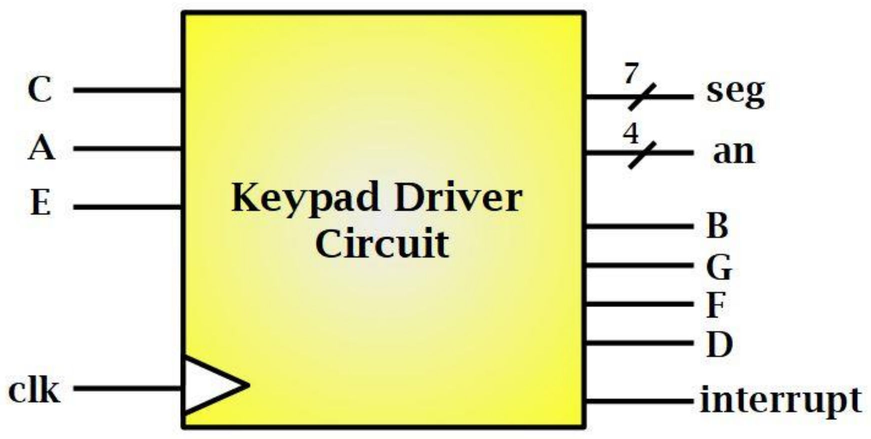
CPE 233: Peripheral assignment 3, Keypad Driver

Prof. Bridget Benson

Marina Dushenko, Luis Gomez

BBD



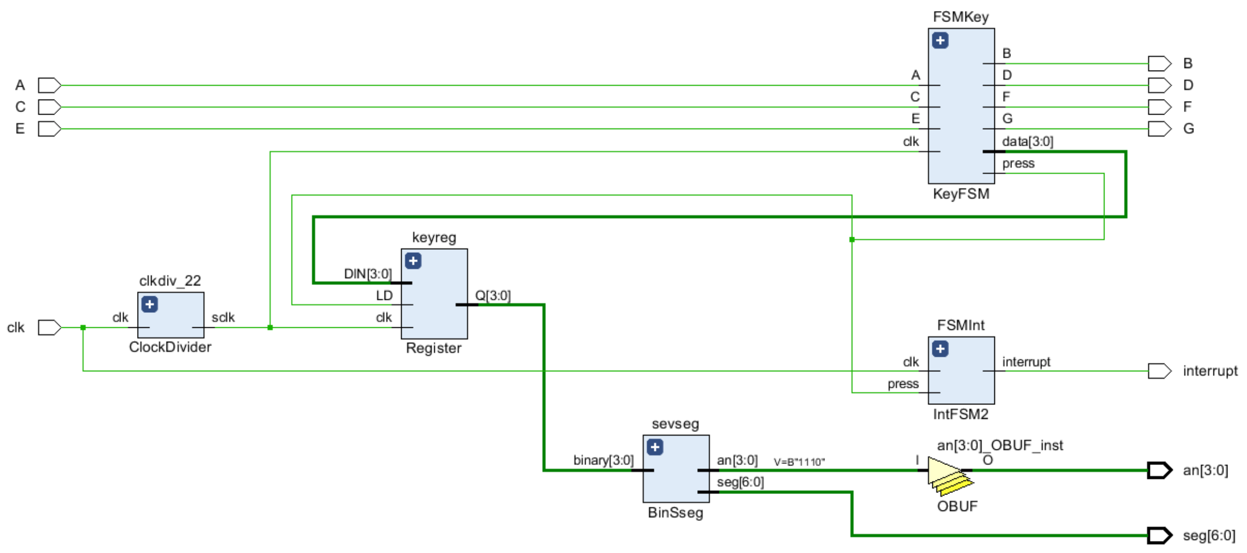
Behavior

The **Keypad Driver** is a sequential circuit whose inputs are 1-bit signals from a 'key-press' on the keypad circuit, and whose output consists of a four 1-bit signals driving the keypad-columns, 11-bits driving a seven-segment display, and 1-bit 60ns interrupt pulse.

The Keypad Driver allows a user to interact with an external keypad circuit, reading from a 3 X 4 matrix of push-buttons. The user's button-press is registered and displayed on a seven-segment display for the duration of the key-press; releasing the button will clear the display digit.

Structural

The structural design of the Keypad Driver is found below:

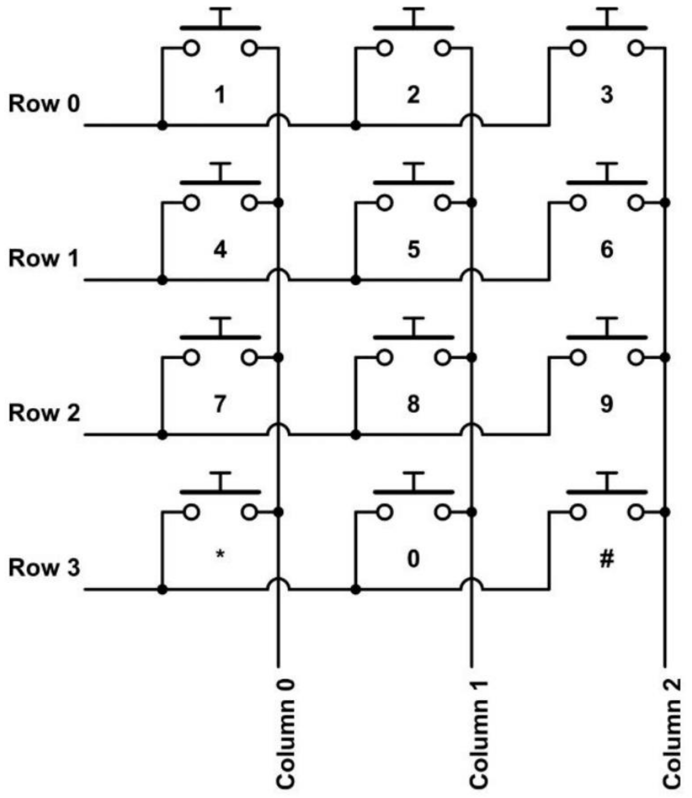


Specifications

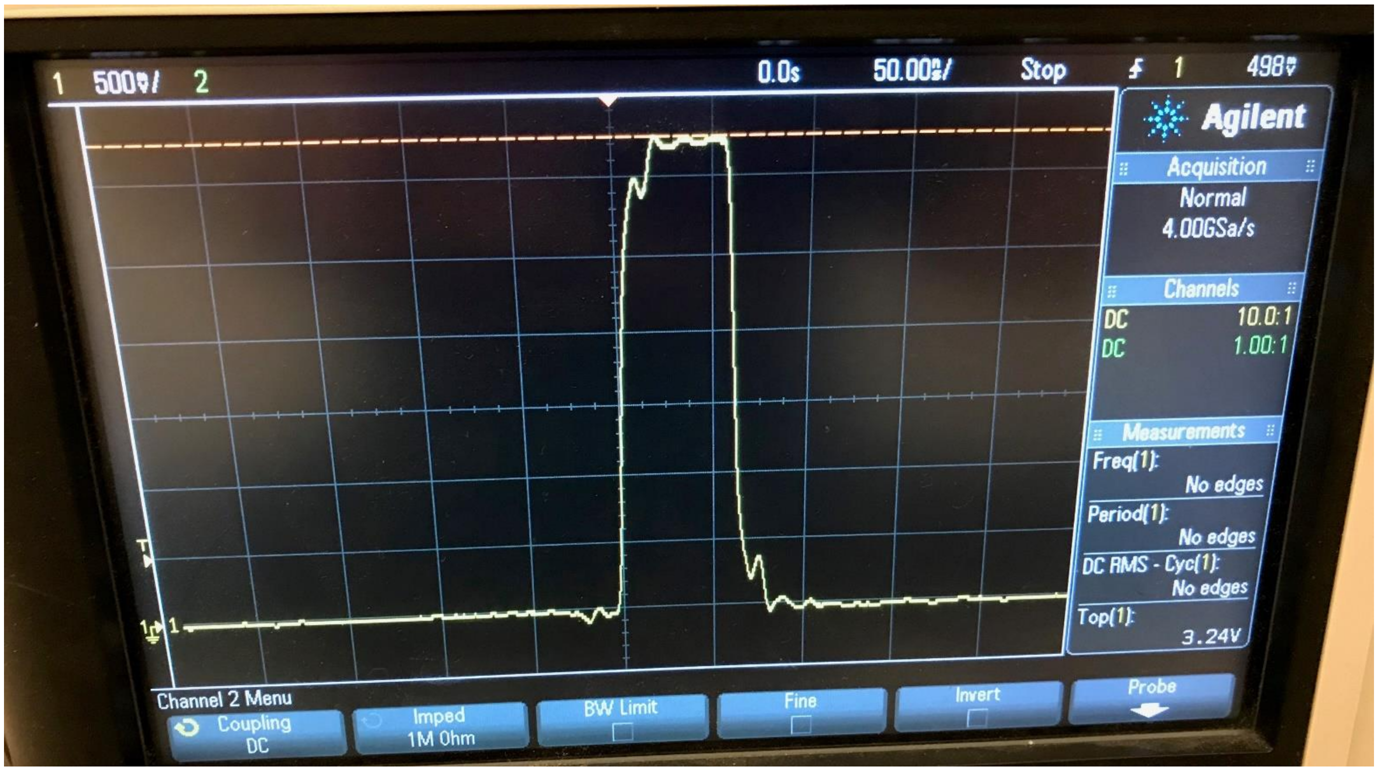
Operating Speed: 22Hz w/ Button Signal Debounced	**Supports 3 X 4 Keypads**
Input Clock: 100Mhz	**Output:** 3.3 V 60ns Interrupt pulse
Output Characters: 0-9, 'A', 'b' to a Seven-Segment Display	

The Key-Pad driver consists of various modules:

- **Key-Press FSM:** Finite state machine which drives high signal through a keypad column each cycle (22 hz). Four states are State B, State G, State F, State D. During each state, the FSM reads in C, A , E inputs and drives the corresponding key-value to the 4-bit Register.
- **Interrupt FSM:** Finite state machine which drives the 60ns Interrupt signal. The FSM has seven states 0-7, during each state the FSM checks if a debounced-button press has been input. When a debounced button-press is detected, the FSM drives the Interrupt signal high for that cycle (10ns). Over the span of 6 additional states, the cumulative period of the interrupt signal amounts to 60ns.
- **Seven-Segment Display Driver:** Converts a 4-bit binary input into a Seven-Segment Display Digit.
- **Clock-Divider:** Generates the 22 hz 'slow' clock
- **4-bit Register:** 4-bit register with a LD input



Verification



Interrupt Signal as seen by the Oscilloscope

Example Assembly Code

```
.EQU KEYPAD_PORT = 0x82
.CSEG
.ORG 0x1
; Code Demonstrating use of the Key Pad Driver
; 8-bit counter with LD

         SEI
start:   MOV R0, 0x00
count:   ADD R0, 0x01
         CMP R0, 0xFF
         BRNE count
         BRN start
.CSEG
.ORG 0x3FF
ISR: IN R0, KEYPAD_PORT
         RETIE
```

System Verilog Source Code

Key-Press FSM (3pgs)

```
`timescale 1ns / 1ps

module KeyFSM(input clk, C, A, E,
              output logic B, G, F, D, press,
              output logic [3:0] data
              );

      typedef enum {STATE_B, STATE_G, STATE_F, STATE_D} STATE;
      STATE NS, PS = STATE_B;

      always_ff @ (posedge clk)
         begin
           PS <= NS;
         end

      always_comb
      begin
      case(PS)
      STATE_B:
        begin
          if(C == 1)
            begin
              data = 1;
              press = 1;
            end
          else if(A == 1)
            begin
              data = 2;
              press = 1;
            end
          else if(E == 1)
            begin
              data = 3;
              press = 1;
            end
          else
            begin
              data = 13;
              press = 0;
            end
        begin
        B = 1; G =0; F = 0; D = 0;
        NS = STATE_G;
        end
        end
```

```
STATE_G:
        begin
          if(C == 1)
            begin
              data = 4;
              press = 1;
            end
          else if(A == 1)
            begin
              data = 5;
              press = 1;
            end
          else if(E == 1)
            begin
              data = 6;
              press = 1;
            end
          else
            begin
              data = 12;
              press = 0;
            end
          begin
            B = 0; G = 1; F = 0; D = 0;
            NS = STATE_F;
          end
        end
      STATE_F:
        begin
          if(C == 1)
            begin
              data = 7;
              press = 1;
            end
          else if(A == 1)
            begin
              data = 8;
              press = 1;
            end
          else if(E == 1)
            begin
              data = 9;
              press = 1;
            end
          else
            begin
               data = 14;
               press = 0;
            end
          begin
            B = 0; G = 0; F = 1; D = 0;
            NS = STATE_D;
          end
        end
```

```
STATE_D:
        begin
          if(C == 1)
            begin
              data = 10;
              press = 1;
            end
          else if(A == 1)
            begin
              data = 0;
              press = 1;
            end
          else if(E == 1)
            begin
              data = 11;
              press = 1;
            end
          else
            begin
              data = 15;
              press = 0;
            end
          begin
            B = 0; G = 0; F = 0; D = 1;
            NS = STATE_B;
          end
        end
      default: NS = STATE_B;
      endcase
      end

endmodule
```

Interrupt FSM (2pgs)

```
`timescale 1ns / 1ps

module IntFSM2(
    input clk, press,
    output logic interrupt
             );

    typedef enum {ST0, ST1, ST2, ST3, ST4, ST5, ST6, ST7} STATE;
    STATE NS, PS = ST0;

     always_ff @ (posedge clk)
       begin
           PS <= NS;
       end

     always_comb
        begin
        interrupt = 0 ;
         case(PS)
          ST0:
            begin
              if (press == 0)
              begin
                interrupt = 0;
                NS = ST0;
              end
              else
                NS = ST1;
            end
          ST1:
            begin
              if(press == 1)
                interrupt = 1;
                NS = ST2;
            end
          ST2:
            begin
              if(press == 1)
                interrupt = 1;
                NS = ST3;
            end
          ST3:
            begin
              if(press == 1)
                interrupt = 1;
                NS = ST4;
                 end
         ST4:
            begin
              if(press == 1)
                interrupt = 1;
                NS = ST5;
            end
          ST5:
            begin
              if(press == 1)
                interrupt = 1;
                NS = ST6;
            end
          ST6:
            begin
             if(press == 1)
               interrupt = 1;
               NS = ST7;
            end
          ST7:
            begin
              if(press == 1)
              begin
                interrupt = 0;
                NS = ST7;
              end
              else
                NS = ST0;
            end
            default: NS = ST0;
            endcase
        end

     endmodule
```

4-bit Register

```
module Register(input clk, LD,
                input [3:0] DIN,
                output logic [3:0] Q
                );

    always_ff @ (posedge clk)
        begin
            case(LD)
            1: Q <= DIN;
            0: Q <= Q;
            endcase
        end

     endmodule
```

Clock Divider

```
module ClockDivider(
    input clk,
    output logic sclk);

  localparam MAX_COUNT = 2272727;
  logic [26:0] count = 0;

   always_ff @ (posedge clk)
   begin
        count <= count + 1;
         if (count == MAX_COUNT)
         begin
            count <= 0;
            sclk <= ~sclk;
         end
   end
    endmodule
```

Seven Segment Display Driver

```
module BinSseg(
    input [3:0] binary,
    output logic [6:0] seg,
     output [3:0] an
    );

    //always block for converting binary into 7 segment format
    always_comb
    begin
        case (binary) //case statement
            0 : seg = 7'b1000000;
            1 : seg = 7'b1111001;
            2 : seg = 7'b0100100;
            3 : seg = 7'b0110000;
            4 : seg = 7'b0011001;
            5 : seg = 7'b0010010;
            6 : seg = 7'b0000010;
            7 : seg = 7'b1111000;
            8 : seg = 7'b0000000;
            9 : seg = 7'b0010000;
            10: seg = 7'b0001000;
            11: seg = 7'b0000011;
            12: seg = 7'b1000110;
            13: seg = 7'b0100001;
            14: seg = 7'b0000110;
            15: seg = 7'b1111111; // Shuts off output
            //15: seg = 7'b0001110;
            //switch off 7 segment character when the bcd digit
is not a decimal number.
            default : seg = 7'b1111111;
        endcase
     end

      assign an = 4'b1110;

     endmodule
```