Keypad Scanner

1. Project Requirements

I want to create a scanner for a keypad with three columns and four rows as in the figure below.

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

4x3 Membrane Matrix Keypad Module

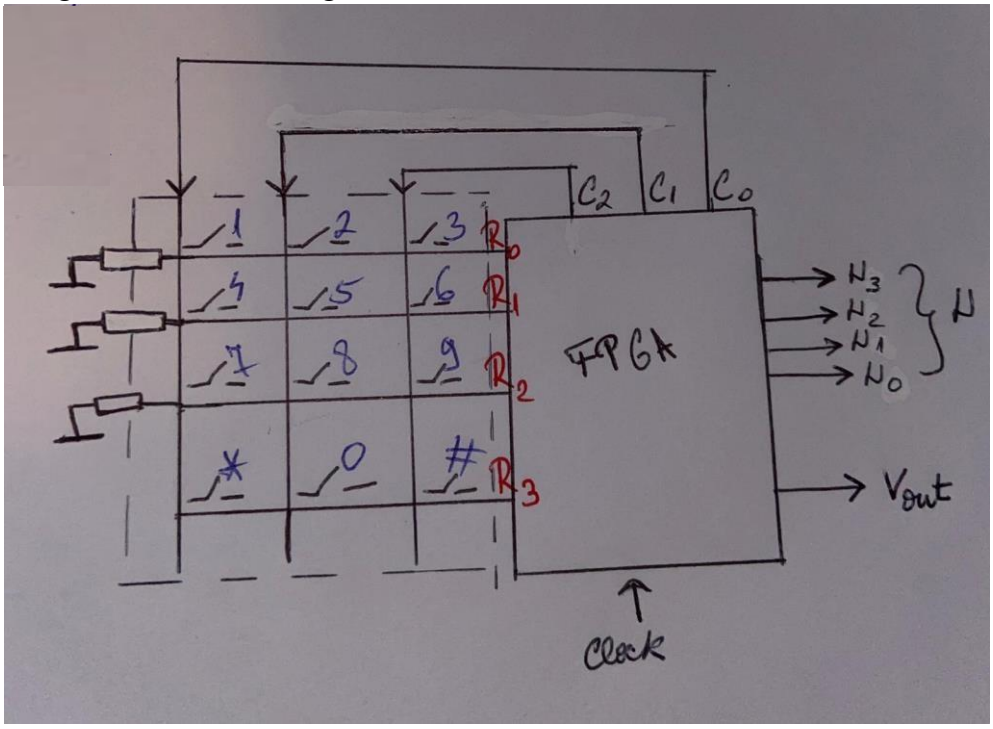
2. Theoretical support

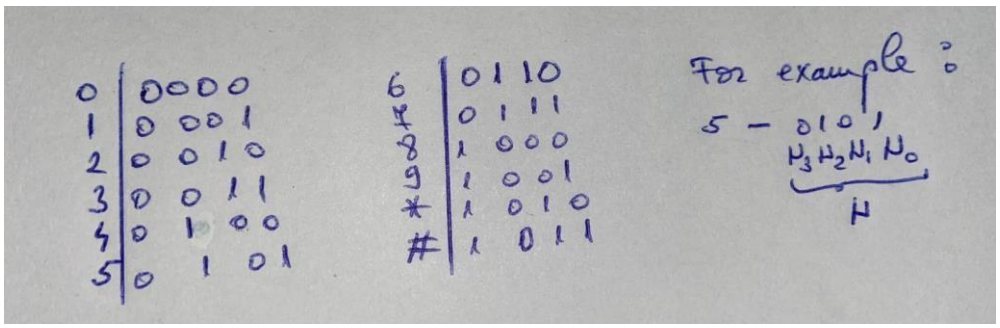
The keypad is wired in matrix form with a switch at the intersection of each row and column. Pressing a key establishes a connection between a row and column.

The purpose of the scanner is to determine which key has been pressed and output a binary number $N = N_3N_2N_1N_0$, which corresponds to the key number. For example, pressing key 0 must output 0000, pressing key 5 must output 0101, pressing the * key must output 1010, and pressing the # key must output 1011.

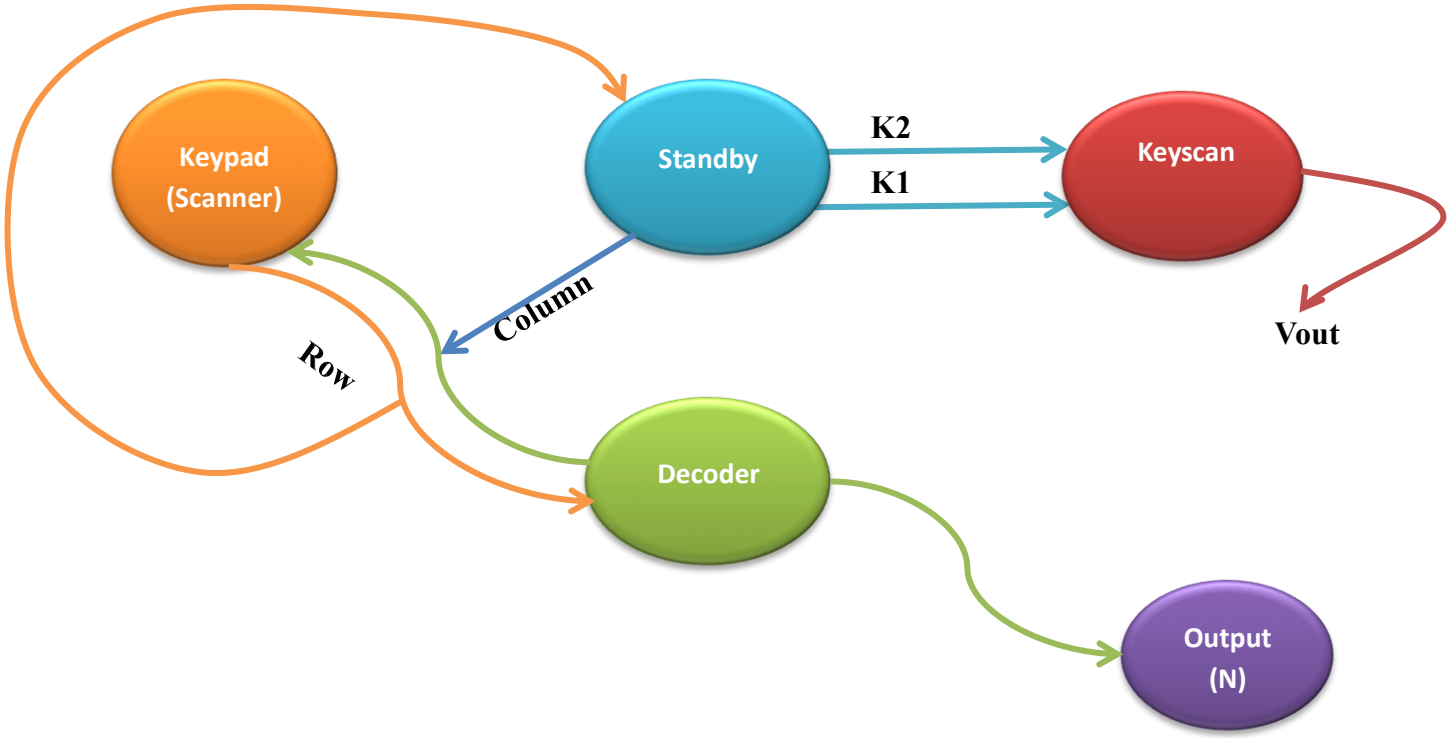
When a valid key has been detected, the scanner should output a signal Vout for one clock time. I assume that only one key is pressed at a time.

The overall block diagram of the circuit is presented in the figure below. The keypad contains resistors that are connected to ground. When a switch is pressed, a path is established from the corresponding column line to the ground.





The block diagram for the FPGA is the next one. The first block will be a **scanner** that scans the rows and columns of the keypad. The **keyscan** block generates the column signals to scan the keypad. The **standby** block generates a signal K1 when a key has been pressed and a signal K2 after it has been debounced (K2 is K1 delayed by two clock cycles). When a valid key is detected, the **decoder** determines the key number from the row and column numbers. (*The standby block can be viewed such as a debounce, which is a function that forces the device to wait a certain amount of time before running again).



For scanning the Keypad:

First is applied logic '1' to columns C0, C1 and C2 and wait. If any key is pressed, a 1 will appear on R0, R1, R2 or R3. Then is applied a 1 to column C0 only. If any of the R's is '1', a valid key is detected. If R0 is received, it is known that switch 1 was pressed. If R1, R2 or R3 is received, switch 4, 7, or * was pressed. If so, set Vout = '1' and output the corresponding N. If no key is detected in the first column, is applied a '1' to C1 and repeat. If no key is detected in the second column, repeat for C2. When a valid key is detected, apply 1's to C0, C1 and C2 and wait until no key is pressed. This last step is necessary so that only one valid signal is generated each time a key is pressed.

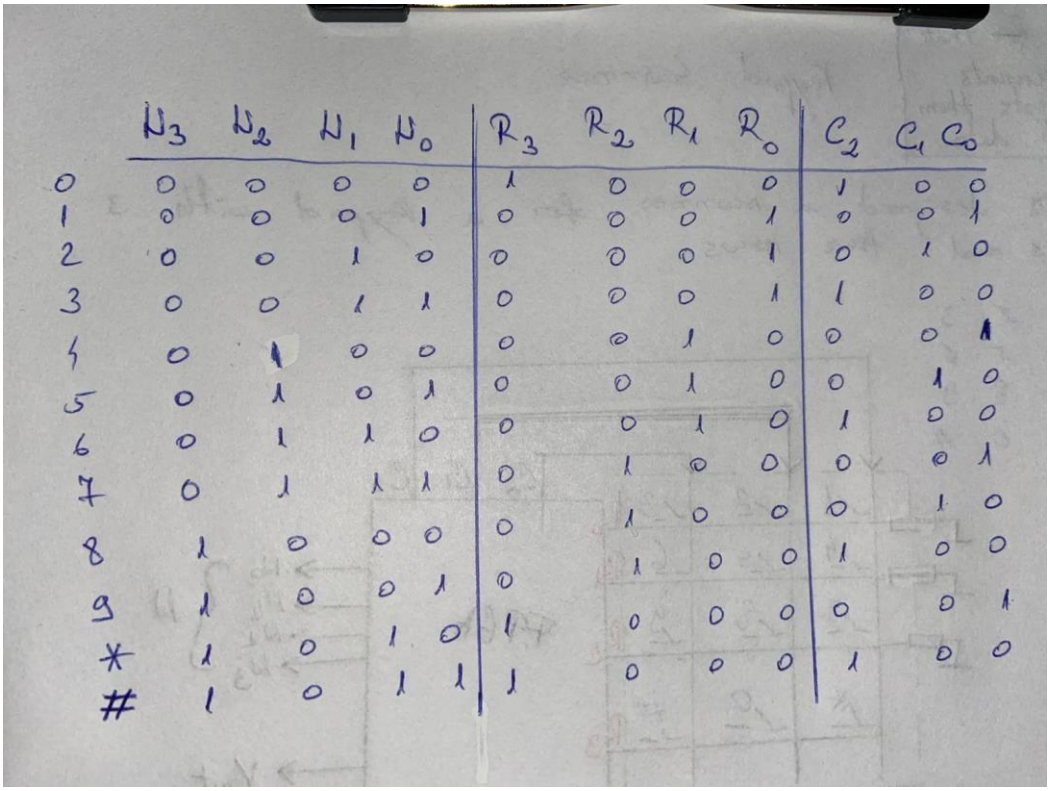
For Standby block:

It is needed to put in "standby" the keys to avoid malfunctions due to switch bounce. The four row signals are connected to an OR gate to from signal K1, which turns on when a key is pressed and a column scan signal is applied. The debounced signal K2 will be fed to the sequential circuit.

For the Decoder block:

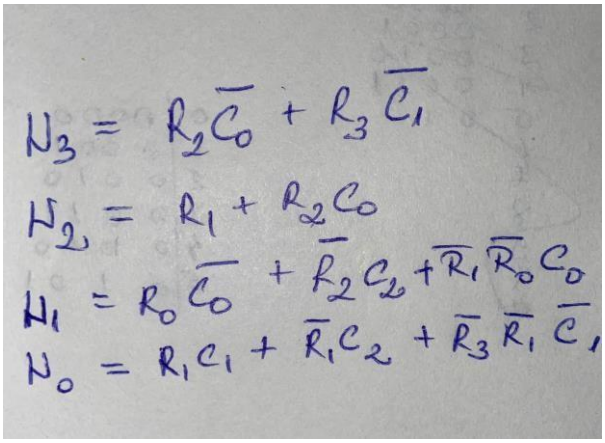
The decoder determines the key number from the row and column numbers using the truth table below. The truth table has one row for each of the 12 keys. The remaining rows have don't care outputs since it is assumed that only one key is pressed at a time. Since the decoder is a combinational circuit (*a combinational logic circuit performs an operation assigned logically by a Boolean expression or truth table), its output will change as the keypad is scanned. At the time a

valid key is detected (K1 = ‘1’ and Vout = ‘1’), its output will have the correct value and this value can be saved in a register.



	N_3	N_2	N_1	N_0	R_3	R_2	R_1	R_0	C_2	C_1	C_0
0	0	0	0	0	1	0	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0	0	1
2	0	0	1	0	0	0	0	1	0	1	0
3	0	0	1	1	0	0	0	1	1	0	0
4	0	1	0	0	0	0	1	0	0	0	1
5	0	1	0	1	0	0	1	0	0	1	0
6	0	1	1	0	0	0	1	0	1	0	0
7	0	1	1	1	0	1	0	0	0	0	1
8	1	0	0	0	0	1	0	0	0	1	0
9	1	0	0	1	0	1	0	0	1	0	0
*	1	0	1	0	1	0	0	0	0	0	1
#	1	0	1	1	1	0	0	0	1	0	0

I used an online computer (Karnaugh map) to determine the functions N3, N2, N1 and N0 (logic equations for decoder):



$$N_3 = R_2\overline{C_0} + R_3\overline{C_1}$$

$$N_2 = R_1 + R_2C_0$$

$$N_1 = R_0\overline{C_0} + \overline{R_2}C_2 + \overline{R_1}\,\overline{R_0}C_0$$

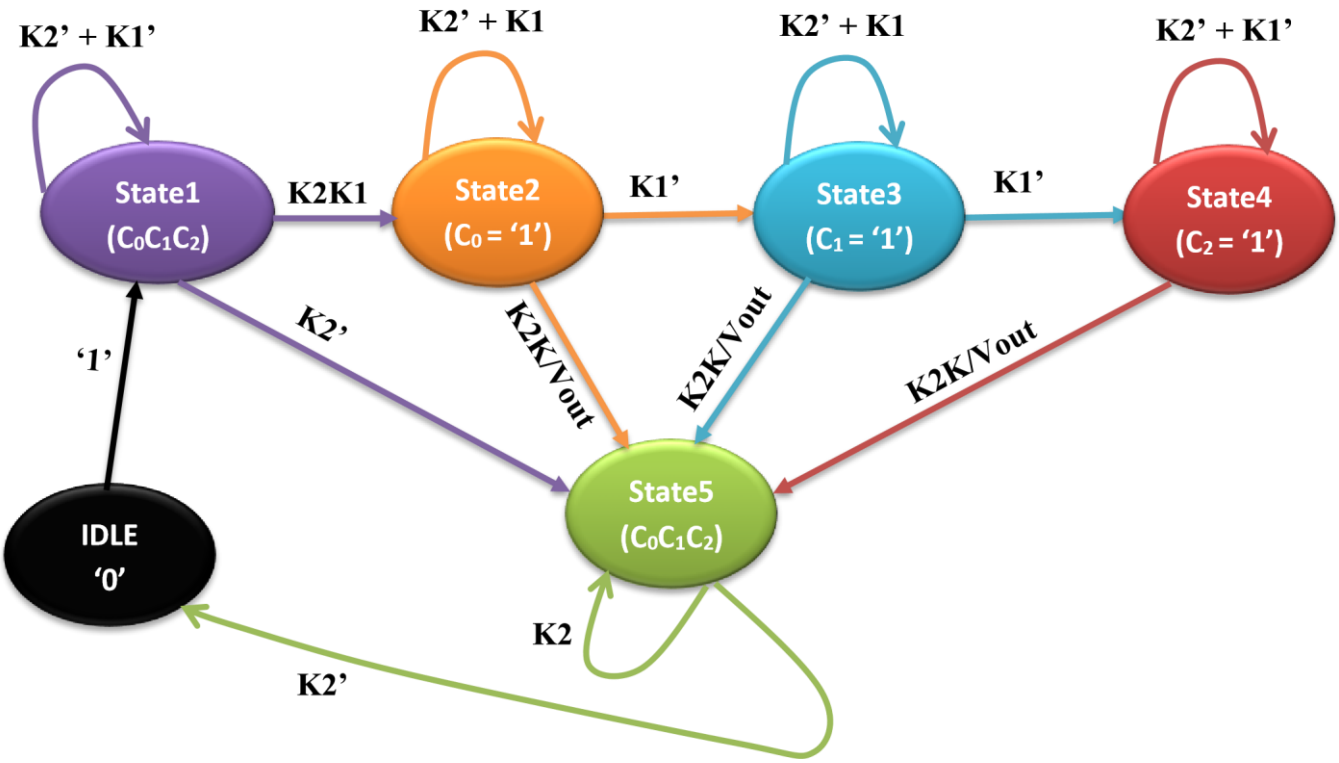
$$N_0 = R_1C_1 + \overline{R_1}C_2 + \overline{R_3}\,\overline{R_1}\,\overline{C_1}$$

For the KeyScan (Controller) block:

Below is the state diagram of the controller for the keypad scanner. It waits in State1 with outputs C0 = C1 = C2 = ’1’ until a key is pressed. In State2, C0 = ’1’, so if the key that was pressed is in column 0, K1 = ’1’, and the circuit outputs a valid signal and goes to State5. If no key press is found in column 0, column 1 is checked in State3 and if necessary, column 2 is checked in State4. In State5, the circuit waits until all keys are released and K2 goes to 0 before resetting.

I will assume that State5 can be reach only if K2 is true (if two clock cycles happened), to be sure that multiple button are not pressed. The key should be pressed long enough for the scanner to go

through the longest path in the state graph from State0 to State5. This may not be a serious problem because usually the digital system clock is much faster than any mechanical switch.



Before transitioning to State5, this circuit waits in state State2, State3 and State4 until K2 also becomes '1'.

3. Code from Vivado

Scanner (keypad)

```
entity Keypad is
    Port ( clk : in BIT;
        R0 : in BIT; -- Ri's are rows
        R1 : in BIT;
        R2 : in BIT;
        R3 : in BIT;
        C0 : inout BIT; -- Ci's are columns
        C1 : inout BIT;
        C2 : inout BIT;
        N0 : out BIT; -- Ni's are binary numbers which forms N
        N1 : out BIT;
        N2 : out BIT;
        N3 : out BIT;
        Vout : out BIT); -- when a valid key has been detected, the scanner should output a signal Vout
for one clock time.
end Keypad;
```

```
architecture Behavior of Keypad is

      -- signals declarations
signal aux, K1, K2: bit;
      -- K1 is the signal generated when a key has been pressed
      -- K2 is K1 delayed by two clock cycles (K2 is used to avoid malfunctions due to switch bounce)
signal current_state, next_state: integer range 0 to 5; -- I have 6 states in my block diagram

begin

   process(clk)
begin
      if clk = '1' and clk'EVENT then
current_state      <=      next_state;
aux <= K1;          K2 <= aux;       end
if;
   end process;

   K1 <= R0 or R1 or R2 or R3; -- this is the decoder block
   N3 <= (R2 and not C0) or (R3 and not C1);
   N2 <= R1 or (R2 and C0);
   N1 <= (R0 and not C0) or (not R2 and C2) or (not R1 and not R0 and C0);
   N0 <= (R1 and C1) or (not R1 and C2) or (not R3 and not R1 and not C1);

   process(current_state, R0, R1, R2, R3, C0, C1, C2, K1, K2, aux)
begin
         -- the Keyscan (controller) block
         -- State0 (IDLE)
         C0 <= '0';
         C1 <= '0';
         C2 <= '0';
         Vout <= '0';

         case current_state is
when 0 => next_state <= 1;

            -- State1
            when 1 => C0 <= '1';
                  C1 <= '1';
C2 <= '1';              if (K2 and K1)
= '1' then
next_state <= 2;              else
next_state <= 1;              end if;

            -- State2
when 2 => C0 <= '1';              if
```

```
(K2 and K1) = '1' then
Vout <= '1';
next_state <= 5;            elsif
K1 = '0' then
next_state <= 3;            else
next_state <= 2;            end if;

            -- State3
when 3 => C1 <= '1';            if
(K2 and K1) = '1' then
Vout <= '1';
next_state <= 5;            elsif
K1 = '0' then
next_state <= 4;            else
next_state <= 3;
            end if;

            -- State4
when 4 => C2 <= '1';            if
(K2 and K1) = '1' then
Vout <= '1';
next_state <= 5;            else
next_state <= 4;            end if;

            -- State5
when 5 => C0 <= '1';
                    C1 <= '1';
C2 <= '1';              if K2 =
'0' then
next_state <= 1;
else                  next_state
<= 5;               end if;

          end case;

   end process; end
Behavior;
```

Scantest (sim_keypad)

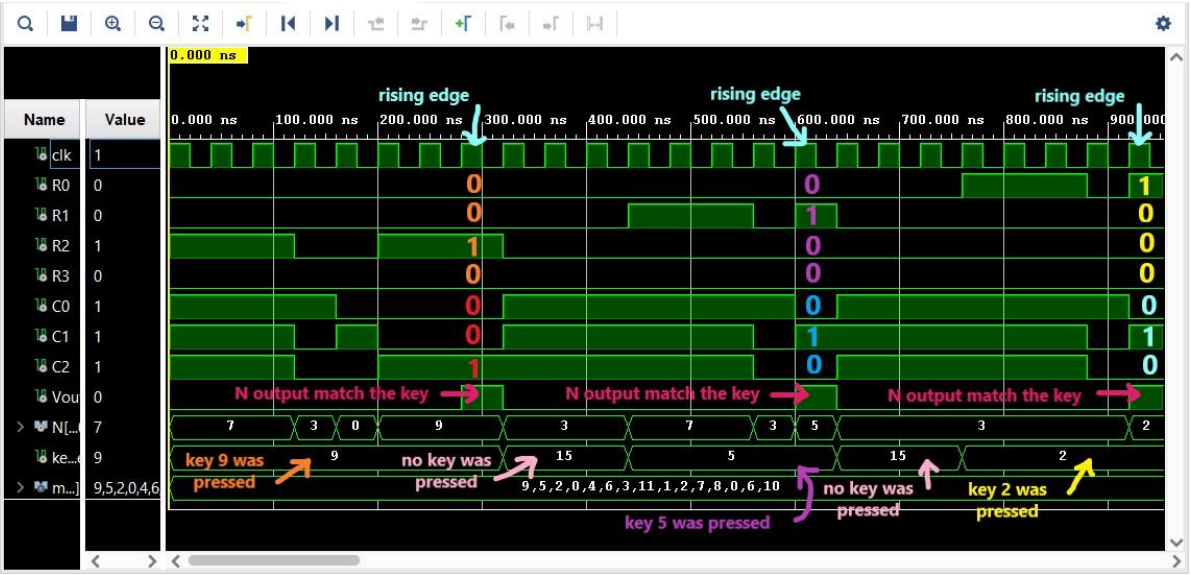
```
entity sim_keypad is
--  Port ( );
end sim_keypad;

architecture Behavioral of sim_keypad is
```

```
component Keypad is
      Port ( clk : in BIT;
           R0 : in BIT;
           R1 : in BIT;
           R2 : in BIT;
           R3 : in BIT;
           C0 : inout BIT;
           C1 : inout BIT;
           C2 : inout BIT;
           N0 : out BIT;
           N1 : out BIT;


           N2 : out BIT;
N3 : out BIT;
           Vout : out BIT);
end component Keypad;

type arr is array (0 to 14) of integer; -- array of keys to test constant my_array: arr :=
(9,5,2,0,4,6,3,11,1,2,7,8,0,6,10); signal clk, R0, R1, R2, R3, C0, C1, C2, Vout, N0, N1, N2, N3: bit; --
interface signals signal N: unsigned(3 downto 0); -- N = N3N2N1N0 (4 binary numbers) --a signal that
is defined as type unsigned means that the signal will be only positive signal key_number: integer; --
key number to test

begin

   uut: Keypad port map(clk => clk,
                  R0 => R0,
                  R1 => R1,
                  R2 => R2,
                  R3 => R3,
                  C0 => C0,
                  C1 => C1,
                  C2 => C2,
                  N0 => N(0),
                  N1 => N(1),
                  N2 => N(2),
                  N3 => N(3),
                  Vout => Vout);

   process -- generate clock signal
begin       clk <= '1'; wait for 20
ns;       clk <= '0'; wait for 20 ns;
end process;

   -- this section imitate/match the keypad
```

```
   R0 <= '1' when (C0 = '1' and key_number = 1) or (C1 = '1' and key_number = 2) or (C2 = '1' and
key_number = 3)
   else '0';
   R1 <= '1' when (C0 = '1' and key_number = 4) or (C1 = '1' and key_number = 5) or (C2 = '1' and
key_number = 6)
   else '0';
   R2 <= '1' when (C0 = '1' and key_number = 7) or (C1 = '1' and key_number = 8) or (C2 = '1' and
key_number = 9)
   else '0';


R3 <= '1' when (C0 = '1' and key_number = 10) or (C1 = '1' and key_number = 0) or (C2 = '1' and
key_number = 11)
   else '0';

   process -- this section tests scanner     begin        for i
in 0 to 14 loop -- test every number in key array
key_number <= my_array(i); -- simulates keypress
wait until (Vout = '1' and rising_edge(clk));
         -- if to_integer(N) = key_number) -- check if output matches
         -- number match (Vout = '1')
         -- else numbers don't match

         key_number <= 15; -- equivalent to no key pressed
wait until rising_edge(clk); -- wait for scanner to reset
wait until rising_edge(clk);           wait until rising_edge(clk);

      end loop;
      -- test complete

   end process; end
Behavioral;
```

4. Simulation

1. read a key number from the array to simulate pressing a key;
2. wait until Vout = '1' and the rising edge of the clock occurs;
3. verifify that the N output from the scanner matches the key number;
4. key_number is set 15 in each loop to simulate no key pressed (15 is not a valid key number, here all R's will be 0;
5. wait until K2 = 0 before selecting a new key.



5. Bibliography

1. https://learn.adafruit.com/matrix-keypad
2. https://www.geeksforgeeks.org/telephone-keypad-scanner/
3. https://www.youtube.com/watch?v=6WdycpGuhRY
4. http://www.csitsun.pub.ro/courses/Masterat/Xilinx%20Synthesis%20Technology/toolbox.xilinx.com/docsan/xilinx4/data/docs/xst/vhdl3.html