PicoBlaze(3)
Interfaces de Entrada/Salida

Diseño de Sistemas con FPGA

1er cuatrimestre 2009

Patricia Borensztejn

Temario

- Esta clase contiene:
  - Interface de entrada/salida en PicoBlaze
  - Ejemplo 1: a^2+b^2 usando switches, leds y pushbottoms
  - Ejemplo 2: a^2+b^2 usando un multiplicador externo.

Interface de E/S

- PicoBlaze tiene una estructura genérica y muy sencilla para interfacear con los dispositivos de E/S.
- La interface específica para cada dispositivo que la aplicación necesite, se debe construir.
- Estructura genérica:
  - Instrucciones input y output para escribir y leer datos de los puertos de E/S
  - Señales:
    - port_id: señal de 8 bits que especifica una dirección de E/S
    - in_port: señal de 8 bits de donde PicoBlaze obtiene su dato después de una instrucción input.
    - out_port: señal de 8 bits donde PicoBlaze pone su dato durante la ejecución de la instrucción output.
    - read_strobe: señal de 1 bit que es seteada en el segundo ciclo de una instrucción input
    - write_strobe: señal de 1 bit que es seteada en el segundo ciclo de una instrucción output

Interface de salida: decodificación

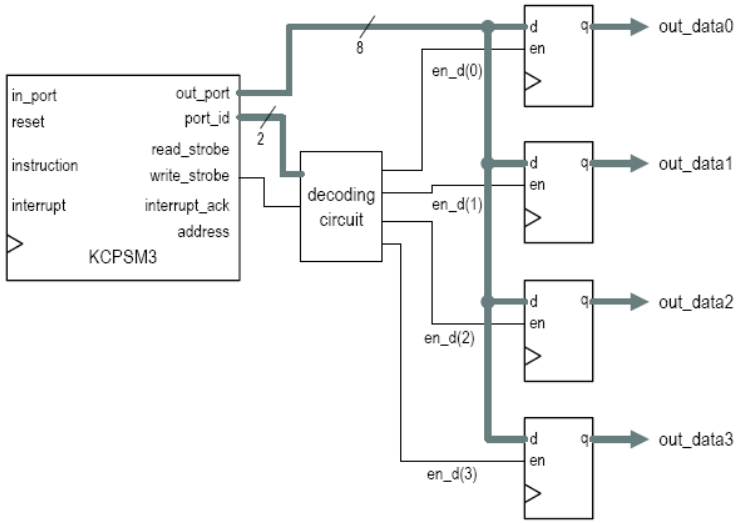


Figure 16.2 Output decoding of four output registers.

- Supongamos que tenemos 4 buffers de salida.
- Las direcciones de los puertos podemos definirlas así: 00,01,10,11
- Si sólo hubiera un puerto de salida, no hay lógica de decodificación y la señal write_strobe se conecta directamente al enable del registro.

```
always @*
   if (write_strobe)
      case (port_id[1:0])
         2'b00: en_d = 4'b0001;
         2'b01: en_d = 4'b0010;
         2'b10: en_d = 4'b0100;
         2'b11: en_d = 4'b1000;
      endcase
   else
      en_d = 4'b0000;
```

Table 17.1 Truth table of a decoding circuit

input			output
write_strobe	port_id[1]	port_id[0]	en_d
0	–	–	0000
1	0	0	0001
1	0	1	0010
1	1	0	0100
1	1	1	1000

Interface de salida: decodificación

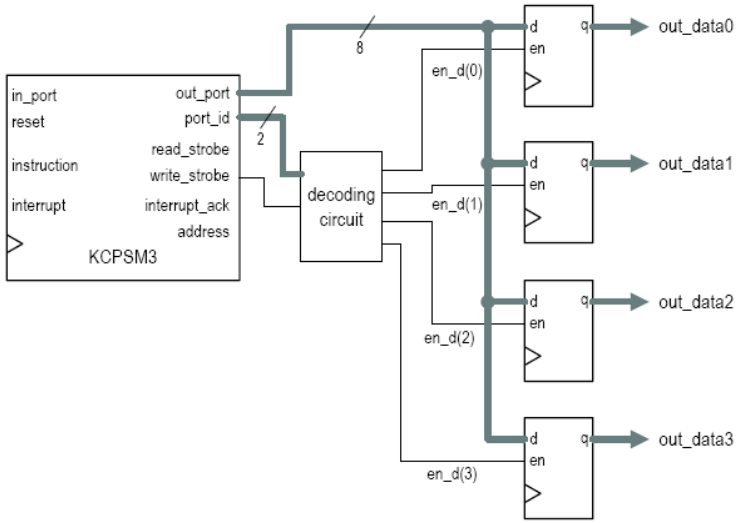


Figure 16.2 Output decoding of four output registers.

- Pero también podemos definirlas así:
  00000001
  00000010
  00000100
  00001000
- De esta manera, el enable es el bit correspondiente de la dirección.

```
always @*
   if (write_strobe)
      en_d = port_id[3:0];
   else
      en_d = 4'b0000;
```

Interface de salida: Output Port

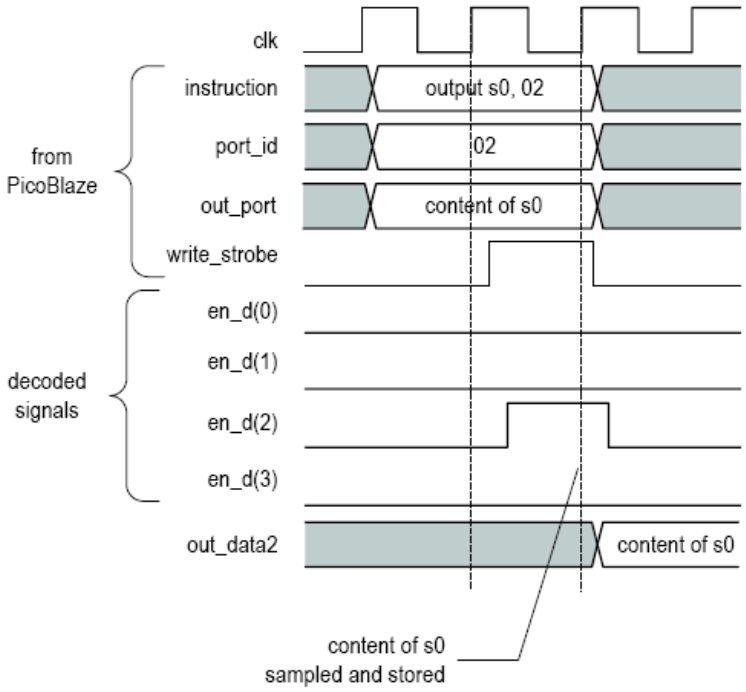


Figure 16.1 Timing diagram of an **output** instruction.

- En el segundo ciclo de la instrucción **output** se activa **write_strobe**.
- Al siguiente flanco del reloj, luego de decodificado el registro de salida, se almacenan los datos.

Interface de Entrada: Instrucción Input

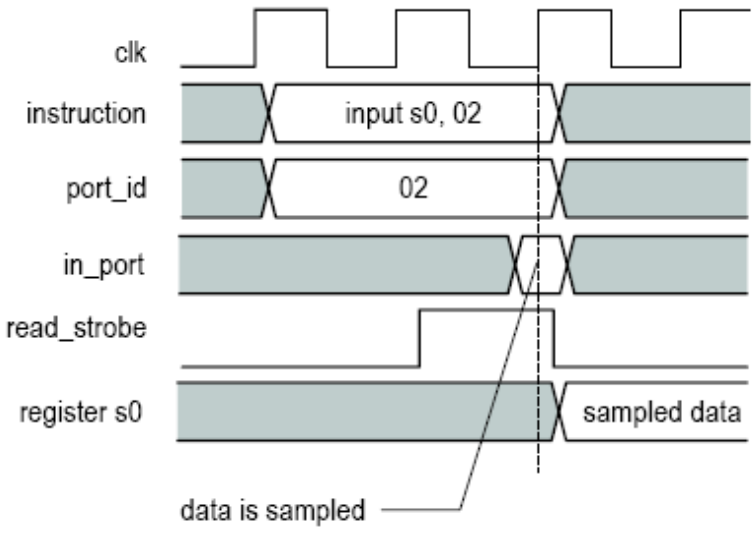


Figure 16.3 Timing diagram of an **input** instruction.

- En el segundo ciclo de la instrucción, se activa read_strobe.
- Al siguiente flanco, se almacena el valor de in_port en el registro s0.
- El circuito externo debe garantizar que el dato de entrada sea estable en ese momento.

Interface de Entrada: Decodificación

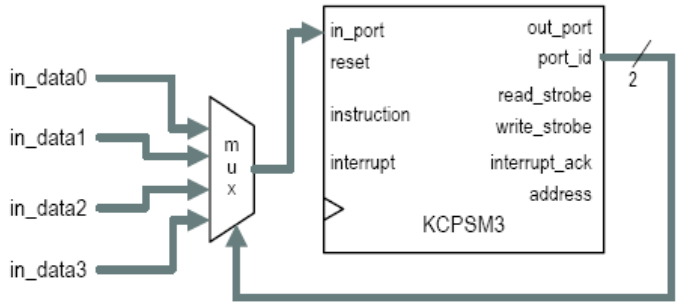


Figure 16.4 Block diagram of four continuous-access ports.

- Si los datos de los puertos de entrada son contínuos: por ejemplo, los switches, con un multiplexor basta para obtener los datos del puerto seleccionado.
- Si los datos de los puertos de entrada son de acceso único (por ejemplo, un carácter de la UART debe recibirse solo una vez) tiene que haber un mecanismo para eliminar el dato leído de los buffers para evitar que sea leído otra vez.

Interface de Entrada: Decodificación

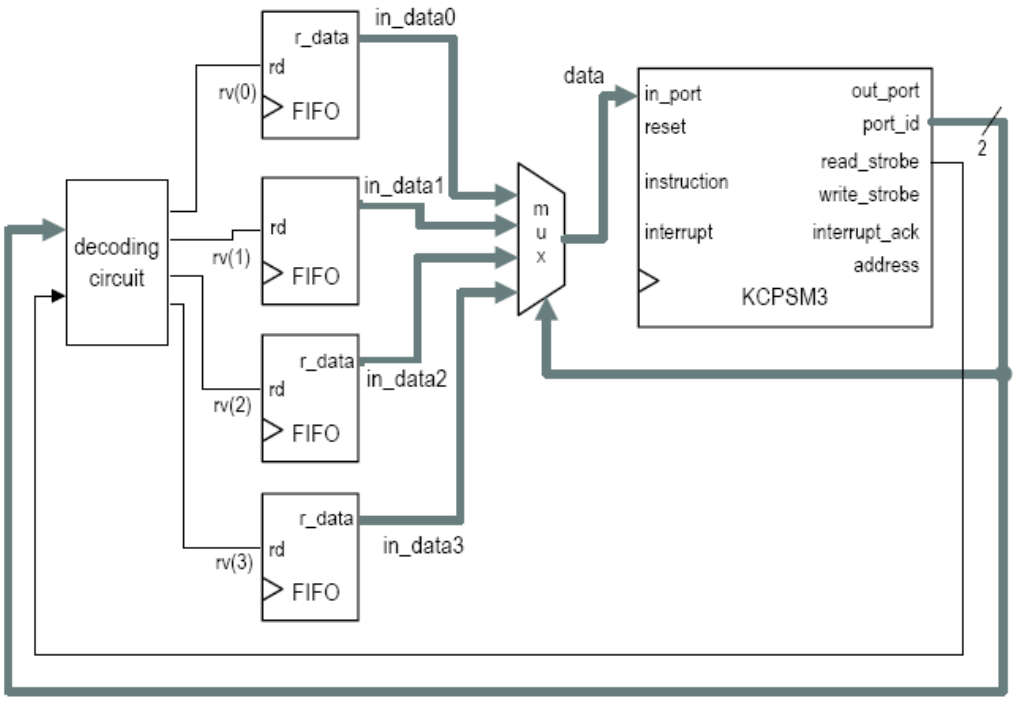


Figure 16.5 Block diagram of four single-access ports.

- Ejemplo: 4 FIFOs.
- Una vez que se ejecutó la instrucción, hay que deshacerse del dato viejo. La señal read_strobe puede ser utilizada a tal efecto. En este caso como la señal rd de las FIFO (read y pop).

Interface de Entrada: Decodificación

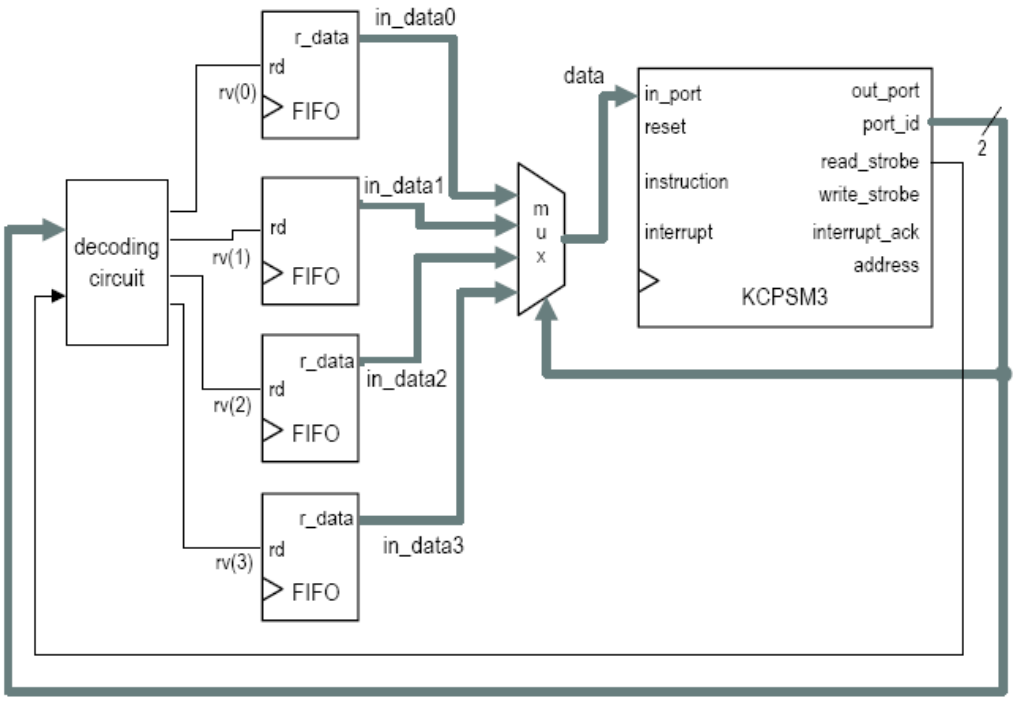


Figure 16.5 Block diagram of four single-access ports.

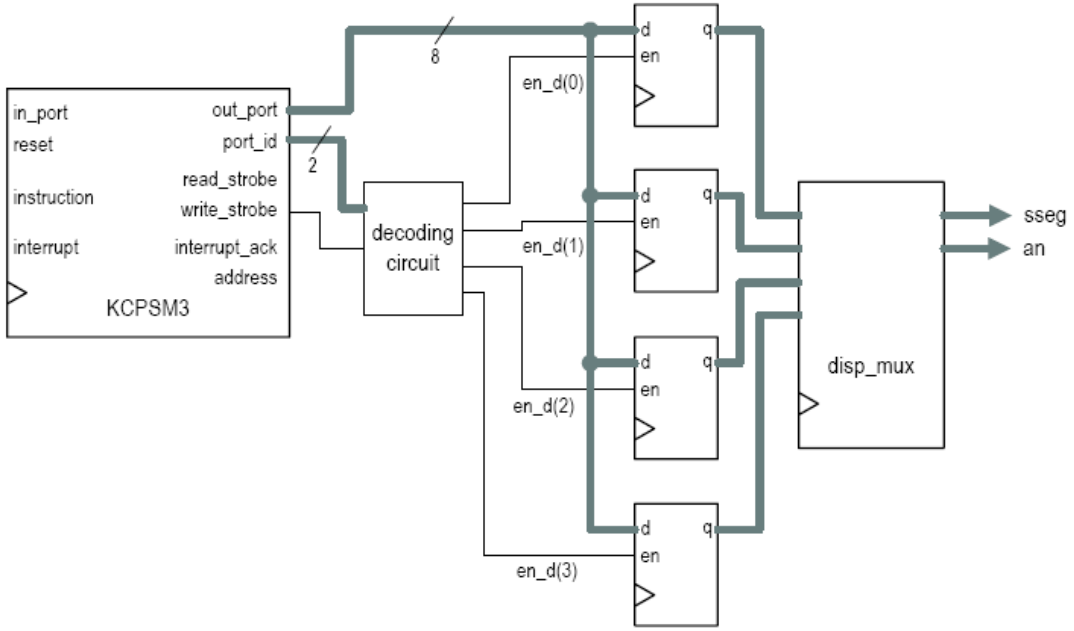
```
// multiplexing circuit
always @*
   case (port_id[1:0])
      2'b00: data = in_data0;
      2'b01: data = in_data1;
      2'b10: data = in_data2;
      2'b11: data = in_data3;
   endcase
// decoding circuit
 always @*
    if (read_strobe)
       case (port_id[1:0])
          2'b00: rv = 4'b0001;
          2'b01: rv = 4'b0010;
          2'b10: rv = 4'b0100;
          2'b11: rv = 4'b1000;
       endcase
    else
       rv = 4'b0000;
```

Ejemplo ampliado : a^2+b^2

- Para poder entrar los valores de a y de b alternativamente como unsigned de 8 bits usamos:
  - Pushbottom 0: indica cargar a o b, alternativamente
  - Switch: provee los valores de a y de b
  - Pushbottom 1: limpia todo. Memoria y registros.
- Además, en los leds (7 segmentos) , que usaremos 4, podemos ver los siguientes valores:
  - a, b, a^2,b^2 y a^2+b^2
  - Para seleccionar lo que queremos ver usaremos los switches:
    - 0: a^2+b^2
    - 1: a
    - 2: b
    - 3: a^2
    - 4: b^2
  - Como el producto tiene 17 bits, usaremos el punto del LED que esta mas a la izquierda como bit 16.

Ejemplo a^2+b^2: salida

- Recordemos que los LEDS comparten los mismos pines. O sea que es necesario un circuito multiplexador para los LEDs.
- Esta implementación lo hace por hardware, es decir para el PicoBlaze hay 4 puertos de salida, uno por LED.



Ensamblador para la salida de los LEDs

```
...
;data RAM address alias
constant led0, 10
constant led1, 11
constant led2, 12
constant led3, 13
...
;output port definitions
constant sseg0_port, 00     ;7-seg led 0
constant sseg1_port, 01     ;7-seg led 1
constant sseg2_port, 02     ;7-seg led 2
constant sseg3_port, 03     ;7-seg led 3
...
disp_led:
    fetch data, led0
    output data, sseg0_port
    fetch data, led1
    output data, sseg1_port
    fetch data, led2
    output data, sseg2_port
    fetch data, led3
    output data, sseg3_port
    return
```

Ejemplo a^2+b^2: disp_mux

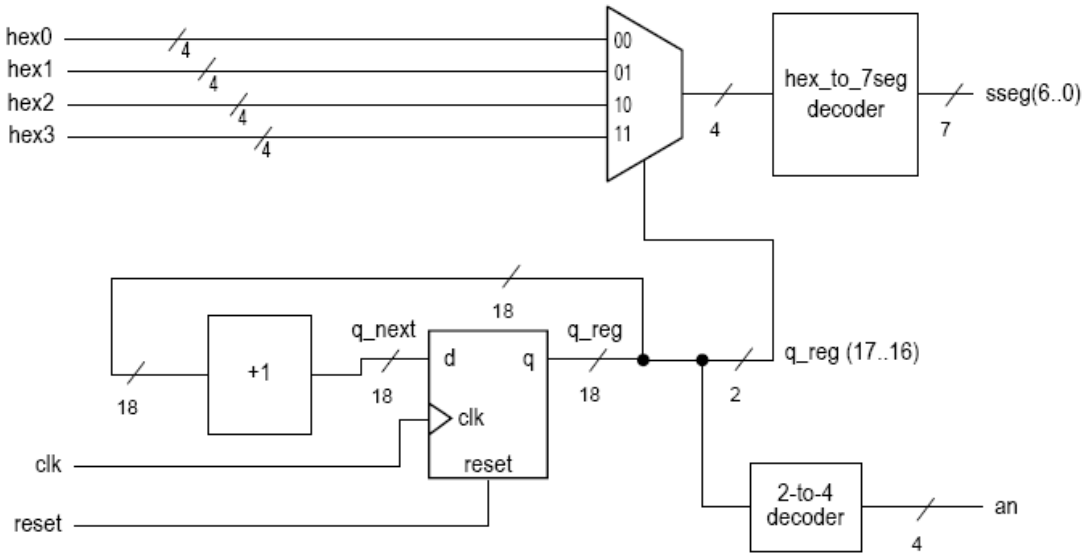


Figure 4.9 Block diagram of a hexadecimal time-multiplexing circuit.

- Se usa un contador de 18 bits, y los dos bits mas altos controlan el multiplexor. Es decir que los cambios se producen cada 2^{16} ciclos. La frecuencia de refresco es de $50MHz/2^{16}= 800Hz$

Ejemplo: Interface de entrada

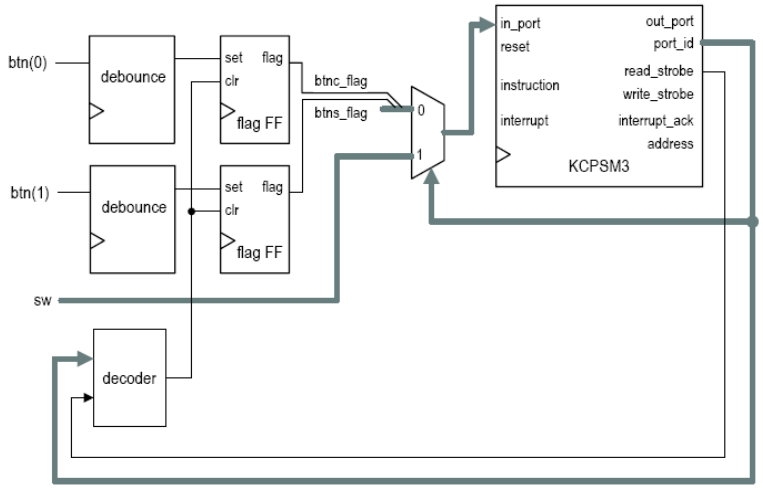


Figure 16.7 Input interface of a square circuit.

- Tenemos dos tipos de entradas:
  - Switch: continuo, los datos están siempre
  - Pushbottom: acceso único, pues cuando uno aprieta el botón solo quiere un evento: cargar a, por ejemplo.
- Como el puerto de datos de PicoBlaze es de 8 bits, en el dibujo se muestra que las entradas de los dos PushBottoms se han unido.
- Los circuitos debounce se utilizan para generar una señal pura en los pushbottoms.
- Hay dos biestables FF que se utilizan para setear o limpiar el evento “apretar botón”

Circuito Antirrebote

- Como el pushbottom es un dispositivo mecánico, al ser este presionado, puede rebotar mas de una vez antes de quedar establecida la señal. El tiempo de establecimiento puede ser aproximadamente de 20 ms.
- El propósito del circuito antirrebote es el de filtrar los rebotes.

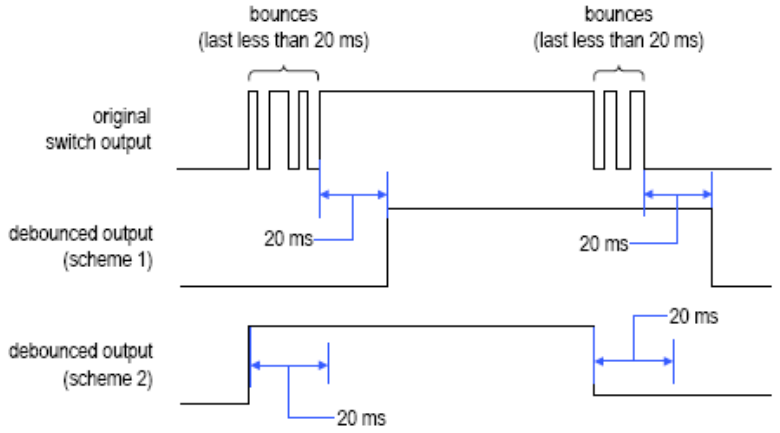
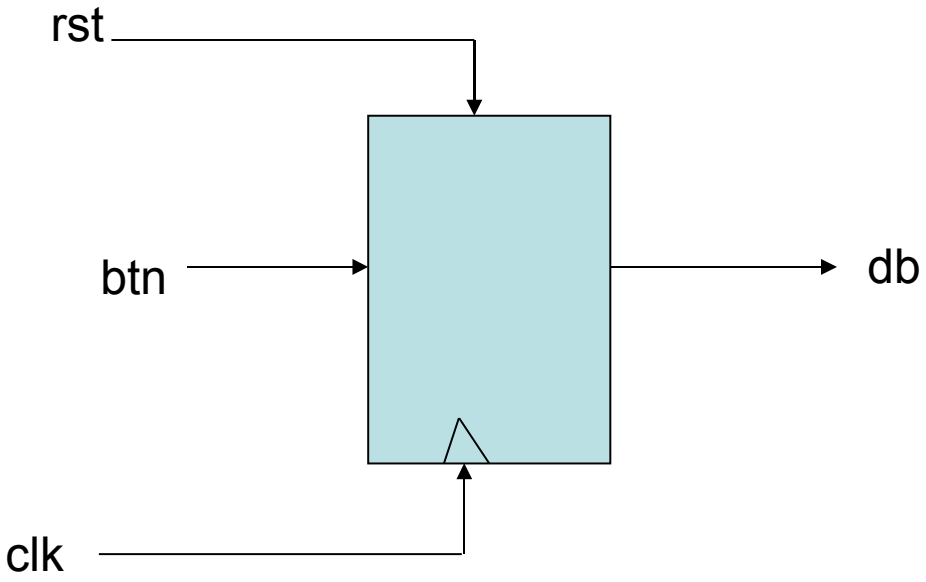


Figure 5.8 Original and debounced waveforms.

Circuito Antirrebote

- La idea es utilizar un timer que genere una señal cada 10 ms, y una FSM. La FSM usa esta información para saber si la entrada está o no estabilizada.
- Si la entrada sw está activa por mas de 20 ms, entonces ha habido un evento.



Circuito Entrada

- Cuando se presiona el btn, el circuito antirrebote pone a “1” la señal. Esta señal permanecerá encendida hasta que el PicoBlaze la obtenga a través de la instrucción input. La misma instrucción activa la señal de clr de los FF.

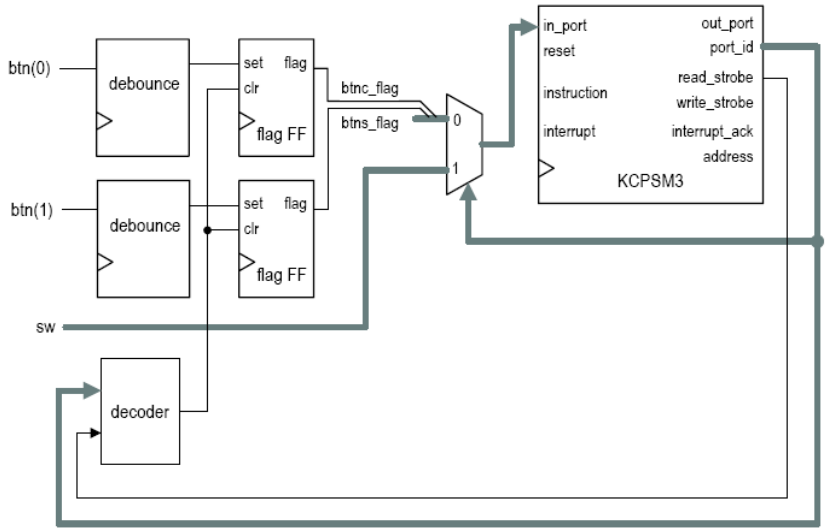
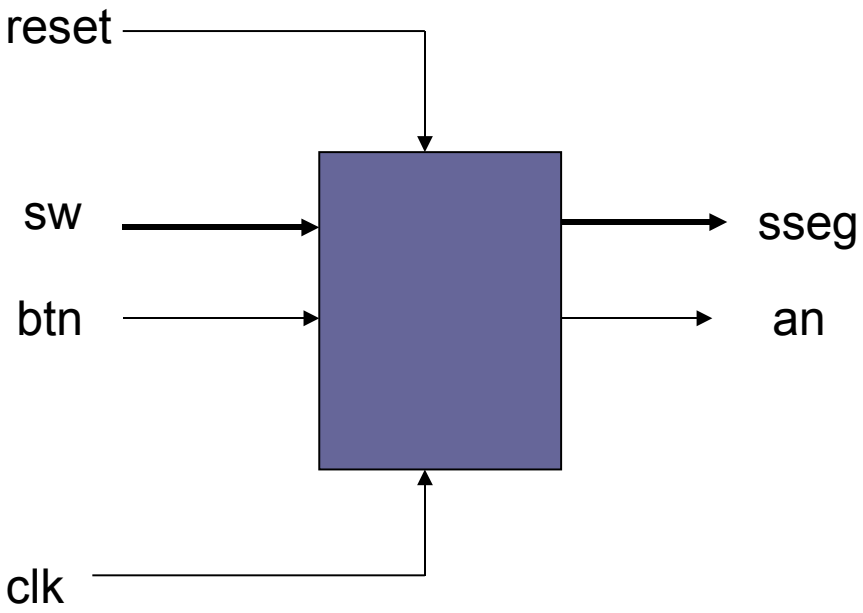


Figure 16.7 Input interface of a square circuit.

```
;==========================================
;routine: proc_btn
;  function: check two buttons and process the display
;  input reg:
;     switch_a_b: ram offset (0 for a and 2 for b)
;  output register:
;     s3: store input port flag
;     switch_a_b: may be toggled
;  temp register used: data, addr
;==========================================
proc_btn:
   input s3, rd_flag_port  ;get flag
   ;check and process c button
   test s3, 01             ;check c button flag
   jump z, chk_btns        ;flag not set
   call init               ;flag set, clear
   jump proc_btn_done
chk_btns:
   ;check and process s button
   test s3, 02             ;check s button flag
   jump z, proc_btn_done   ;flag not set
   input data, sw_port     ;get switch
   load addr, a_lsb        ;get addr of a
   add addr, switch_a_b    ;add offset
   store data, (addr)      ;write data to ram
   ;update current disp position
   xor switch_a_b, 02      ;toggle between 00, 02
proc_btn_done:
   return
```

Aplicación



- Codigo ensamblador: btn_rom
- Código Verilog para el sistema completo: pico_btn
- Módulos Verilog:
  - disp_mux
  - debounce
  - kcpsm3

Ejemplo más ampliado: a2+b2

- Agregamos al diseño dos periféricos mas:
  - Un multiplicador combinatorio, para acelerar la multiplicación
    - PicoBlaze necesita un máximo de 60 instrucciones para multiplicar. A dos ciclos por instrucción.
    - Una alternativa es utilizar los multiplicadores embebidos de Spartan y Virtex.
  - Una UART para comunicarnos con la PC

Multiplicador Hardware

- Para poder comunicarse con el multiplicador, PicoBlaze lo ve como un dispositivo de E/S.
- Es necesario ampliar los puertos de E/S para incluir:
  - Salida: los dos operandos de 8 bits

```
constant mult_src0_port, 05    ;multiplier operand 0
constant mult_src1_port, 06    ;multiplier operand 1
```

  - Entrada: El producto de 16 bits

```
constant mult_prod0_port, 03   ;multiplication product 8 LSBs
constant mult_prod1_port, 04   ;multiplication product 8 MSBs
```

- Código Ensamblador

```
square:
   ;calculate a*a
   fetch s3, a_lsb        ;load a
   fetch s4, a_lsb        ;load a
   call mult_hard         ;calculate a*a
   store s6, aa_lsb       ;store lower byte of a*a
   store s5, aa_msb       ;store upper byte of a*a
   …….

   return
```

```
mult_hard:
   output s3, mult_src0_port
   output s4, mult_src1_port
   input s5, mult_prod1_port
   input s6, mult_prod0_port
   return
```

Multiplicador Hardware

- Ahora es necesario agregar las entradas y salidas a las interfaces de entrada/salida que ya teníamos.

```
// ======================
 //  output interface
 // ======================
 //    outport port id:
 //      0x00: ds0
 //      0x01: ds1
 //      0x02: ds2
 //      0x03: ds3
 //      0x04: uart_tx_fifo
 //      0x05: m_src0
 //      0x06: m_src1
 // ========================
 // registers
 always @(posedge clk)
    begin
       if (en_d[0])          ds0_reg <= out_port;
       if (en_d[1]           ds1_reg <= out_port;
       if (en_d[2])          ds2_reg <= out_port;
       if (en_d[3])          ds3_reg <= out_port;
       if (en_d[5])          m_src0_reg <= out_port;
       if (en_d[6])          m_src1_reg <= out_port;
    end
```

```
// decoding circuit for enable signals
  always @*
     if (write_strobe)
        case (port_id[2:0])
           3'b000: en_d = 7'b0000001;
           3'b001: en_d = 7'b0000010;
           3'b010: en_d = 7'b0000100;
           3'b011: en_d = 7'b0001000;
           3'b100: en_d = 7'b0010000;
           3'b101: en_d = 7'b0100000;
           default: en_d = 7'b1000000;
        endcase
     else
        en_d = 7'b0000000;
```

Multiplicador Hardware, Entrada

```
//  input interface
//
//    input port id
//      0x00: flag
//      0x01: switch
//       0x03: prod lower byte
//      0x04: prod higher byte
// ======================================
   // input register (for flags)
   always @((clr_btn_flag)  ? 1'b0 :posedge clk)
      begin
         btnc_flag_reg <= btnc_flag_next;
         btns_flag_reg <= btns_flag_next;
      end
   assign btnc_flag_next = (set_btnc_flag) ? 1'b1 :
                           btnc_flag_reg;
   assign btns_flag_next = (set_btns_flag) ? 1'b1 :
                          (clr_btn_flag)  ? 1'b0 :
                           btns_flag_reg;
```

```
   // decoding circuit for clear signals
    assign clr_btn_flag = read_strobe && (port_id[0]==1'b0);
    // input multiplexing
    always @*
       case(port_id[0])
          3'b000: in_port = {6'b0, btns_flag_reg, btnc_flag_reg};
          3'b001: in_port = sw;
          3'b011: in_port = prod[7:0];
          3'b100  : in_port = prod[15:8];
           default: in_port= ……

       endcase

endmodule
```

Multiplicador Hardware

- Finalmente, es necesario incluir el multiplicador en el código Verilog.

```
// combinational multiplier
 assign prod = m_src0_reg * m_src1_reg;
```