Secuenciales III (Metodología de Diseño)

Diseño de Sistemas con FPGA

Patricia Borensztejn

Revisado 10/2011

Modelado de Sistemas Secuenciales

- Según la función del siguiente estado se dividen en:
  - Circuitos secuenciales regulares: el estado siguiente se construye con un incrementador o un shifter
  - FSM (Finite State Machines): el estado siguiente no exhibe un pattern repetitivo, sino mas bien random.
  - FSMD (FSM with Data Path): combina los dos tipos anteriores de sistemas secuenciales, es decir, está compuesto por dos partes:
    - FSM: llamado “control path” es el encargado de examinar las entradas externas y de generar las señales para el funcionamiento de los
    - Circuitos secuenciales regulares: llamado “data path”.

FSMD. Metodología RTL

- Se usa para implementar algoritmos representados con la metodología RT: Register Transfer
- En la metodología RT las operaciones son especificadas como transferencias entre registros.
- RTL se utiliza para especificar sistemas complejos (por ejemplo, un procesador)

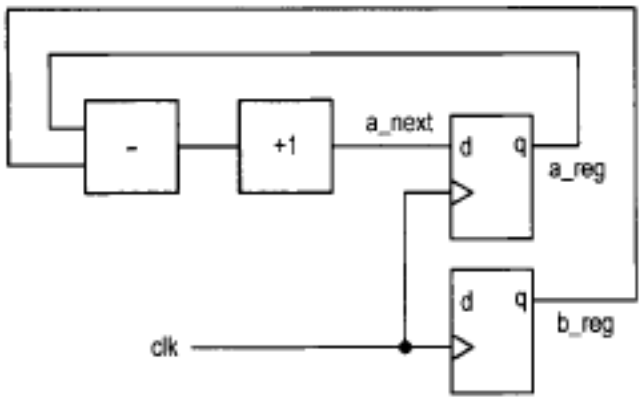
RTL

$$r_{dest} \leftarrow f(r_{src1}, r_{src2}, \ldots, r_{srcn})$$

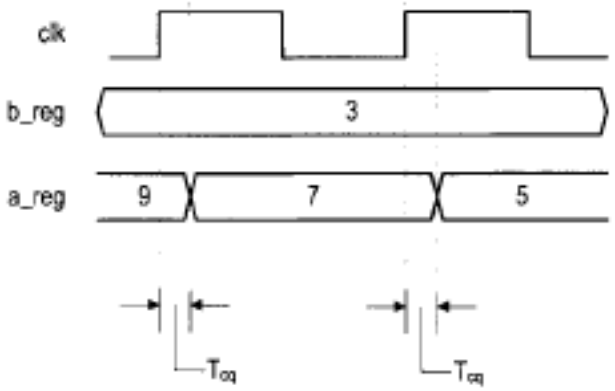
- donde:
  - f es la operación realizada por un circuito combinacional
  - r_{src} son los registros fuente
  - r_{dst} es el registro destino donde se almacena el resultado en el flanco ascendente del reloj

RTL

- Ejemplos:
  - r1←0
  - r1←r3
  - r1←r2 >>3
  - i←i+1
  - a←a-b+1



(a) Block diagram



(b) Timing diagram

- Una operación RT se implementa construyendo un circuito combinacional para la funcion f y conectando las entradas y salidas a los registros correspondientes

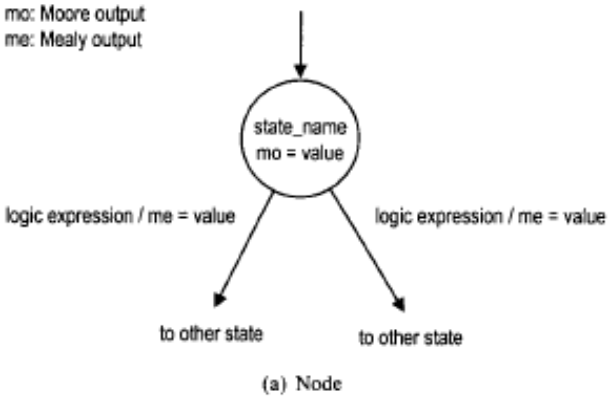
RTL, FSM y ASM

- Como cada operación RT se realiza en cada flanco de reloj, su temporización es similar a la de un FSM y por lo tanto éstos pueden ser utilizados para representarlas.
- Recordemos que:
  - Las máquinas de estado finito (FSM) se pueden representar de dos maneras:
    - Mediante Diagramas de Estado (lo que venimos usando hasta ahora) (FSM)
    - Mediante Diagramas Algoritmicos de Estado (ASM)

ASM

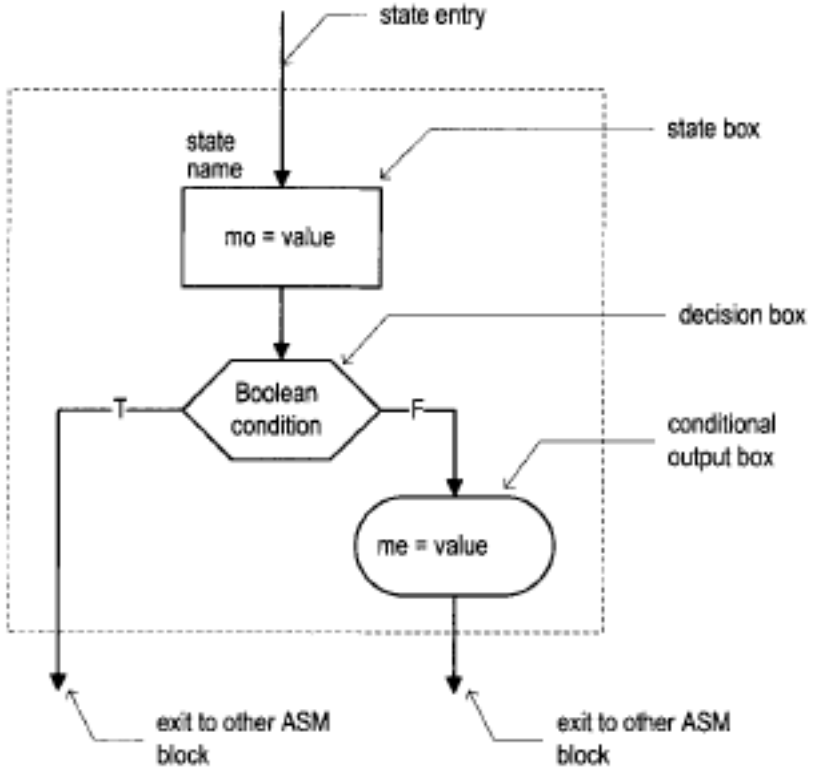
- Un diagrama ASM (Algorithmic State Machine) está formado por una red de bloques.
- Cada bloque contiene:
  - State Box: en cuyo interior pondremos el valor las salidas de Moore. Corresponde a los estados de la representación FSM, pero a diferencia de estos, los State Box tienen una única salida
  - Decision Box: Testea la entrada y determina el camino a tomar. Tiene dos caminos de salida: el camino T(rue) y el camino F(alse)
  - Conditional Output Box: corresponde a las salidas de Mealy, que se activaran unicamente cuando se de la condición

FSM y ASM



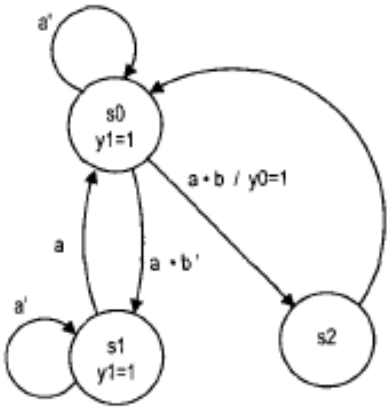
(a) Node

mo: Moore output
me: Mealy output

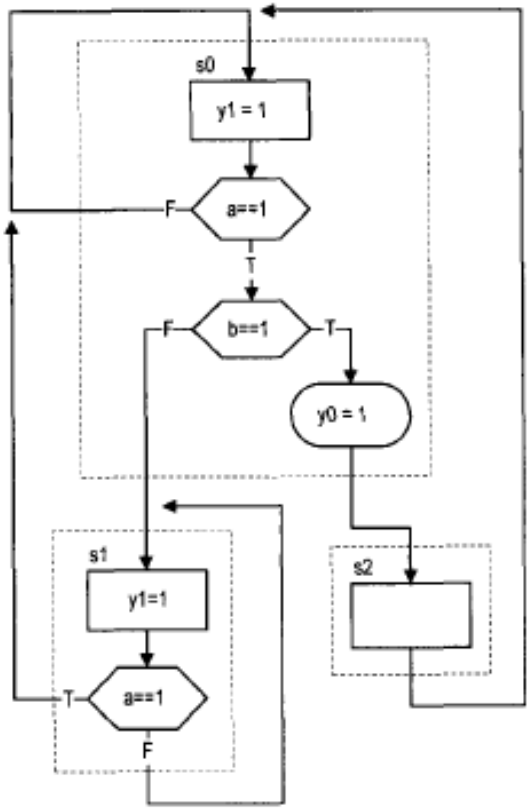


(b) ASM block

Ejemplo: FSM y ASM



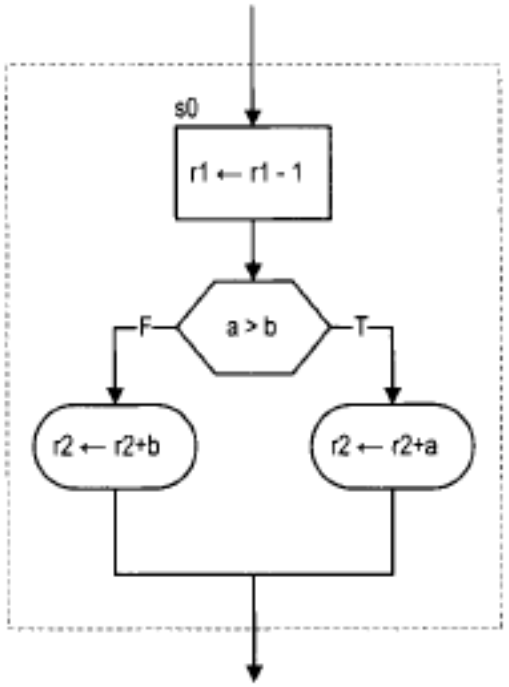
(a) State diagram

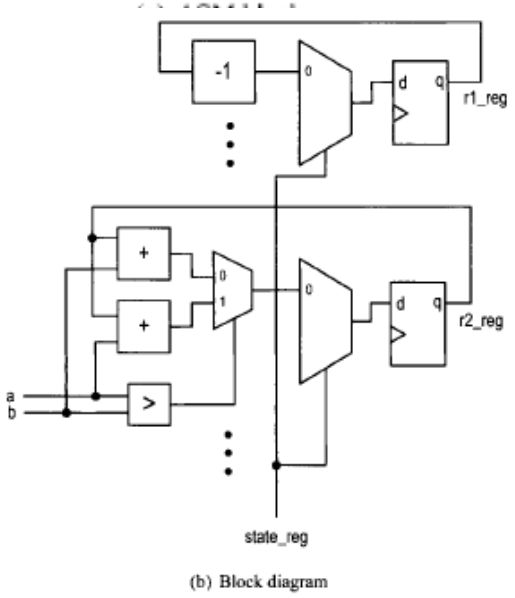


(b) ASM chart

ASMD

- Usaremos ASM para especificar estos sistemas mas complejos descritos con RTL, y los llamaremos ASMD (ASM con DataPath)
- Las operaciones RT se ubican en el mismo lugar que ubicábamos las salidas (Moore y Mealy)
- Veamos un nodo ASMD y el circuito combinacional que lo implementa.
- Algunas observaciones importantes:
  - Estamos mezclando salidas de Moore con salidas de Mealy. Esto también se puede hacer con los FSM
  - Los registros se actualizan cuando el FSMD sale del bloque!

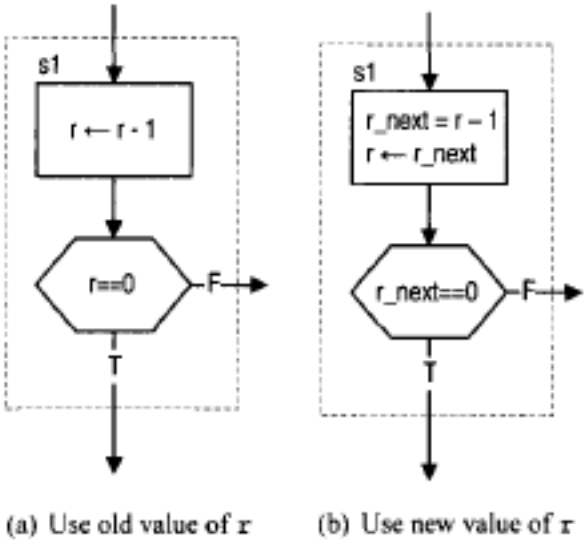




(b) Block diagram

Delayed Store

- Debido a que los registros se actualizan al salir del bloque, hay que tener mucho cuidado cuando especificamos condiciones que dependen de los valores almacenados.



(a) Use old value of r (b) Use new value of r

- La inscripción r←r-1 significa:

  r_next=r_reg-1

  r_reg ←r_next cuando se de el flanco del reloj

FSMD: Finite State Machine con Data Path

- Control Path, FSM conteniendo:
  - Registro de estado
  - Salida
  - Siguiente Estado
- Data Path, realiza las operaciones RT y contiene:
  - Registros para los datos
  - Unidades funcionales donde se computan las operaciones
  - Ruteo entre los datos y las unidades funcionales
- Son dos tipos de sistemas secuenciales pero ambos están controlados por el mismo reloj.

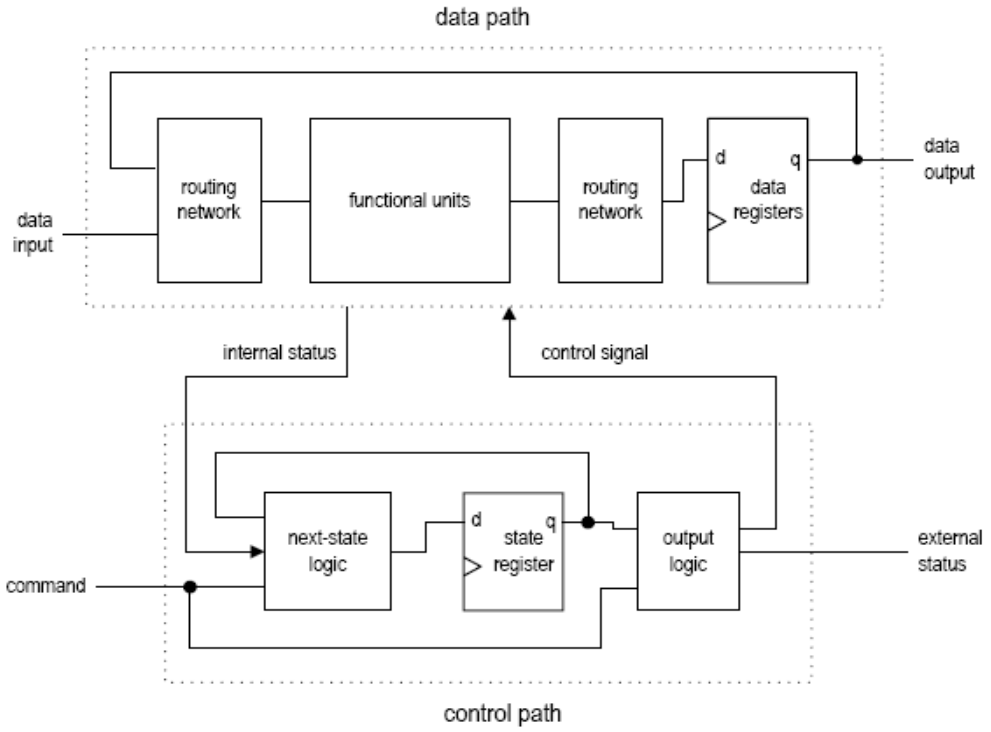


Figure 6.5 Block diagram of an FSMD.

Desarrollo de código: circuito antirebote.

- Vamos a usar el mismo circuito que se usó en secuenciales II, pero ahora utilizaremos la metodología RT para diseñarlo.
- Recordemos entonces ….

Circuito Antirrebote

- Como el switch 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.

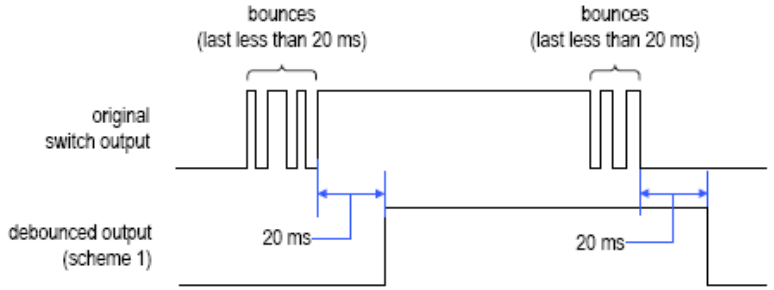
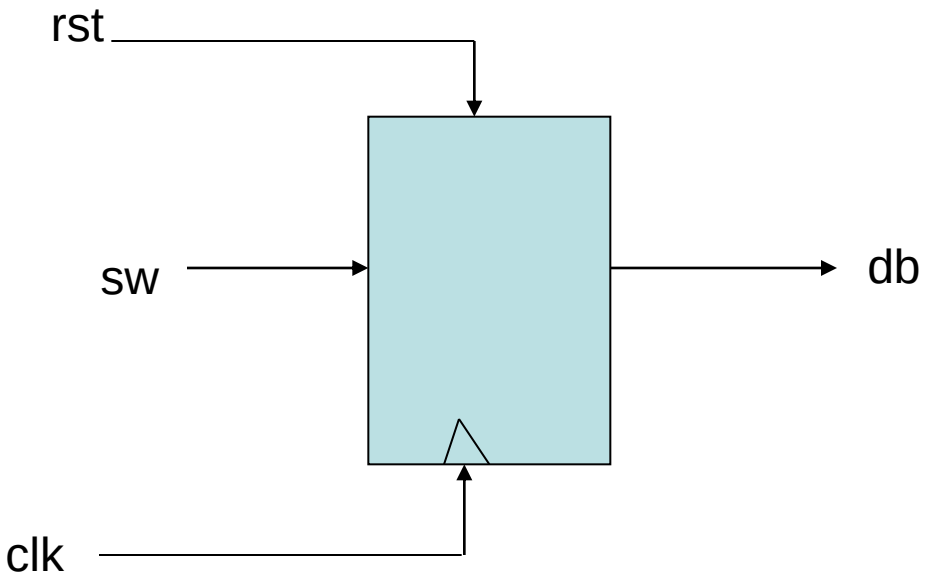


Figure 5.8 Original and debounced waveforms.

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 Antirrebote: Diagrama de Estados (FSM)

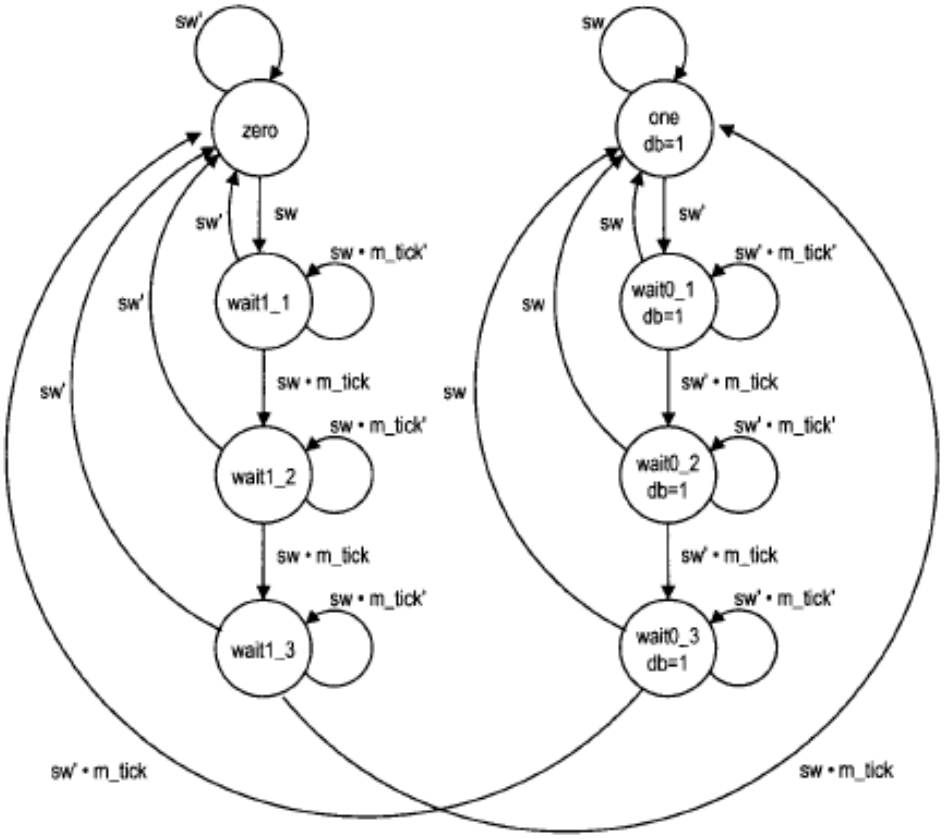


Figure 5.9 State diagram of a debouncing circuit.

Código Circuito Antirrebote

```
module db_fsm
   (
    input wire clk, reset,
    input wire sw,
    output reg db
   );

   // symbolic state declaration
   localparam  [2:0]
               zero    = 3'b000,
               wait1_1 = 3'b001,
               wait1_2 = 3'b010,
               wait1_3 = 3'b011,
               one     = 3'b100,
               wait0_1 = 3'b101,
               wait0_2 = 3'b110,
               wait0_3 = 3'b111;

   // number of counter bits (2^N * 20ns = 10ms tick)
   localparam N =19;

   // signal declaration
   reg [N-1:0] q_reg;
   wire [N-1:0] q_next;
   wire m_tick;
   reg [2:0] state_reg, state_next;
```

```
// body

   //=============================================
   // counter to generate 10 ms tick
   //=============================================
   always @(posedge clk)
      q_reg <= q_next;
   // next-state logic
   assign q_next = q_reg + 1;
   // output tick
   assign m_tick = (q_reg==0) ? 1'b1 : 1'b0;

   //=============================================
   // debouncing FSM
   //=============================================
   // state register
    always @(posedge clk, posedge reset)
      if (reset)
         state_reg <= zero;
      else
         state_reg <= state_next;
```

Código Circuito Antirrebote

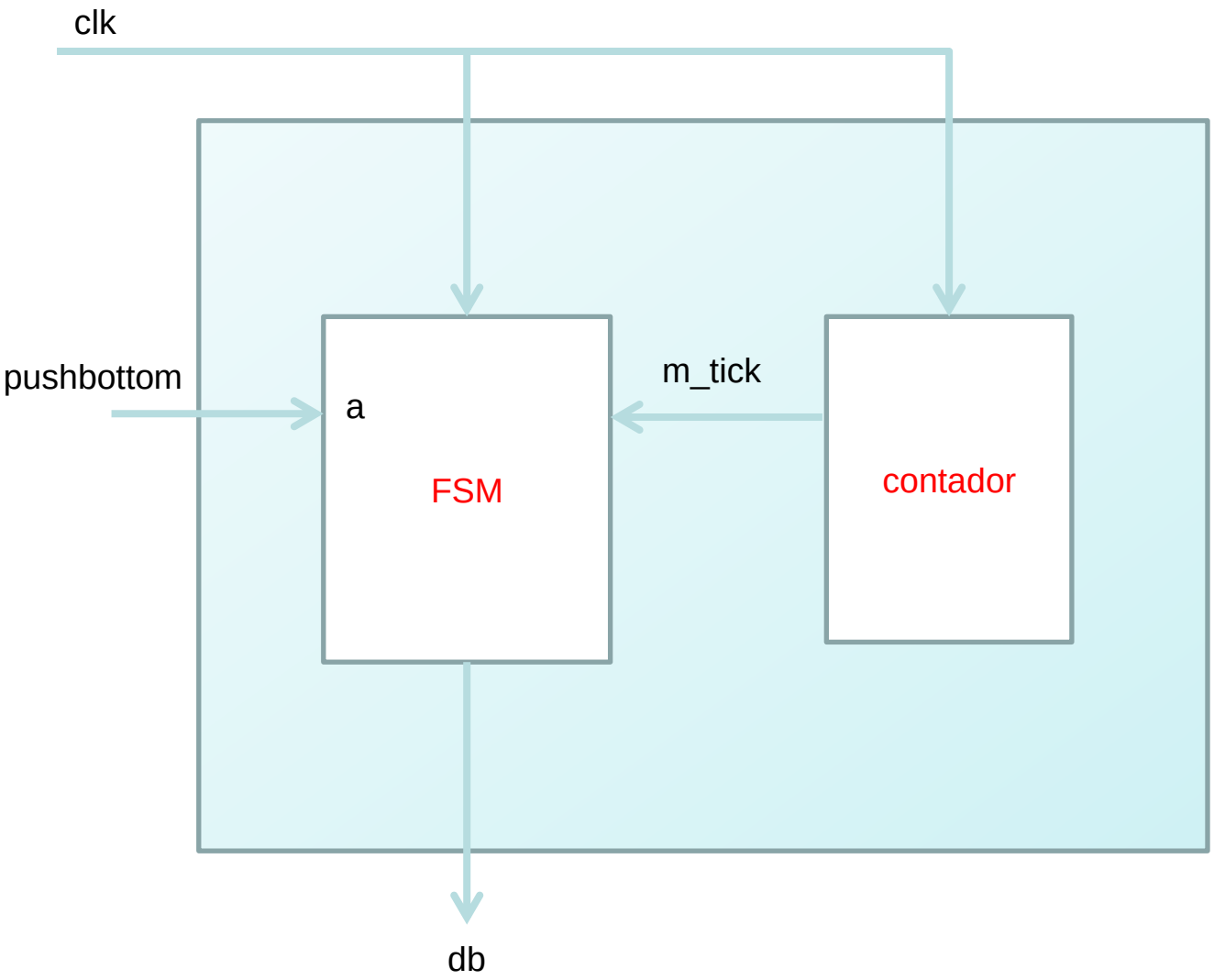
```
// next-state logic and output logic
   always @*
   begin
      state_next = state_reg;  // default state: the same
      db = 1'b0;               // default output: 0
      case (state_reg)
         zero:
            if (sw)
               state_next = wait1_1;
         wait1_1:
            if (~sw)
               state_next = zero;
            else
               if (m_tick)
                  state_next = wait1_2;
         wait1_2:
            if (~sw)
               state_next = zero;
            else
               if (m_tick)
                  state_next = wait1_3;
         wait1_3:
            if (~sw)
               state_next = zero;
            else
               if (m_tick)
                  state_next = one;
```

```
one:
            begin
              db = 1'b1;
              if (~sw)
                 state_next = wait0_1;
            end
         wait0_1:
            begin
              db = 1'b1;
              if (sw)
                 state_next = one;
              else
                if (m_tick)
                   state_next = wait0_2;
            end
         wait0_2:
            begin
              db = 1'b1;
              if (sw)
                 state_next = one;
              else
                if (m_tick)
                   state_next = wait0_3;
            end
```

```
wait0_3:
            begin
              db = 1'b1;
              if (sw)
                 state_next = one;
              else
                if (m_tick)
                   state_next = zero;
            end
         default: state_next = zero;
      endcase
   end

endmodule
```

Antirrebote



Debouncing, con la técnica RT

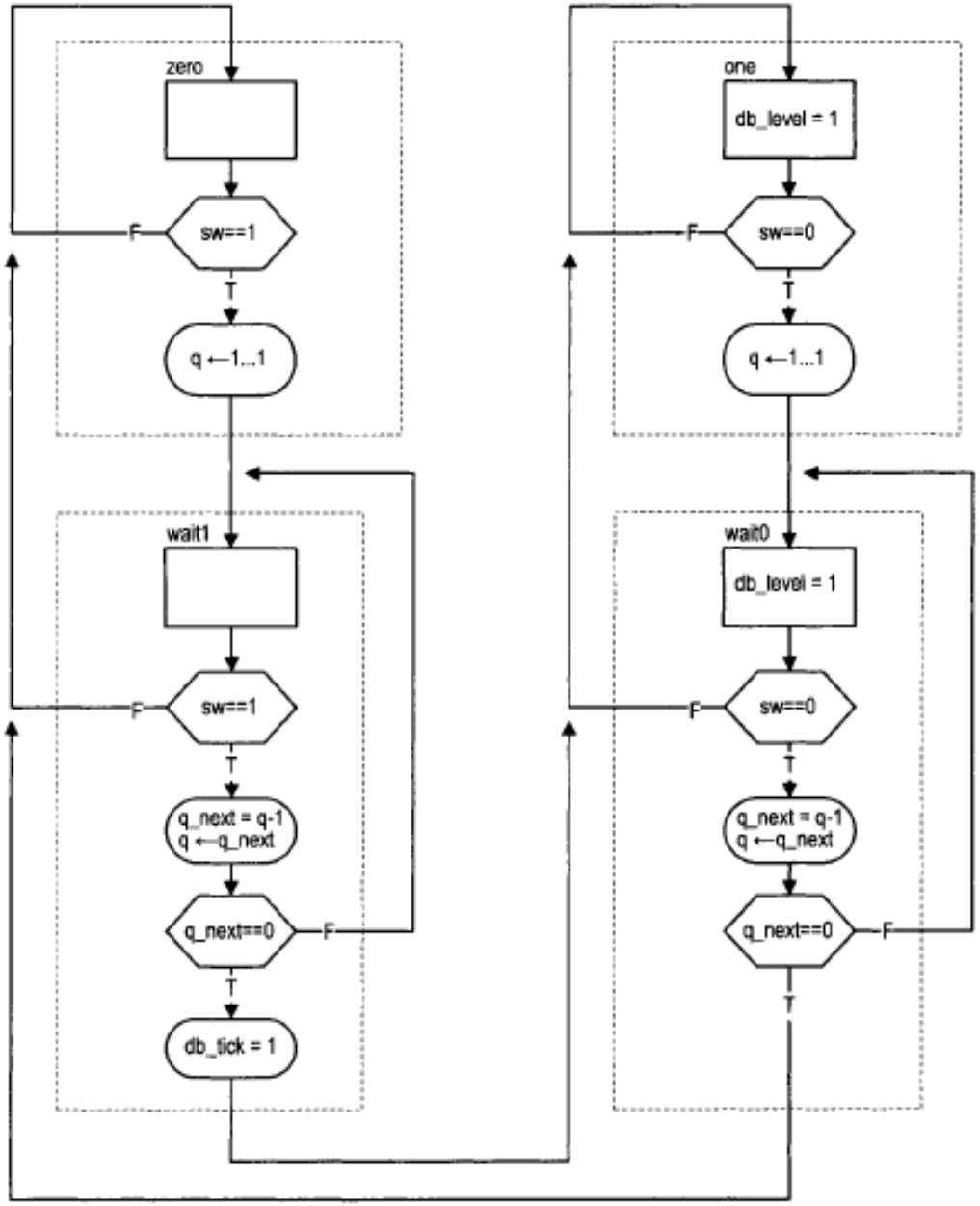
- En la implementación que tenemos, hay un contador y una FSM. El contador genera ticks cada 10mseg, y esta señal es una entrada del FSM➔ por lo tanto, cuando la FSM está en el estado wait1_1 o wait0_1, no sabe exactamente cuantos mseg pasaron desde que la señal sw se puso a 0 o a 1. Es un valor que está entre 0 y 10 mseg. Por lo tanto, el tiempo total que se espera a la estabilización de la señal es variable
- La siguiente implementación, intenta resolver ese problema.

Debouncing

- Vamos a utilizar la FSM para controlar la inicialización del timer y obtener el intervalo exacto.
- Para ello se agrega una nueva señal, db_tick que se activa por un ciclo en el flanco ascendente del reloj

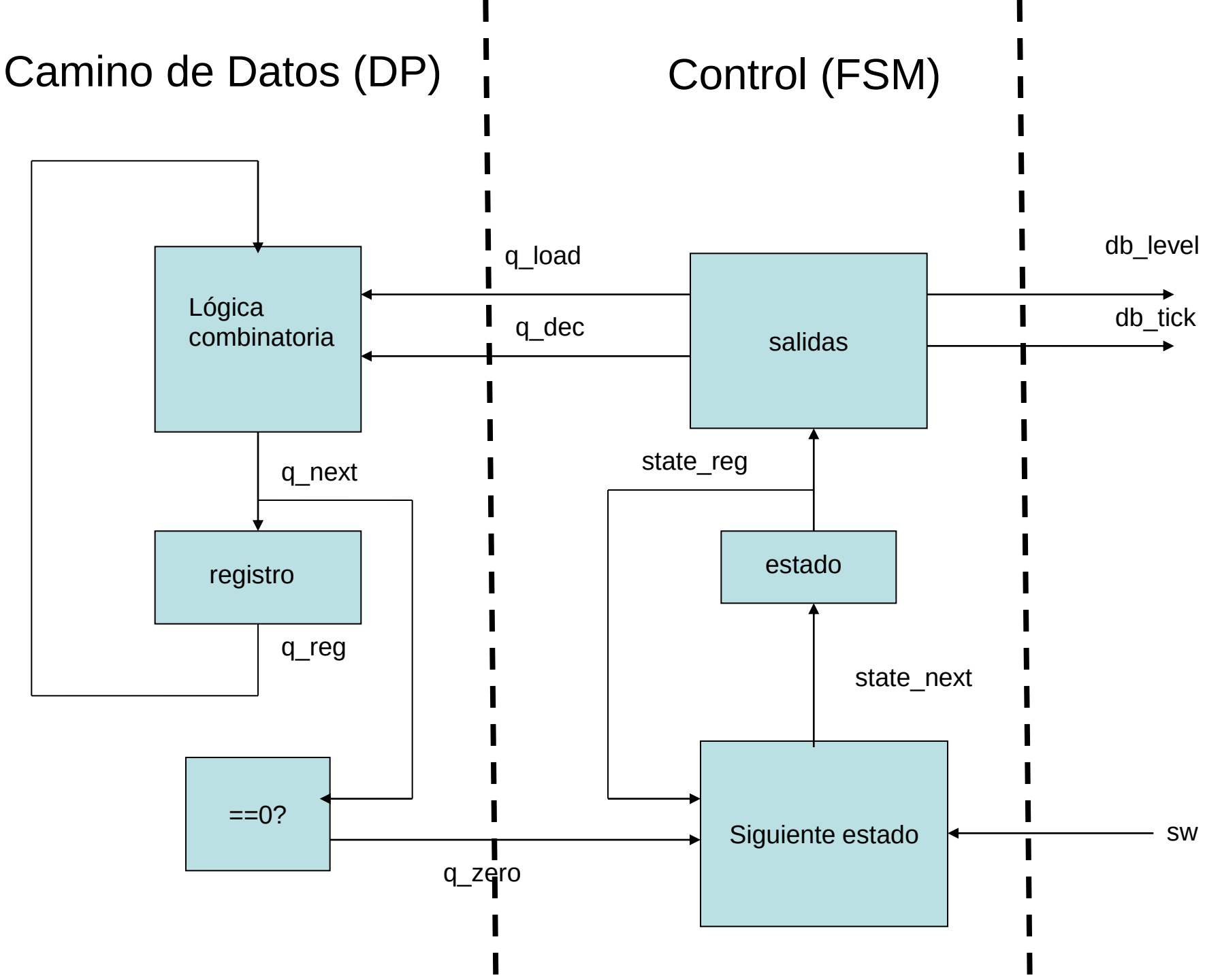
Debouncing

- Es muy parecido a lo que teníamos pero ahora el contador q está dentro del FSM.
- En el estado cero, el contador se inicializa a su valor máximo.
- En el estado uno, si el sw se ha apretado, se decrementa en cada ciclo.
- Cuando el contador llega a cero, se activa la señal db_tick



Desarrollo de código

- Código separado para el control y el camino de datos
- Camino de Datos:
  - Hay que identificar los elementos principales del camino de datos a partir del ASMD.
  - Estos son los registros y sus operaciones que expresamos con la metodología RTL: en nuestro caso un contador que:
    - Se inicializa a un valor
    - Se decrementa
    - Activa una señal cuando llega a cero
  - Camino de Datos formado por:
    - Un registro de 21 bits (q_reg)
    - Un comparador para detección de cero (q_cero)
    - Lógica del siguiente estado en función de las señales q_load y q_dec


Camino de Datos (DP)
Control (FSM)
Lógica combinatoria
registro
==0?
salidas
estado
Siguiente estado
q_load
q_dec
q_next
q_reg
q_zero
state_reg
state_next
db_level
db_tick
sw

Código

```
module debounce_explicit
    (
     input wire clk, reset,
     input wire sw,
     output reg db_level, db_tick
    );

    // symbolic state declaration
    localparam  [1:0]
                zero  = 2'b00,
                wait0 = 2'b01,
                one   = 2'b10,
                wait1 = 2'b11;

    // number of counter bits (2^N * 20ns = 40ms)
    localparam N=21;

    // signal declaration
    reg [1:0] state_reg, state_next;
    reg [N-1:0] q_reg;
    wire [N-1:0] q_next;
    wire q_zero;
    reg q_load, q_dec;

    // body
```

Código

```
    // fsmd state & data registers
     always @(posedge clk, posedge reset)
        if (reset)
           begin
30            state_reg <= zero;
              q_reg <= 0;
           end
        else
           begin
35            state_reg <= state_next;
              q_reg <= q_next;
           end

    // FSMD data path (counter) next-state logic
40  assign q_next = (q_load) ? {N{1'b1}} :    // load 1..1
                    (q_dec)  ? q_reg - 1 :    // decrement
                               q_reg;
    // status signal
    assign q_zero = (q_next==0);
45
```

Código

```
// FSMD control path next-state logic
always @*
begin
   state_next = state_reg;  // default state: the same
   q_load = 1'b0;           // default output: 0
   q_dec = 1'b0;            // default output: 0
   db_tick = 1'b0;          // default output: 0
   case (state_reg)
      zero:
         begin
            db_level = 1'b0;
            if (sw)
               begin
                  state_next = wait1;
                  q_load = 1'b1;
               end
         end
      wait1:
         begin
            db_level = 1'b0;
            if (sw)
               begin
                  q_dec = 1'b1;
                  if (q_zero)
                     begin
                        state_next = one;
                        db_tick = 1'b1;
                     end
               end
            else // sw==0
               state_next = zero;
         end
      one:
         begin
            db_level = 1'b1;
            if (~sw)
               begin
                  state_next = wait0;
                  q_load = 1'b1;
               end
         end
      wait0:
         begin
            db_level = 1'b1;
            if (~sw)
               begin
                  q_dec = 1'b1;
                  if (q_zero)
                     state_next = zero;
               end
            else // sw==1
               state_next = one;
         end
      default: state_next = zero;
   endcase
end

endmodule
```

Reporte de Síntesis

Found 21-bit subtractor for signal <q_next$addsub0000> created at line 41.
Found 21-bit register for signal <q_reg>.
Summary:
inferred 1 Finite State Machine(s).
inferred 21 D-type flip-flop(s).
inferred 1 Adder/Subtractor(s).
Unit <debounce_explicit> synthesized.

Alternativa de Diseño

- Dos alternativas de diseño:
  - Separar en el código los componentes del camino de datos, como acabamos de hacer.
  - Embeber las operaciones RT dentro del control. En este caso, las operaciones RT se listan en los estados correspondientes de la FSM.
    - En este caso, nos ahorramos las señales de control q_load y q_dec
    - Pero, hacemos que la herramienta de síntesis infiera el data path. Esto puede o no ser conveniente.

Alternativa II para el código

```
module debounce
   (
    input wire clk, reset,
    input wire sw,
    output reg db_level, db_tick
   );

   // symbolic state declaration
   localparam  [1:0]
               zero  = 2'b00,
               wait0 = 2'b01,
               one   = 2'b10,
               wait1 = 2'b11;

   // number of counter bits (2^N * 20ns = 40ms)
   localparam N=21;

   // signal declaration
   reg [N-1:0] q_reg, q_next;
   reg [1:0] state_reg, state_next;
```

```
// body
  // fsmd state & data registers
   always @(posedge clk, posedge reset)
      if (reset)
         begin
            state_reg <= zero;
            q_reg <= 0;
         end
      else
         begin
            state_reg <= state_next;
            q_reg <= q_next;
         end
```

Código (cont)

```
// next-state logic & data path functional units/routing
  always @*
  begin
     state_next = state_reg;   // default state: the same
     q_next = q_reg;           // default q: unchnaged
     db_tick = 1'b0;           // default output: 0
     case (state_reg)
        zero:
           begin
              db_level = 1'b0;
              if (sw)
                 begin
                    state_next = wait1;
                    q_next = {N{1'b1}}; // load 1..1
                 end
           end
```

```
wait1:
           begin
              db_level = 1'b0;
              if (sw)
                 begin
                    q_next = q_reg - 1;
                    if (q_next==0)
                       begin
                          state_next = one;
                          db_tick = 1'b1;
                       end
                 end
              else // sw==0
                 state_next = zero;
           end
```

Código (cont)

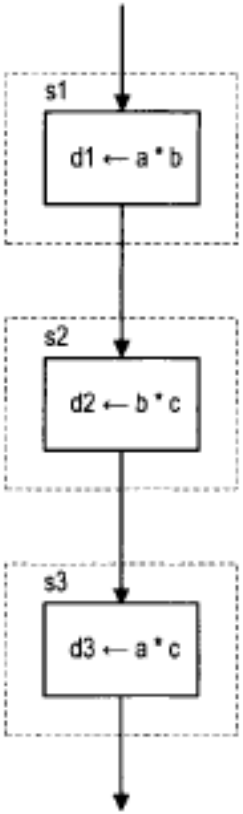
```
one:
        begin
          db_level = 1'b1;
          if (~sw)
            begin
              state_next = wait0;
              q_next = {N{1'b1}}; // load 1..1
            end
        end
```

```
wait0:
        begin
          db_level = 1'b1;
          if (~sw)
            begin
              q_next = q_reg - 1;
              if (q_next==0)
                state_next = zero;
            end
          else // sw==1
            state_next = one;

        end
      default: state_next = zero;
    endcase
  end

endmodule
```

Otro ejemplo




```
case (state_reg)
   s1:
     begin
       d1_next = a * b;
       ...
     end
   s2:
     begin
       d2_next = b * c;
       ...
     end
   s3:
     begin
       d3_next = a * c;
       ...
     end
```

Aquí se infieren tres multiplicadores
En el data path

```
case (state_reg)
   s1:
     begin
       in1 = a;
       in2 = b;
       d1_next = m_out;
       ...
     end
   s2:
     begin
       in1 = b;
       in2 = c;
       d2_next = m_out;
       ...
     end
   s3:
     begin
       in1 = a;
       in2 = c;
       d3_next = m_out;
       ...
```

Aquí solo usamos un multiplicador

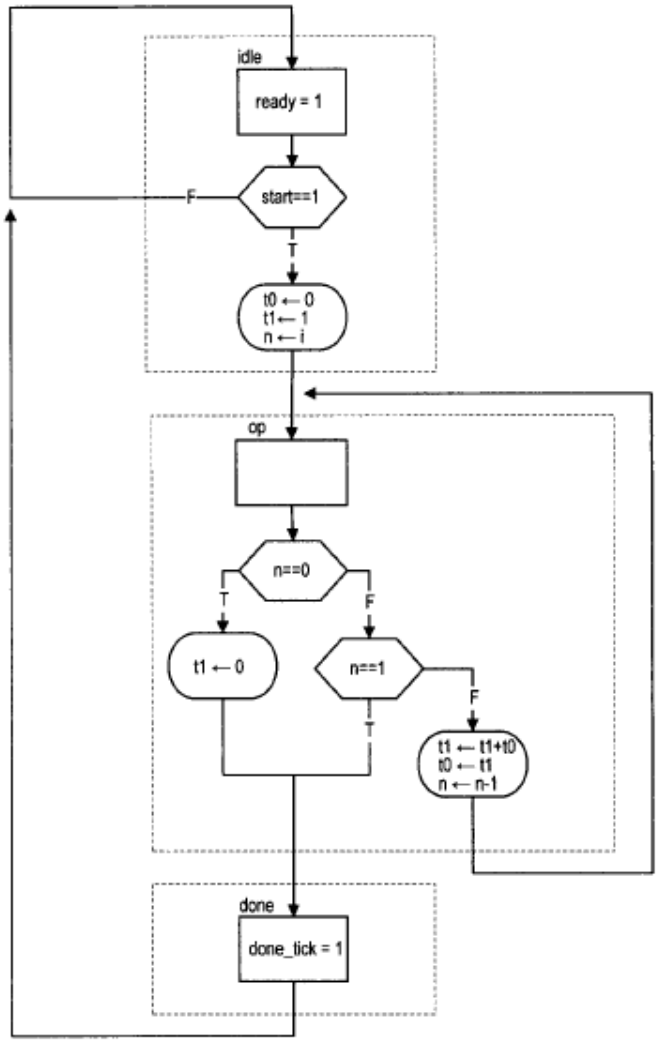
ASSIGN m_out=in1*in2

Fibonacci

```
Int fibonacci (int i) {
      int t0 =0;
      int t1 =1;
      int tmp =0;
      int n =i;
      while (n>2) {
                        tmp=t1;
                        t1=t1 + t0;
                        t0=tmp;
                        n=n-1;
      }
      If (n==0) t1=0;
}
```

$$fib(i) = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i = 1 \\ fib(i-1) + fib(i-2) & \text{if } i > 1 \end{cases}$$

Fibonacci



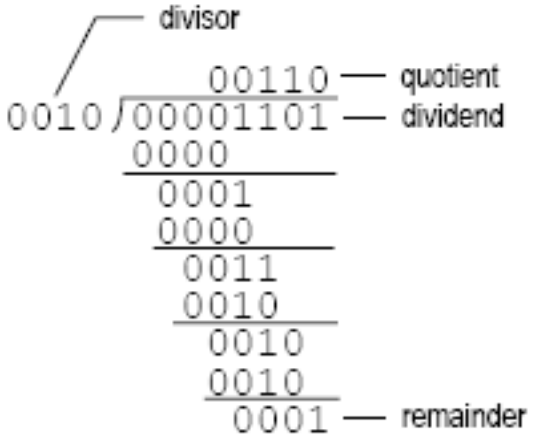
$$fib(i) = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i = 1 \\ fib(i-1) + fib(i-2) & \text{if } i > 1 \end{cases}$$

- Se utilizan tres registros. Dos temporales para almacenar fib (i-1) y fib(i-2) y un índice (n)
- La entrada es i
- La salida es f
- Se incluyen otras señales:
  - Start
  - Ready
  - Done_tick

La División

1. Extender con ceros el dividendo (duplicar la longitud). Alinear el divisor a la izquierda del dividendo.
2. Si los bits del dividendo son mayores o iguales que el divisor, restar el divisor del dividendo, y hacer 1 el bit del cociente. Sino, mantener los bits del dividendo, el cociente es 0.
3. Agregar un bit al resultado anterior de la resta, y hacer shift del divisor una posición a la derecha.
4. Repetir los pasos 2 y 3 hasta que todos los bits del dividendo se hayan utilizado.

Observación: Lo que desplazaremos a la izquierda será el dividendo, y en cada iteración entrará por la derecha un bit del cociente. Cuando terminemos, el resto quedará en la parte alta y el cociente en la parte baja.



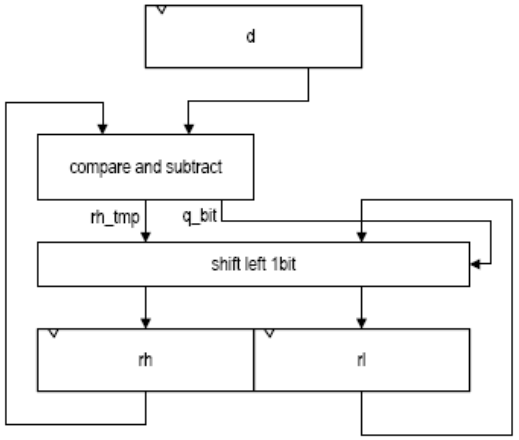
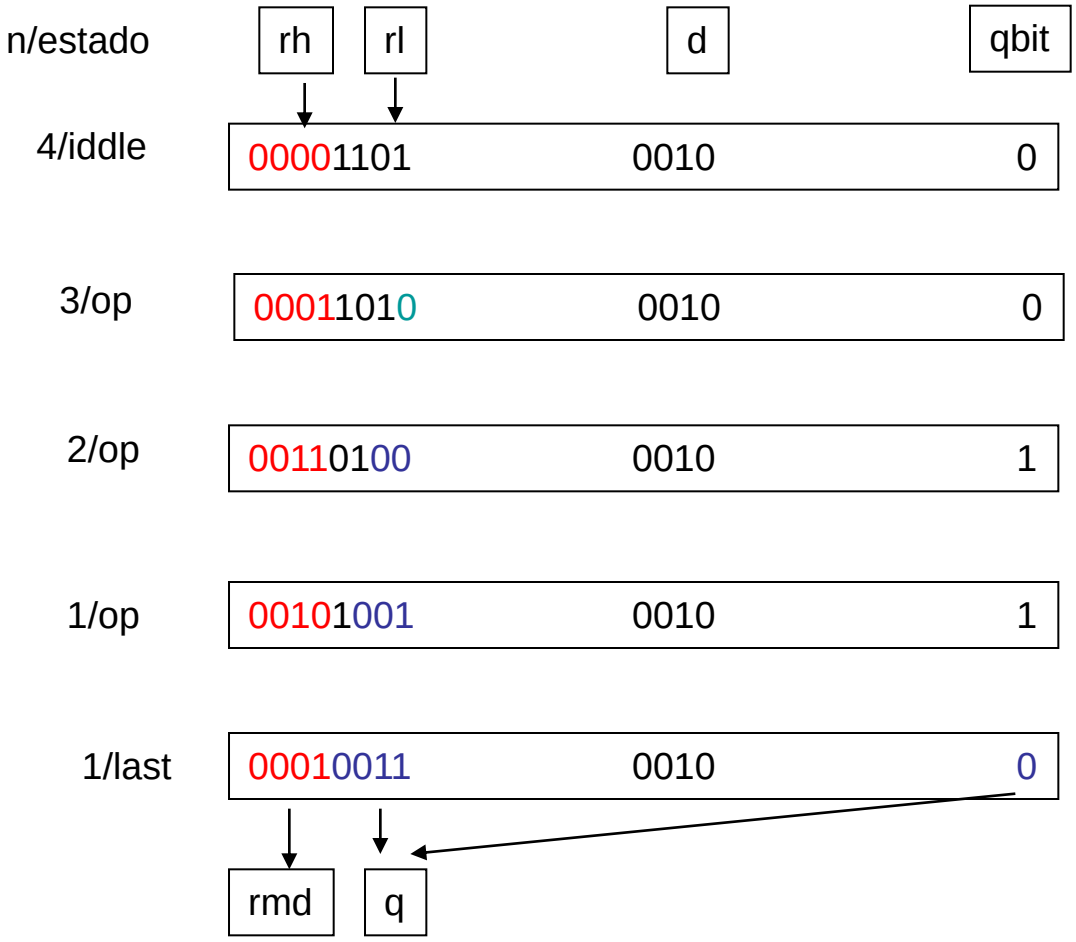


Figure 6.11 Sketch of division circuit's data path.

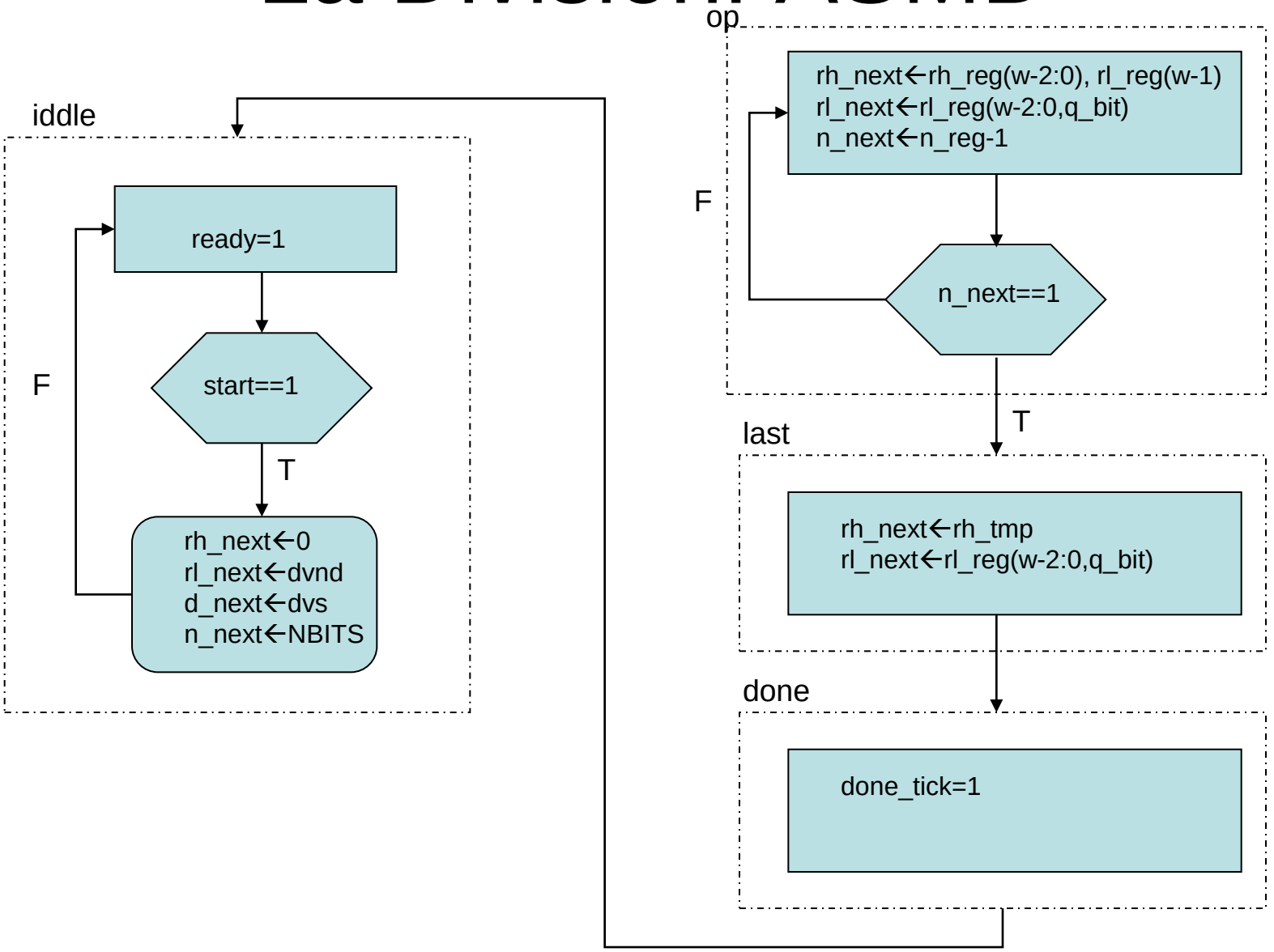
La División: Simulación



r_tmp

(0011) – (0010) = 0001

La División: ASMD



Aplicación: Reaction Timer

- Eye-hand coordination is the ability of the eyes and hands to work together to perform a task. A reaction timer circuit measures how fast a human hand can respond after a person sees a visual stimulus. This circuit operates as follows:
  1. The circuit has three input pushbuttons, corresponding to the **c l e a r** , **s t a r t** , and **s t o p** signals. It uses a single discrete LED as the visual stimulus and displays information on the seven-segment LED display.

Reaction Timer

2. A user pushes the c l e a r button to force the circuit to return to the initial state, in which the seven-segment LED shows a welcome message, "HI," and the stimulus LED is off.
3. When ready, the user pushes the start button to initiate the test. The seven-segment LED goes off.
4. After a random interval between 2 and 15 seconds, the stimulus LED goes on and the timer starts to count upward. The timer increases every millisecond and its value is displayed in the format of "0.000" second on the seven-segment LED.

Reaction Timer

5. After the stimulus LED goes on, the user should try to push the s t o p button as soon as possible. The timer pauses counting once the s t o p button is asserted. The seven segment LED shows the reaction time. It should be around 0.15 to 0.30 second for most people.
6. If the s t o p button is not pushed, the timer stops after 1 second and displays " 1 .000”.
7. If the s t o p button is pushed before the stimulus LED goes on, the circuit displays "9.999" on the seven-segment LED and stops.