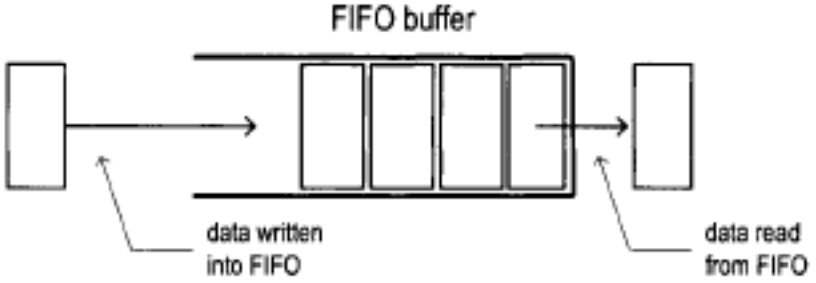
FIFO

Diseño de Sistemas con FPGA

Patricia Borensztejn

FIFO

- Una cola FIFO es una estructura de datos con dos señales: wr y rd.
- La señal wr, cuando está activa, permite que un dato sea escrito en la FIFO.
- La cabeza de la cola está siempre visible, o sea que siempre podemos leer. La señal rd, cuando está activa, indica que el elemento de la cabeza se puede eliminar.



FIFO: Implementación con una cola circular

- Dos señales de control:
  - wr: si está activada, se escribe el dato
  - rd: si está activada, el primer elemento de la cola se remueve.
  - El primer elemento de la cola puede leerse siempre.
- Implementación:
  - Los registros forman una cola circular, con dos punteros:
    - wr ptr: apunta a la cabeza de la cola (head)
    - rd ptr : apunta a la cola de la cola (tail)
    - Los punteros avanzan una posición por cada escritura o lectura
    - Hay dos señales de estado: full, empty

FIFO

- La implementación consiste de:
  - Banco de Registros
  - Punteros a la cola y a la cabeza
  - Flags lleno y vacío
  - Lógica del estado futuro (punteros)
  - Lógica de salida (flags)

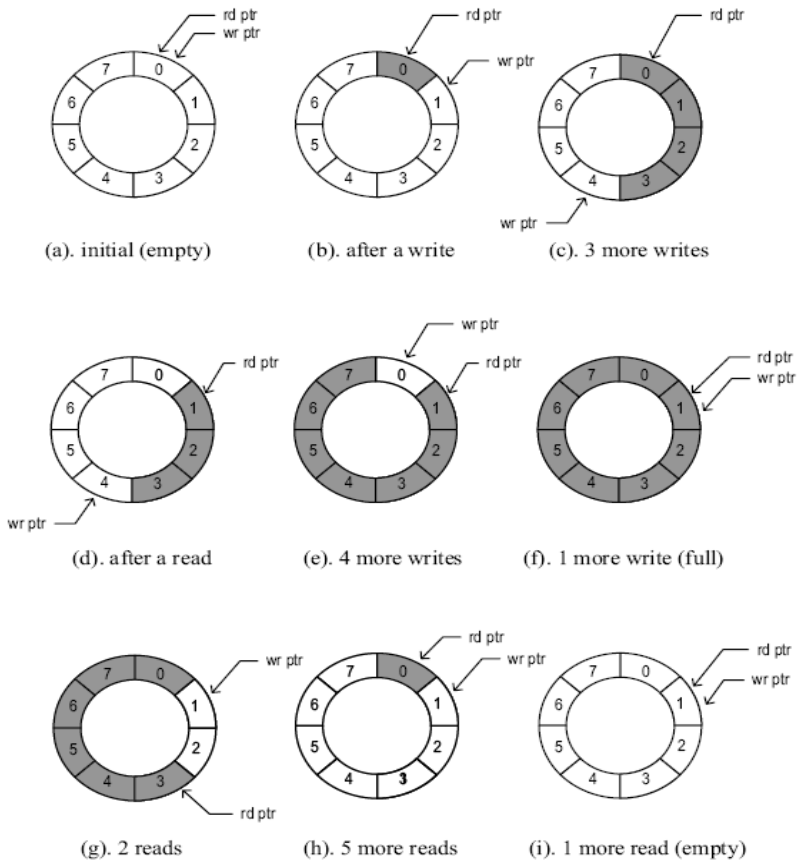
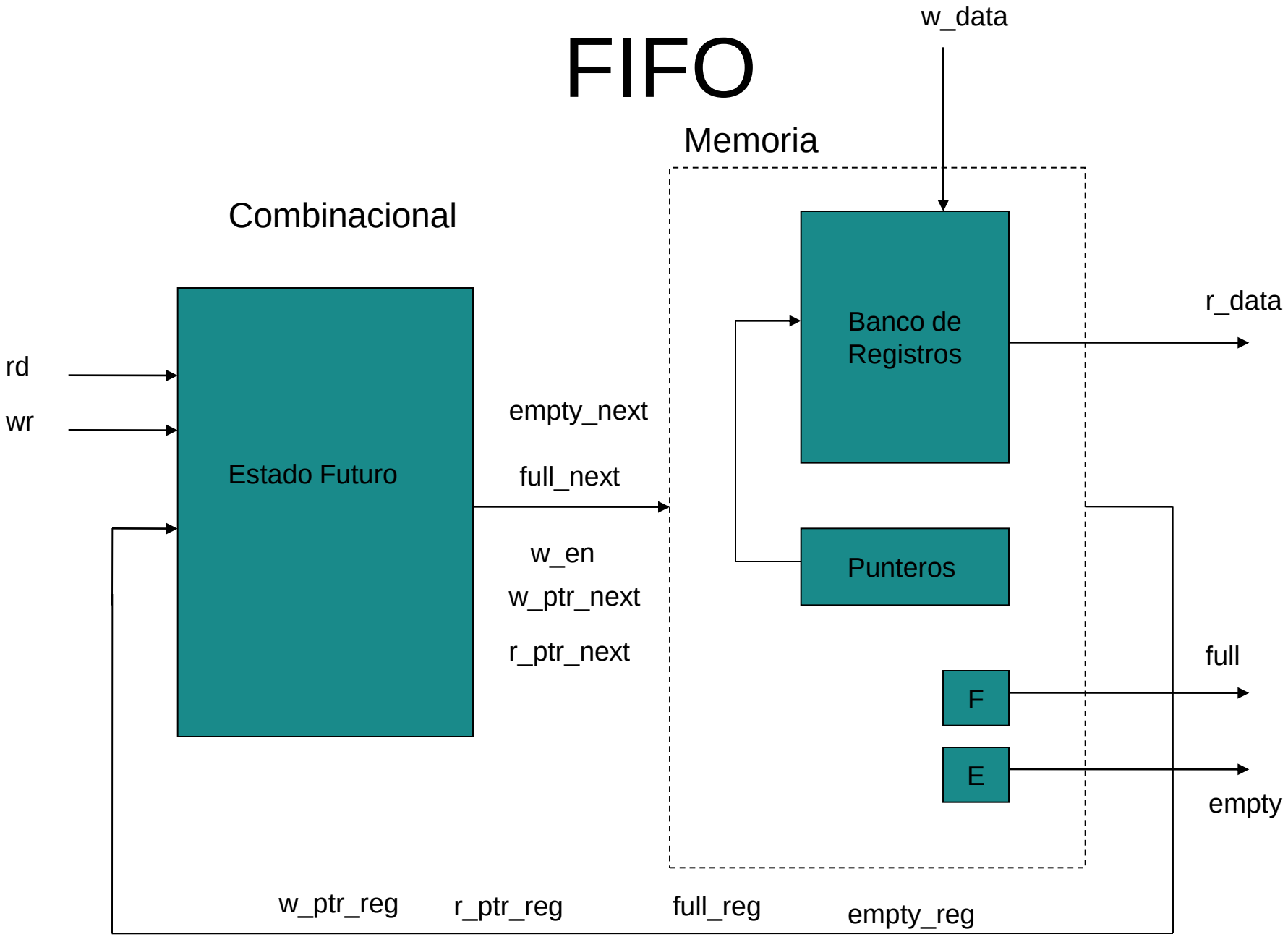


Figure 4.11 FIFO buffer based on a circular queue.

FIFO


w_data
Memoria
Combinacional
Banco de Registros
r_data
rd
wr
Estado Futuro
empty_next
full_next
w_en
w_ptr_next
r_ptr_next
Punteros
full
F
E
empty
w_ptr_reg
r_ptr_reg
full_reg
empty_reg

FIFO: Implementación

```
module fifo
  #(
   parameter B=8, // number of bits in a word
            W=4  // number of address bits
  )
  (
   input wire clk, reset,
   input wire rd, wr,
   input wire [B-1:0] w_data,
   output wire empty, full,
   output wire [B-1:0] r_data
  );

  //signal declaration
  reg [B-1:0] array_reg [2**W-1:0];  // register array
  reg [W-1:0] w_ptr_reg, w_ptr_next, w_ptr_succ;
  reg [W-1:0] r_ptr_reg, r_ptr_next, r_ptr_succ;
  reg full_reg, empty_reg, full_next, empty_next;
  wire wr_en;
```

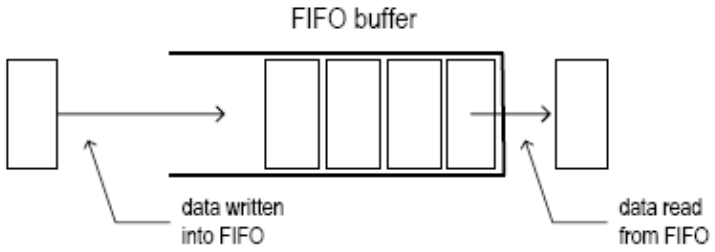
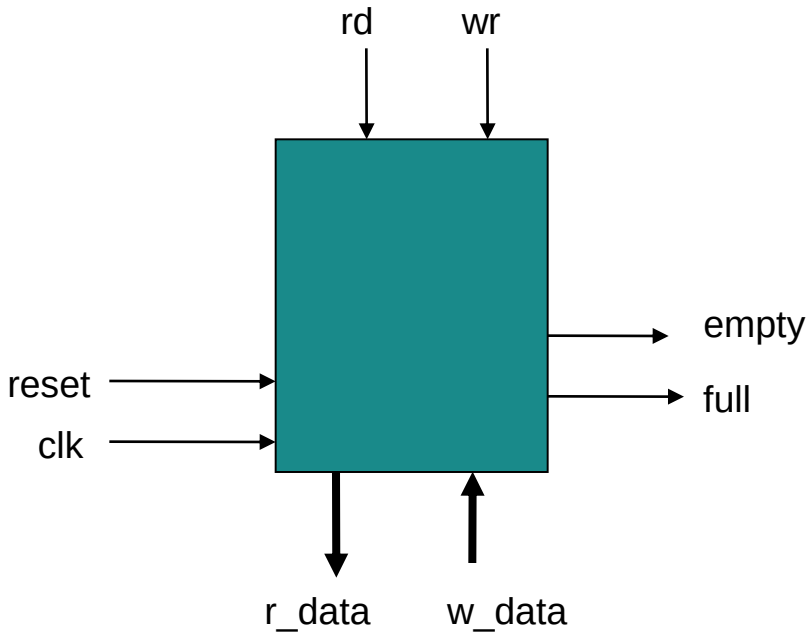
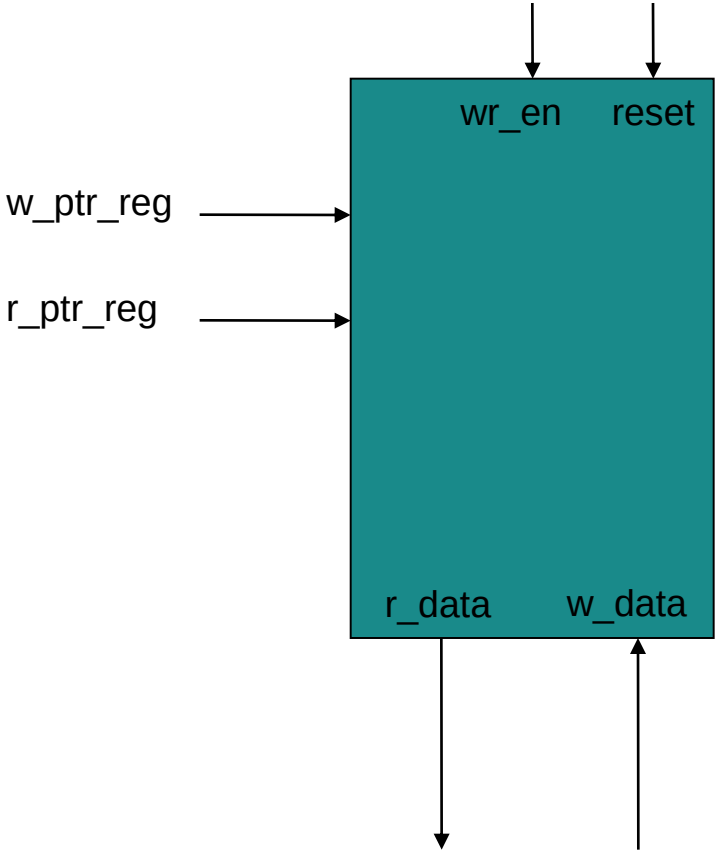


Figure 4.10 Conceptual diagram of a FIFO buffer.



FIFO: Banco de Registros

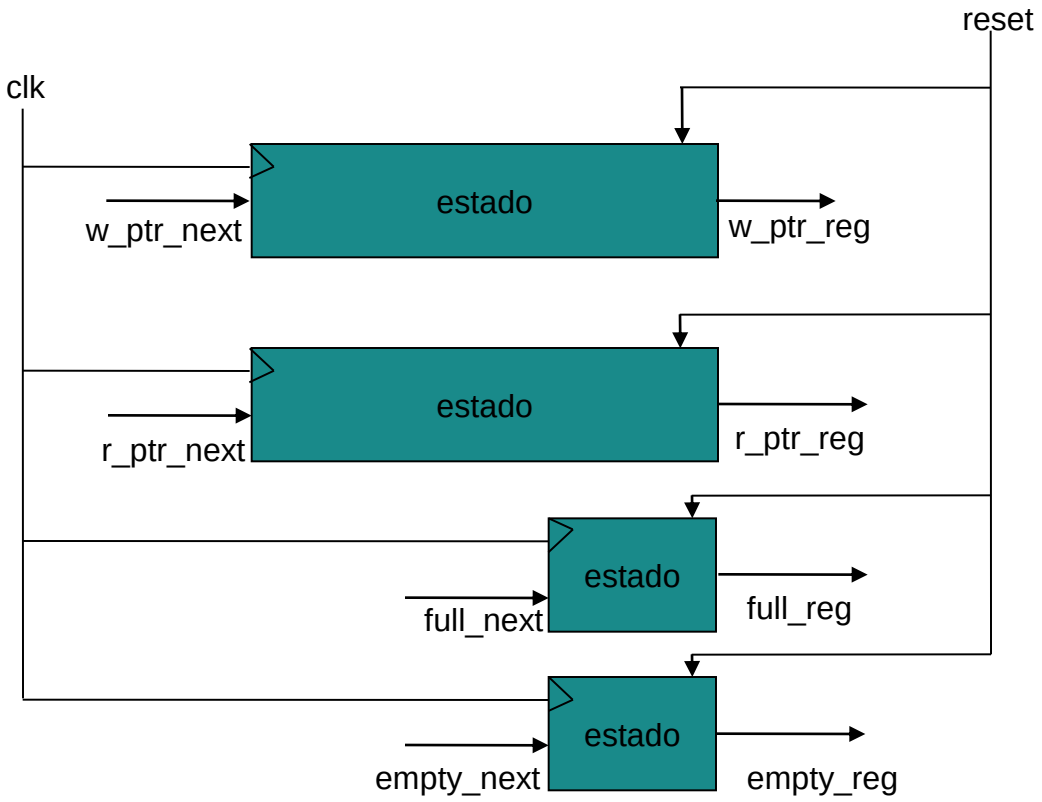



```
// body
// register file write operation
always @(posedge clk)
   if (wr_en)
      array_reg[w_ptr_reg] <= w_data;

// register file read operation
assign r_data = array_reg[r_ptr_reg];

// write enabled only when FIFO is not full
assign wr_en = wr & ~full_reg;
```

Escritura Síncrona, Lectura Asíncrona

FIFO : Punteros y Señales de Estado

```
 // fifo control logic
// register for read and write pointers
always @(posedge clk, posedge reset)
  if (reset)
    begin
      w_ptr_reg <= 0;
      r_ptr_reg <= 0;
      full_reg <= 1'b0;
      empty_reg <= 1'b1;
    end
  else
    begin
      w_ptr_reg <= w_ptr_next;
      r_ptr_reg <= r_ptr_next;
      full_reg <= full_next;
      empty_reg <= empty_next;
    end
```



FIFO: Lógica para el Estado Futuro

```
    // next-state logic for read and write
pointers
   always @*
   begin
      // successive pointer values
      w_ptr_succ = w_ptr_reg + 1;
      r_ptr_succ = r_ptr_reg + 1;
      // default: keep old values
      w_ptr_next = w_ptr_reg;
      r_ptr_next = r_ptr_reg;
      full_next = full_reg;
      empty_next = empty_reg;
```

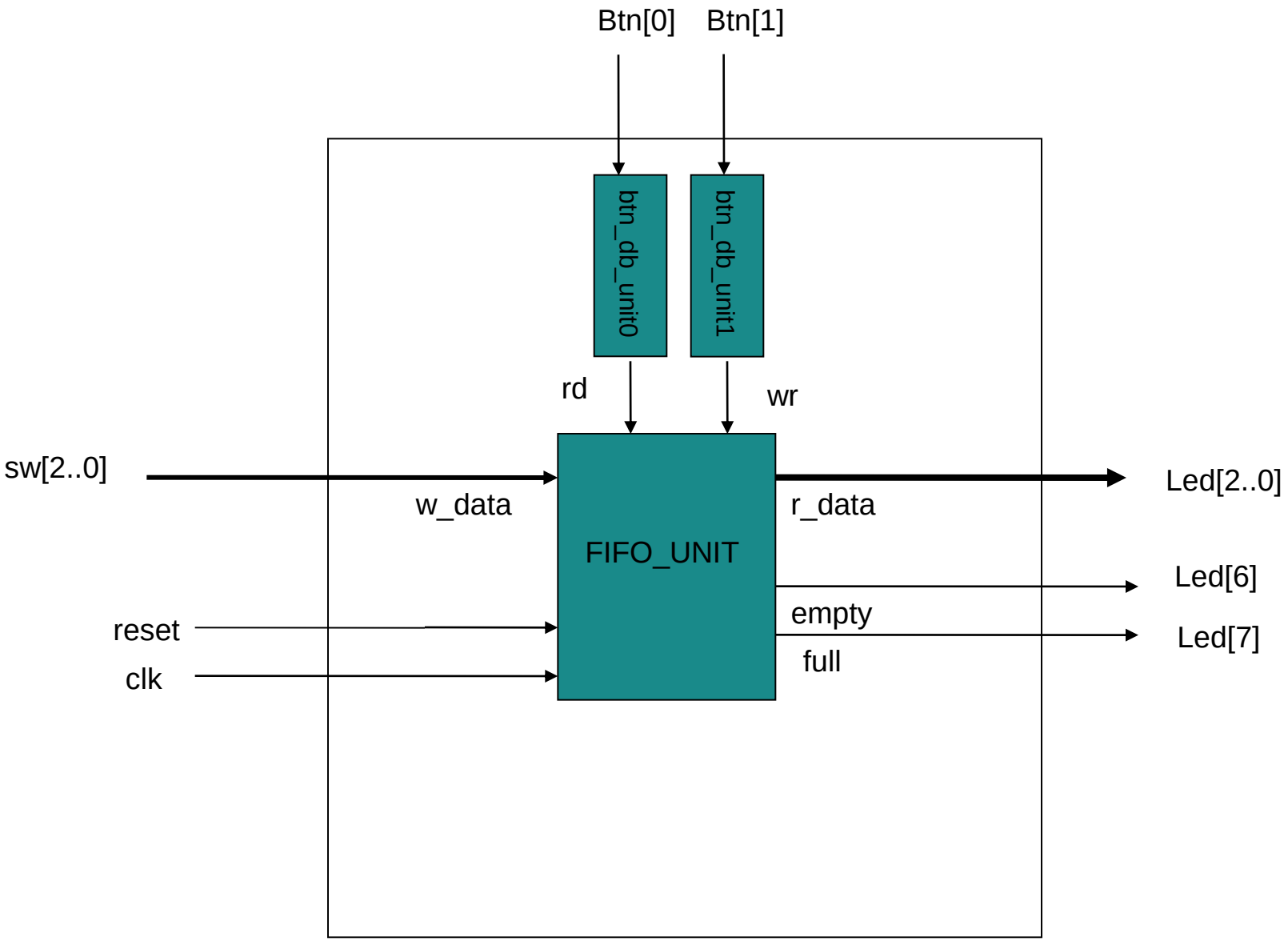
Always block combinacional

```
case ({wr, rd})
        // 2'b00:  no op
        2'b01: // read
          if (~empty_reg) // not empty
            begin
              r_ptr_next = r_ptr_succ;
              full_next = 1'b0;
              if (r_ptr_succ==w_ptr_reg)
                empty_next = 1'b1;
            end
        2'b10: // write
          if (~full_reg) // not full
            begin
              w_ptr_next = w_ptr_succ;
              empty_next = 1'b0;
              if (w_ptr_succ==r_ptr_reg)
                full_next = 1'b1;
            end
        2'b11: // write and read
          begin
            w_ptr_next = w_ptr_succ;
            r_ptr_next = r_ptr_succ;
          end
      endcase
   end
```

FIFO: Lógica para la Salida

```
 // output
   assign full = full_reg;
   assign empty = empty_reg;

endmodule
```

FIFO Test


Btn[0]
Btn[1]
btn_db_unit0
btn_db_unit1
rd
wr
sw[2..0]
w_data
FIFO_UNIT
r_data
Led[2..0]
Led[6]
empty
Led[7]
full
reset
clk

Código Test FIFO

```
// Listing 4.21
module fifo_test
   (
    input wire clk, reset,
    input wire [1:0] btn,
    input wire [2:0] sw,
    output wire [7:0] led
   );

   // signal declaration
   wire [1:0] db_btn;

   // debounce circuit for btn[0]
   debounce btn_db_unit0
      (.clk(clk), .reset(reset), .sw(btn[0]),
       .db_level(), .db_tick(db_btn[0]));
   // debounce circuit for btn[1]
   debounce btn_db_unit1
      (.clk(clk), .reset(reset), .sw(btn[1]),
       .db_level(), .db_tick(db_btn[1]));
   // instantiate a 2^2-by-3 fifo
   fifo #(.B(3), .W(2)) fifo_unit
      (.clk(clk), .reset(reset),
      .rd(db_btn[0]), .wr(db_btn[1]), .w_data(sw),
      .r_data(led[2:0]), .full(led[7]), .empty(led[6]));
   // disable unused leds
   assign led[5:3] = 3'b000;

 endmodule
```