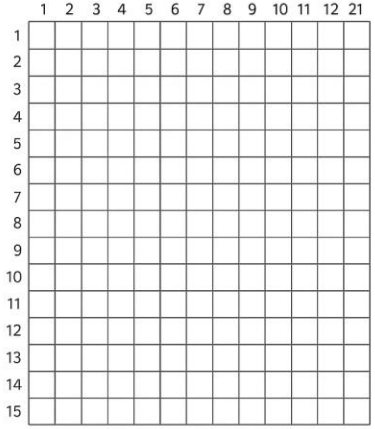
Visual Diagram

21×21 Keyboard Matrix Controller
using Verilog

- A keyboard controller for a 21×21 matrix keypad (total 441 keys)
- Scans rows and columns to detect pressed keys
- Uses Verilog HDL, suitable for FPGA simulation or implementation

Purpose

- Reduce I/O pins for large keypads
- Efficiently detect key presses in real time
- Built by extending the idea of a 3×3 keypad controller



Design Components

Inputs:
- clk
- col[20:0]

Outputs:
- row[20:0]
- key_state[440:0]

Registers:
- scan_row – selects which row is active
- counter – timing control and debouncing

Verilog Logic Flow

always @(posedge clk):

- Increment counter
- When counter hits limit → scan next row
- case(scan_row) sets active row LOW
- case(scan_row) reads column values
- Store values in key_state
- Repeat for all 21 rows

Full Verilog Code

```
module keyboard21x21 (
input clk,
input [20:0] col,
output reg [20:0] row,
output reg [440:0] key_state
);
reg [4:0] scan_row = 0;
reg [15:0] counter = 0;
always @(posedge clk) begin
counter <= counter + 1;
if (counter == 50000) begin
counter <= 0;
row <= 21'b111111111111111111111;
case (scan_row)
5'd0: row <= 21'b111111111111111111110;
5'd1: row <= 21'b111111111111111111101;
5'd2: row <= 21'b111111111111111111011;
5'd3: row <= 21'b111111111111111110111;
5'd4: row <= 21'b111111111111111101111;
5'd5: row <= 21'b111111111111111011111;
5'd6: row <= 21'b111111111111110111111;
5'd7: row <= 21'b111111111111101111111;
```

Full Verilog Code

```
5'd8: row <= 21'b111111111111011111111;
5'd9: row <= 21'b111111111110111111111;
5'd10: row <= 21'b111111111101111111111;
5'd11: row <= 21'b111111111011111111111;
5'd12: row <= 21'b111111110111111111111;
5'd13: row <= 21'b111111101111111111111;
5'd14: row <= 21'b111111011111111111111;
5'd15: row <= 21'b111110111111111111111;
5'd16: row <= 21'b111101111111111111111;
5'd17: row <= 21'b111011111111111111111;
5'd18: row <= 21'b110111111111111111111;
5'd19: row <= 21'b101111111111111111111;
5'd20: row <= 21'b011111111111111111111;
endcase
case (scan_row)
5'd0: key_state[20:0] <= ~col;
5'd1: key_state[41:21] <= ~col;
5'd2: key_state[62:42] <= ~col;
5'd3: key_state[83:63] <= ~col;
5'd4: key_state[104:84] <= ~col;
5'd5: key_state[125:105] <= ~col;
5'd6: key_state[146:126] <= ~col;
```

Full Verilog Code

```
5'd7: key_state[167:147] <= ~col;
5'd8: key_state[188:168] <= ~col;
5'd9: key_state[209:189] <= ~col;
5'd10: key_state[230:210] <= ~col;
5'd11: key_state[251:231] <= ~col;
5'd12: key_state[272:252] <= ~col;
5'd13: key_state[293:273] <= ~col;
5'd14: key_state[314:294] <= ~col;
5'd15: key_state[335:315] <= ~col;
5'd16: key_state[356:336] <= ~col;
5'd17: key_state[377:357] <= ~col;
5'd18: key_state[398:378] <= ~col;
5'd19: key_state[419:399] <= ~col;
5'd20: key_state[440:420] <= ~col;
endcase
scan_row <= (scan_row == 20) ? 0 : scan_row + 1;
end
end
endmodule
```