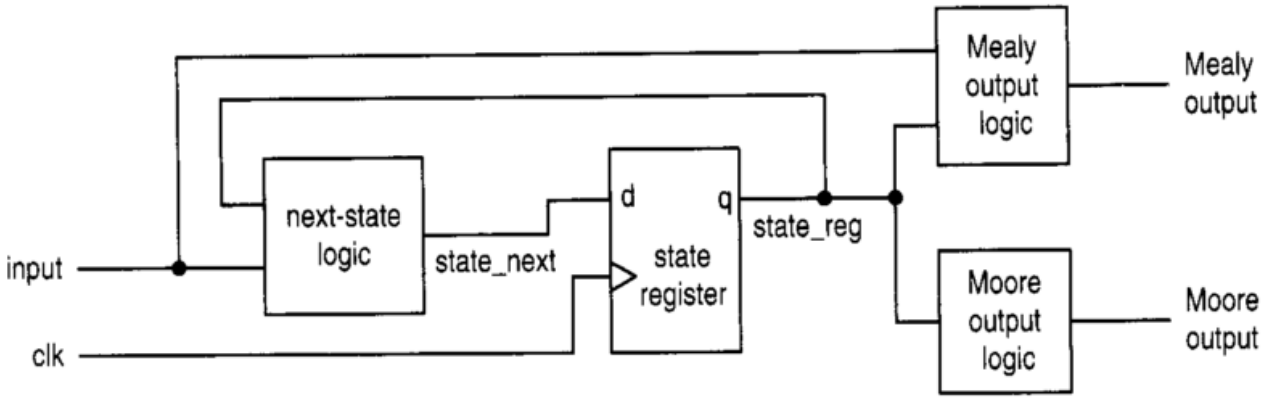
Circuitos Secuenciales en FPGA
Parte III: Finite State Machine con Datapath

Introducción a la Microfabricación y las FPGA

Instituto Balseiro

1 de Septiembre 2014

La clase pasada



- FSM: Seguir el mismo método de diseño sincrónico, dividiendo claramente el problema en 4 pasos.
- Especificación con diagramas de FSM o ASM.
- Seguir guidelines de codificación para cada una de las cuatro partes.
- Diseño jerárquico.

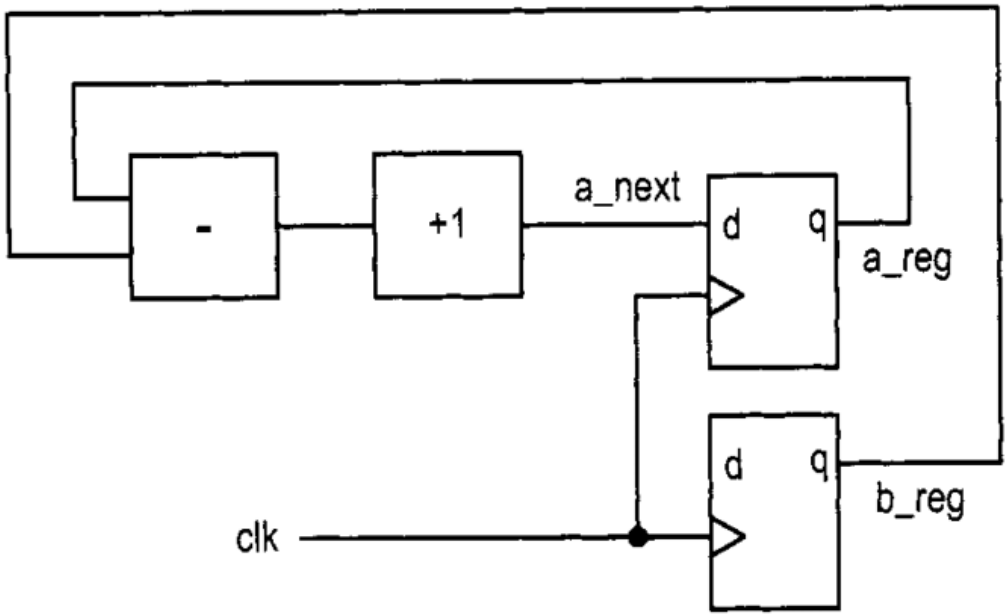
FSMD

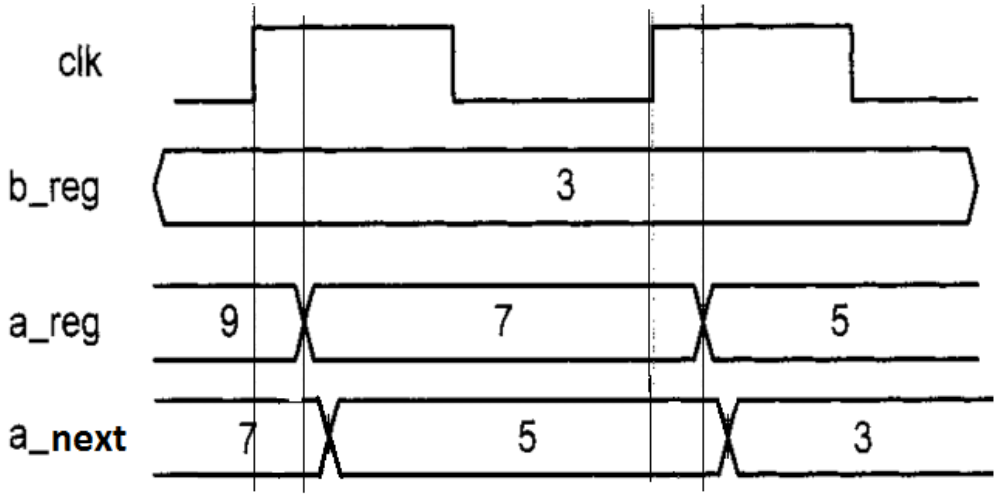
- Finite State Machine con Datapath. Combina una FSM y uno o varios circuitos regulares secuenciales.
  - *FSM o Control-path*: examina las entradas y el estado, y genera las señales de control para especificar las operaciones de los circuitos regulares secuenciales.
  - *Data path*: circuitos secuenciales regulares que hacen operaciones entre registros.
- Se usa para implementar sistemas descritos con la metodología de Register Transfer. Manipulaciones de datos entre registros.

Metodología Register Transfer

$r_{dest} \leftarrow f(r_{src1}, r_{src2}, ..., r_{srcn})$
Una operación RT puede implementarse con un circuito combinacional para $f()$, conectando la entrada y la salida de los registros.

Ejemplo
$a \leftarrow a - b + 1$
El resultado no se guarda en a hasta el proximo clk.
Está sincronizado.
$a_{next} \leftarrow a_{reg} - b_{reg} + 1$

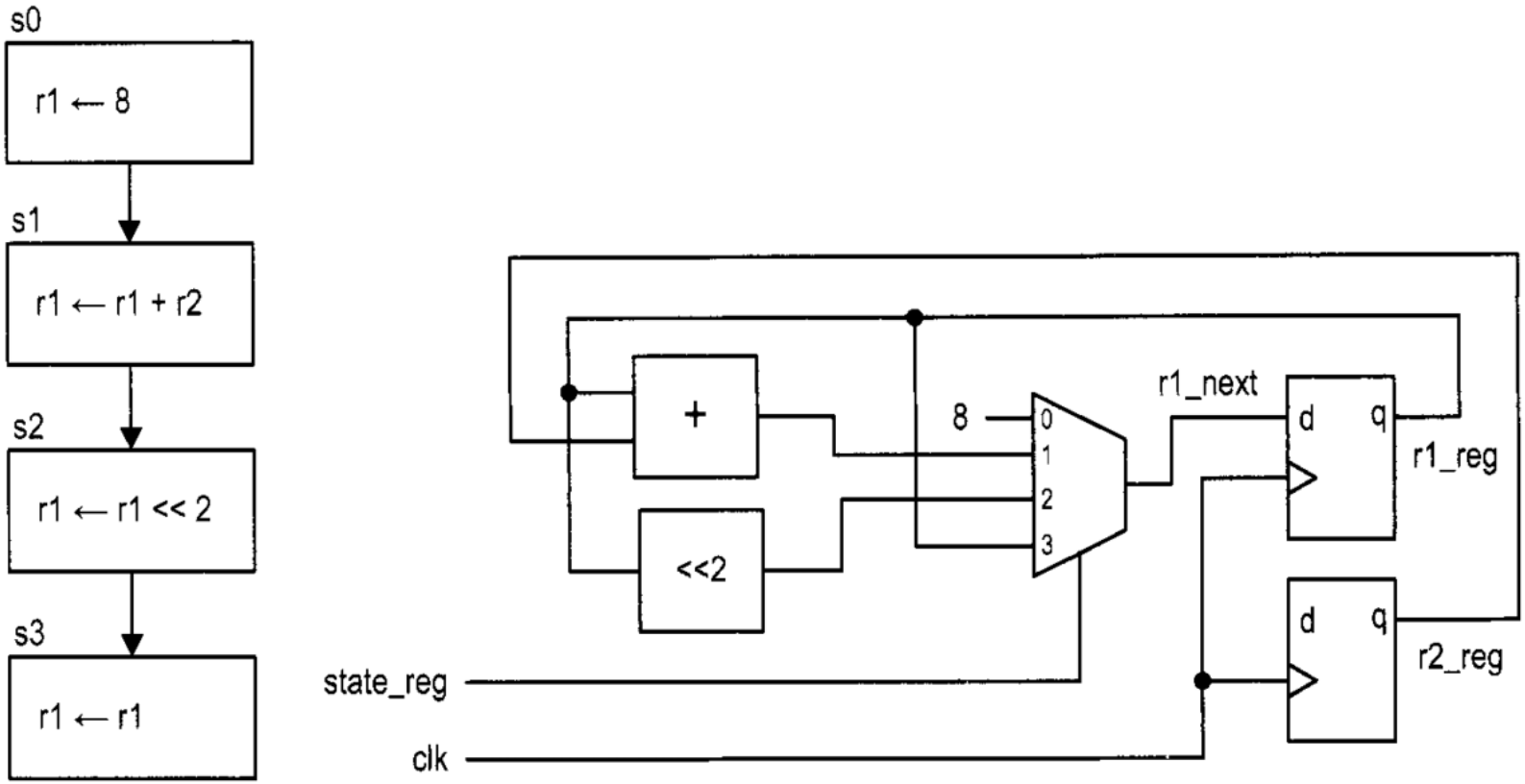




Representación: ASMD y FSMD

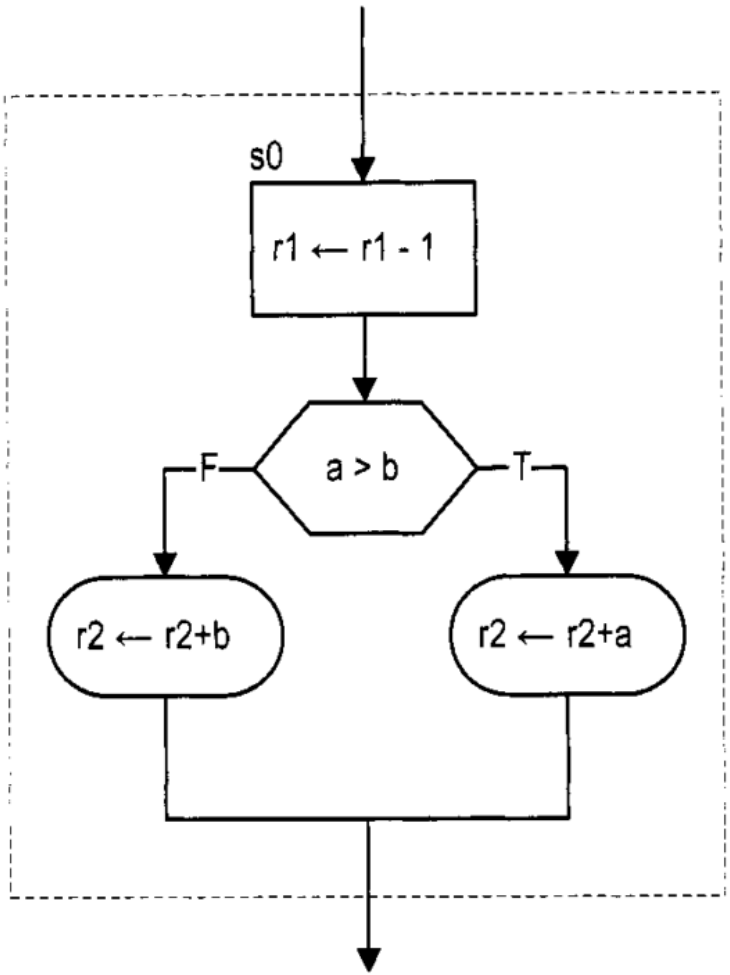
- Un circuito basado en la metodología RT tiene que especificar qué operaciones RT se van a hacer en cada paso (i.e, en cada estado).
- Diseño: extensión de ASM o FSM para incluir esta información. ASMD/FSMD: con Data Path.
- Las operaciones RT son tratadas como otro tipo de actividad, y se ponen donde van los outputs: o en los estados (Moore) o como dependiendo de los estados y las entradas (Mealy).

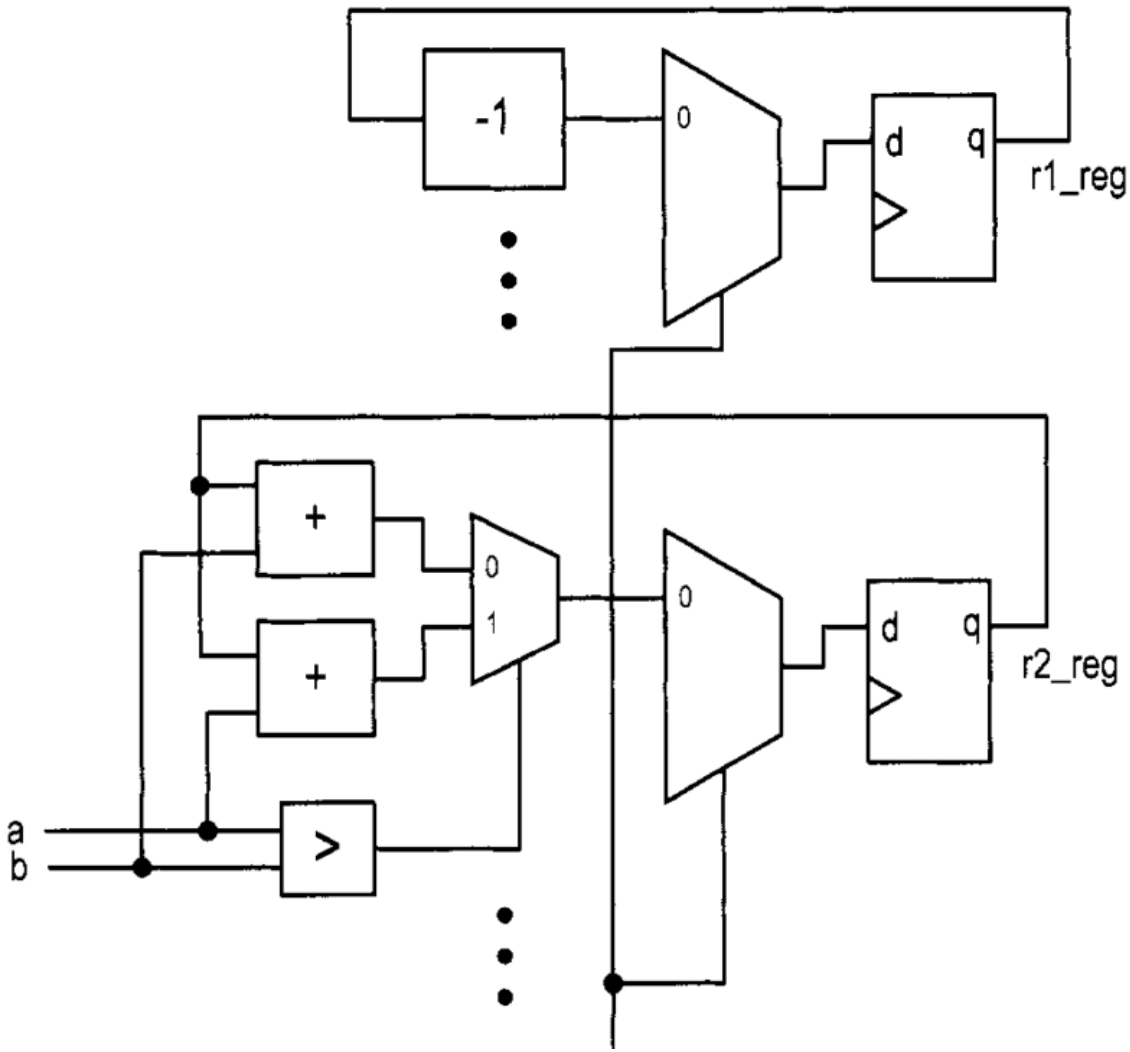
ASMD - Moore



- Se necesita **rutear** el resultado adecuado al registro.
- El **control** de las operaciones entre registros la hace el registro de estado, que es la salidad de la FSM.

ASMD - Mealy





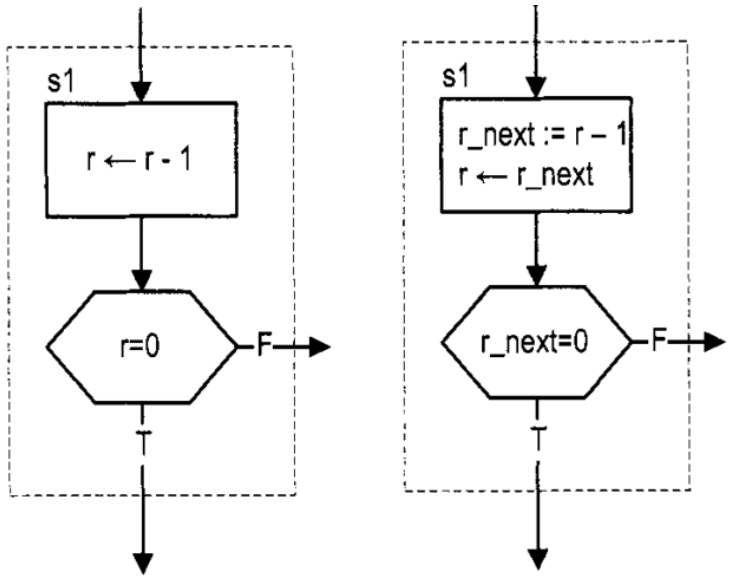
Diseño Sincrónico

Las operaciones RT en un ASMD están controladas por un clock, porque son transferencias entre registros. Entonces, el nuevo valor NO se actualiza hasta que se sale del bloque, i.e, se hace la trancision de estado.

$r \leftarrow r + 1$ es :

`r_next <= r_reg + 1`

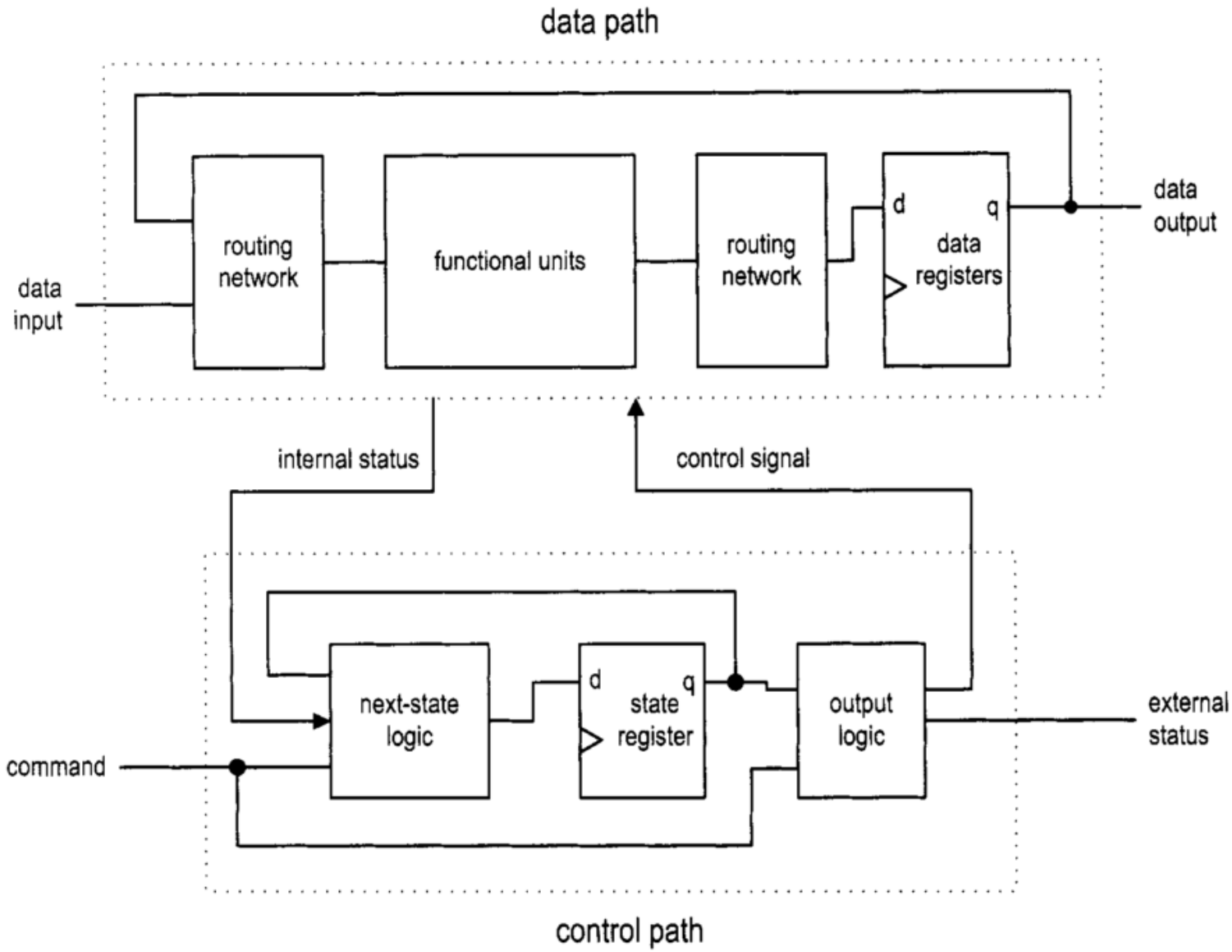
`r_reg <= r_next` al momento de hacer la transición.



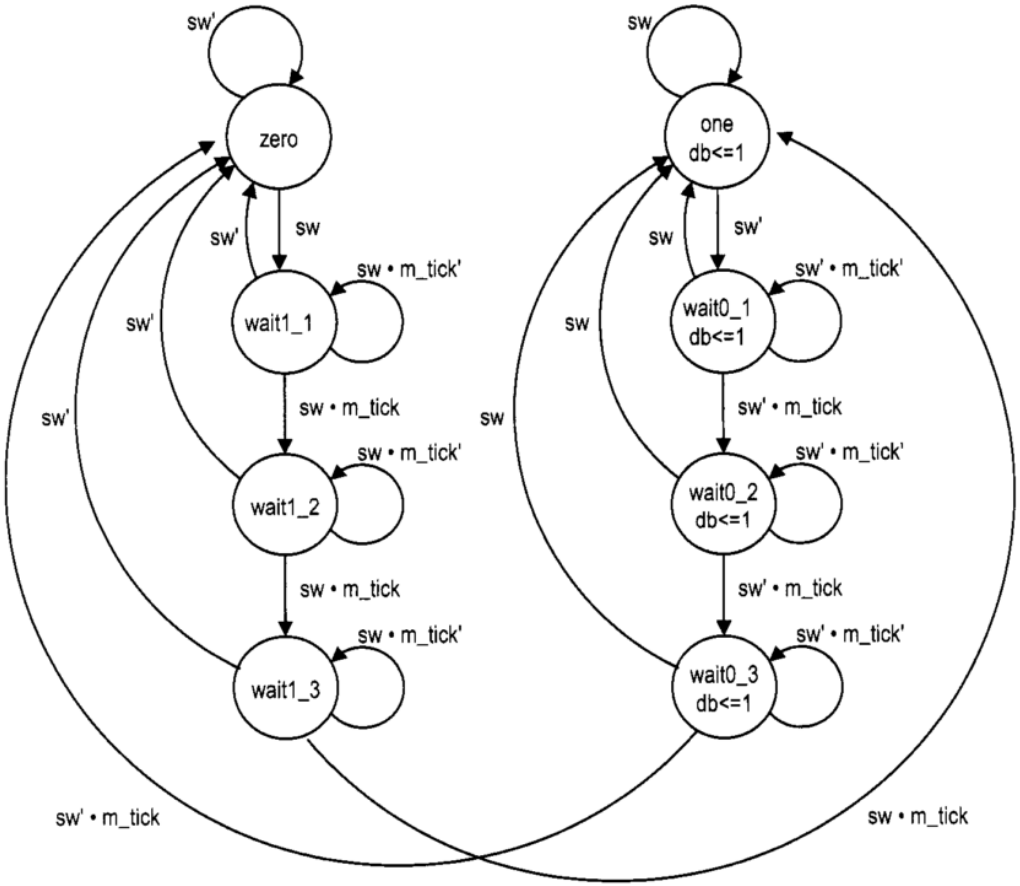
a) Usa el viejo valor de r

b) Usa el nuevo valor de r

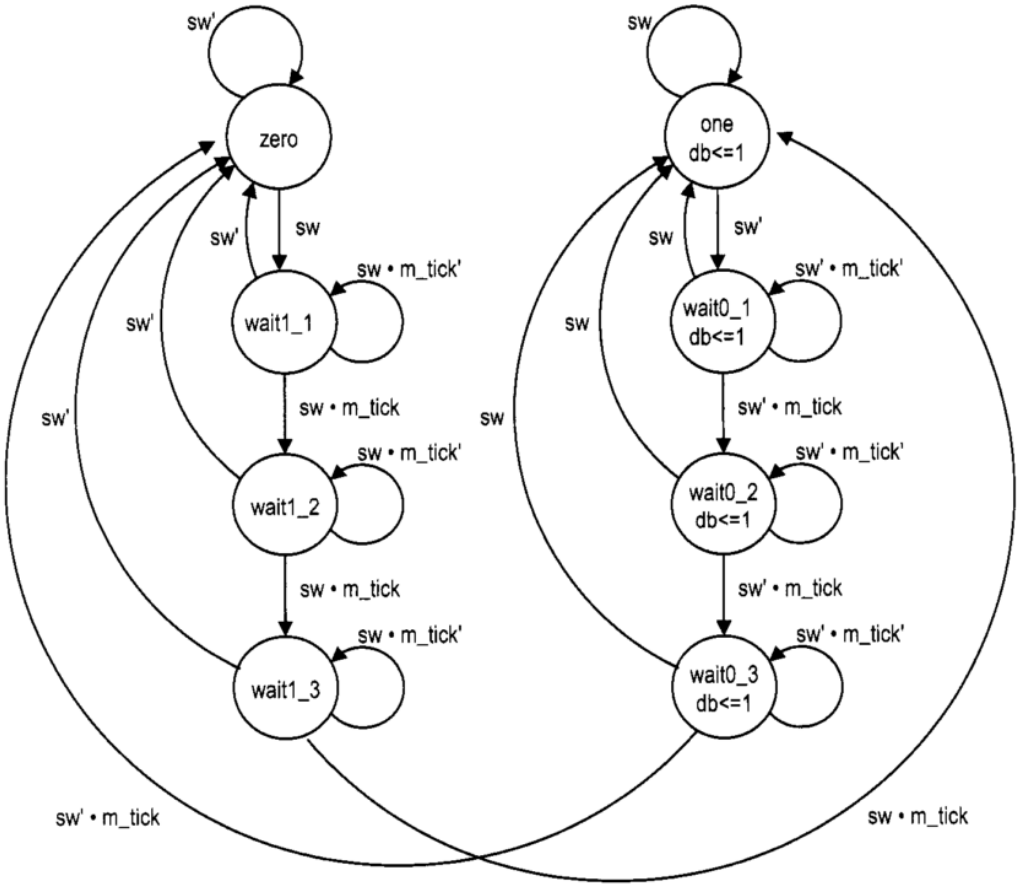
Diagrama de Bloques



Debouncer

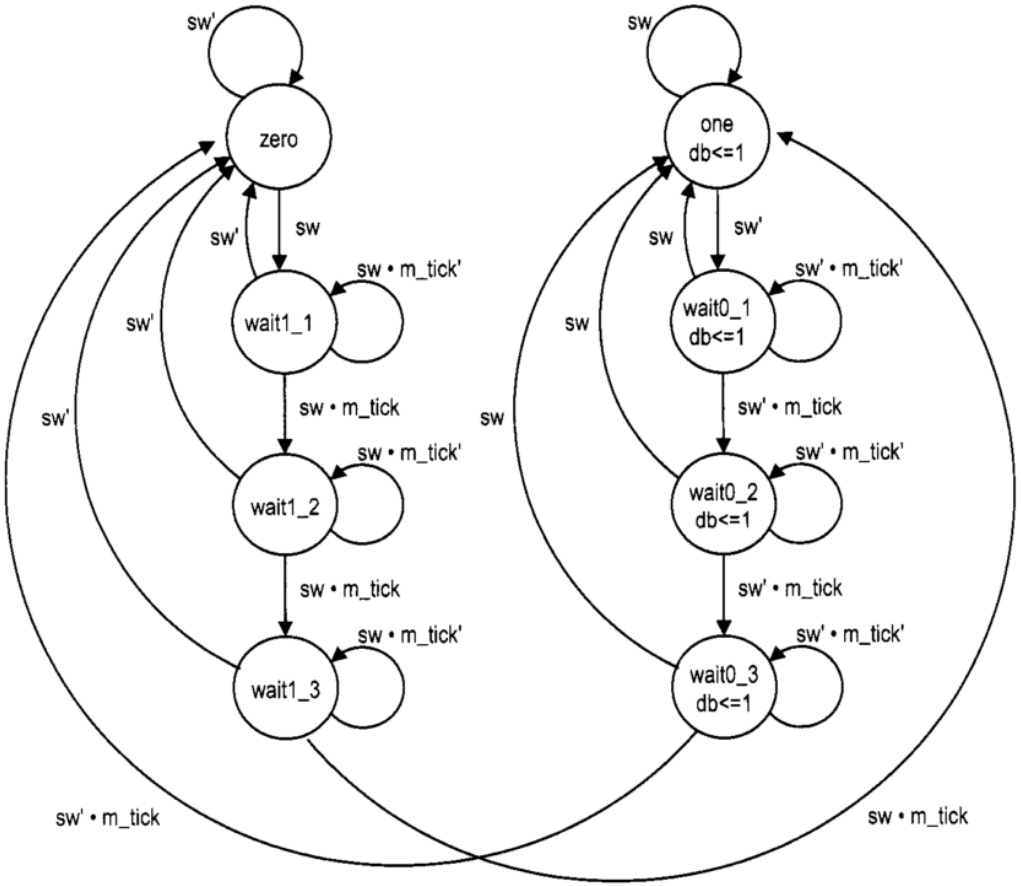

sw'
zero
sw
sw'
sw • m_tick'
wait1_1
sw'
sw • m_tick
sw • m_tick'
wait1_2
sw'
sw • m_tick
sw • m_tick'
wait1_3
sw' • m_tick
sw
one
db<=1
sw
sw'
sw' • m_tick'
wait0_1
db<=1
sw
sw' • m_tick
sw' • m_tick'
wait0_2
db<=1
sw
sw' • m_tick
sw' • m_tick'
wait0_3
db<=1
sw • m_tick

Debouncer



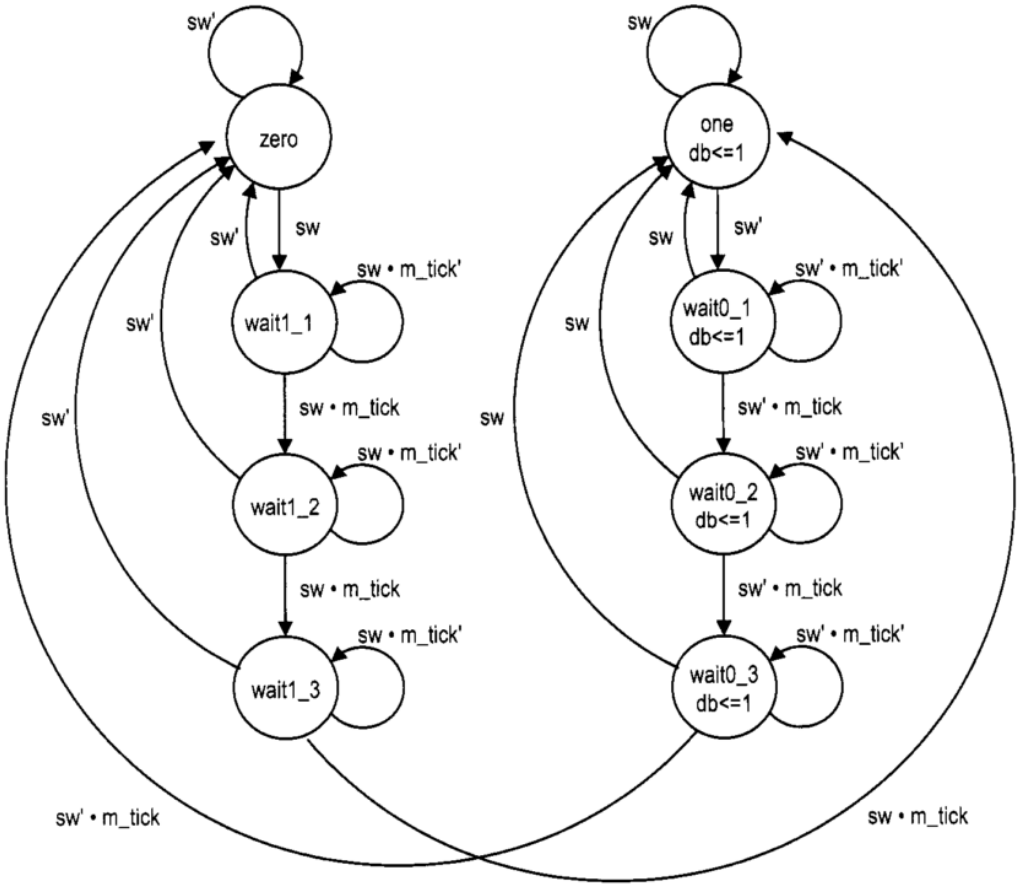
- Usa una FSM y un timer (sec. regular), pero no es metodología RT. La FSM no tiene ningún control del timer.

Debouncer



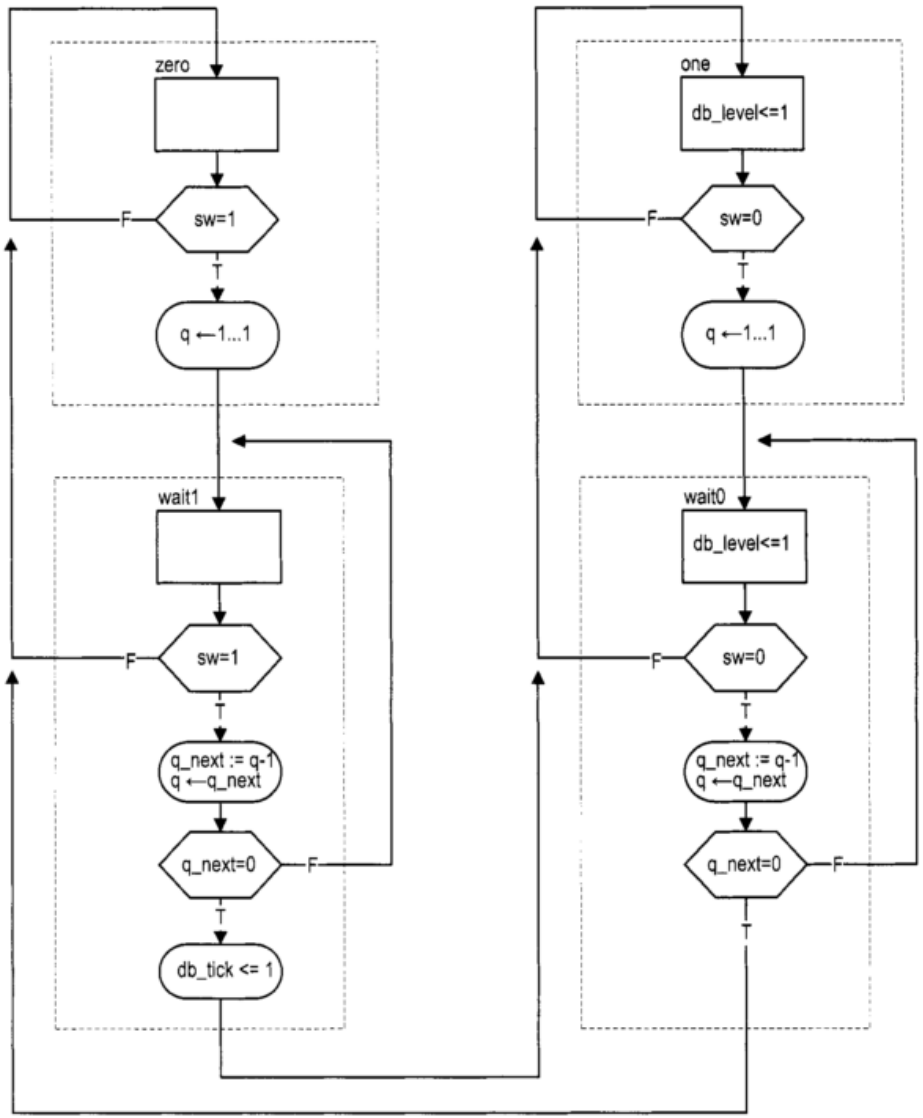
- Usa una FSM y un timer (sec. regular), pero no es metodología RT. La FSM no tiene ningún control del timer.
- Como en el estado `wait1_1` (idem `wait0_1`) la señal `m_tick` puede assertearse en cualquier momento, no se puede medir exactamente cuánto tiempo pasa desde que se estabiliza la señal.

Debouncer

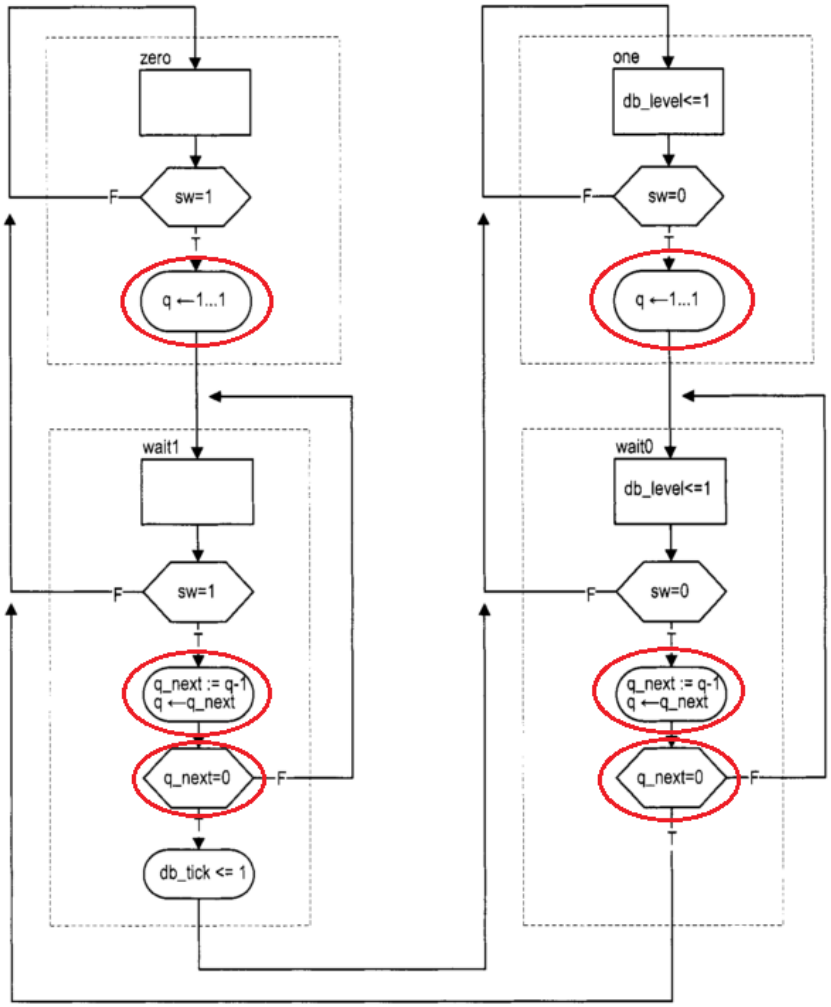


- Usa una FSM y un timer (sec. regular), pero no es metodología RT. La FSM no tiene ningún control del timer.
- Como en el estado `wait1_1` (idem `wait0_1`) la señal `m_tick` puede assertearse en cualquier momento, no se puede medir exactamente cuánto tiempo pasa desde que se estabiliza la señal.
- Para solucionarlo: usar RT, que la FSM *controle* el timer.

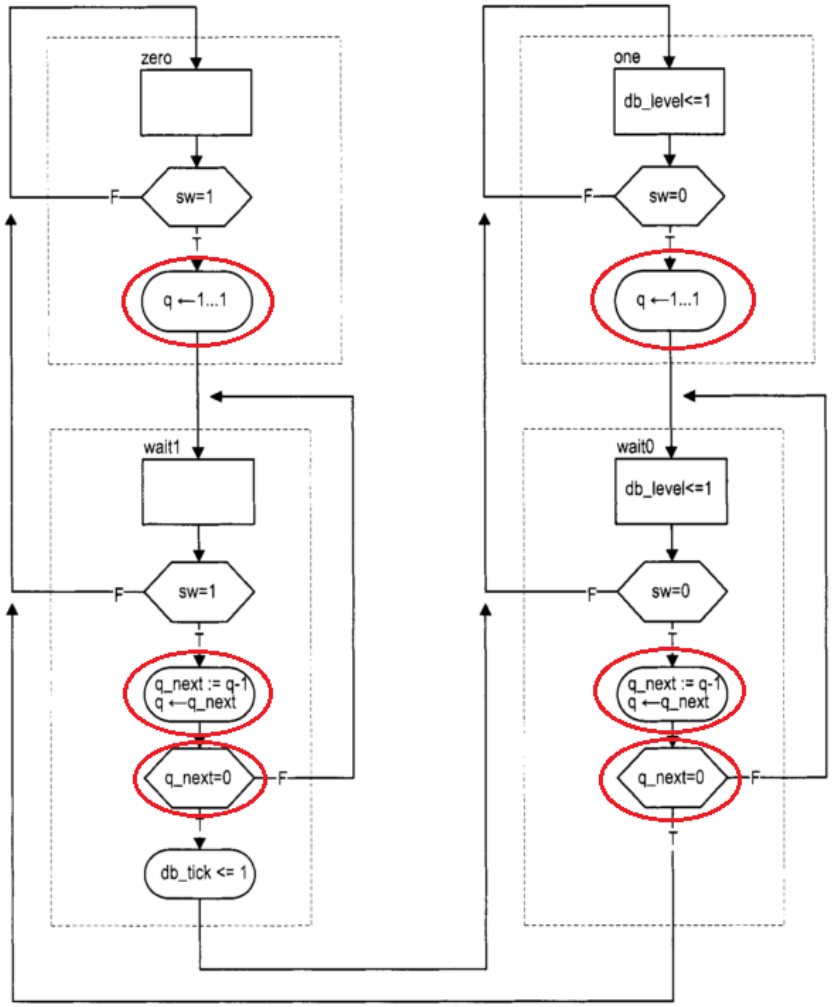
Diseño: ASMD


zero
sw=1
F
T
q ←1...1
wait1
sw=1
F
T
q_next := q-1
q ←q_next
q_next=0
F
T
db_tick <= 1
one
db_level<=1
sw=0
F
T
q ←1...1
wait0
db_level<=1
sw=0
F
T
q_next := q-1
q ←q_next
q_next=0
F
T

Diseño: Datapath en ASMD

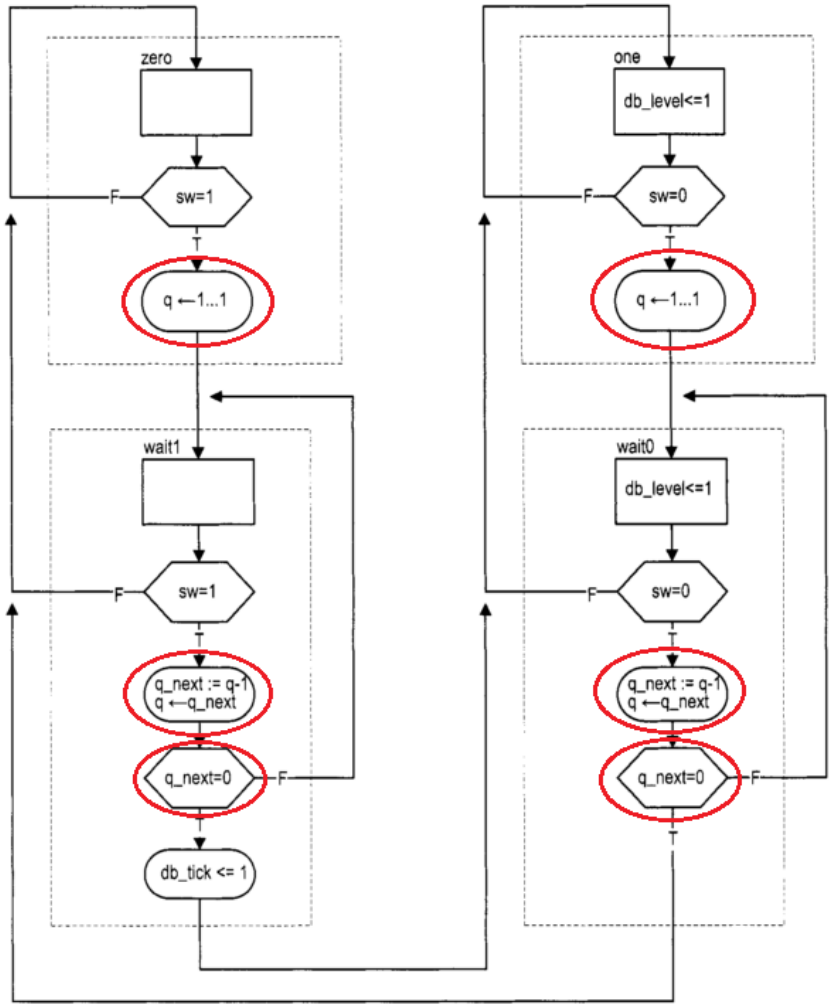


Diseño: Datapath en ASMD



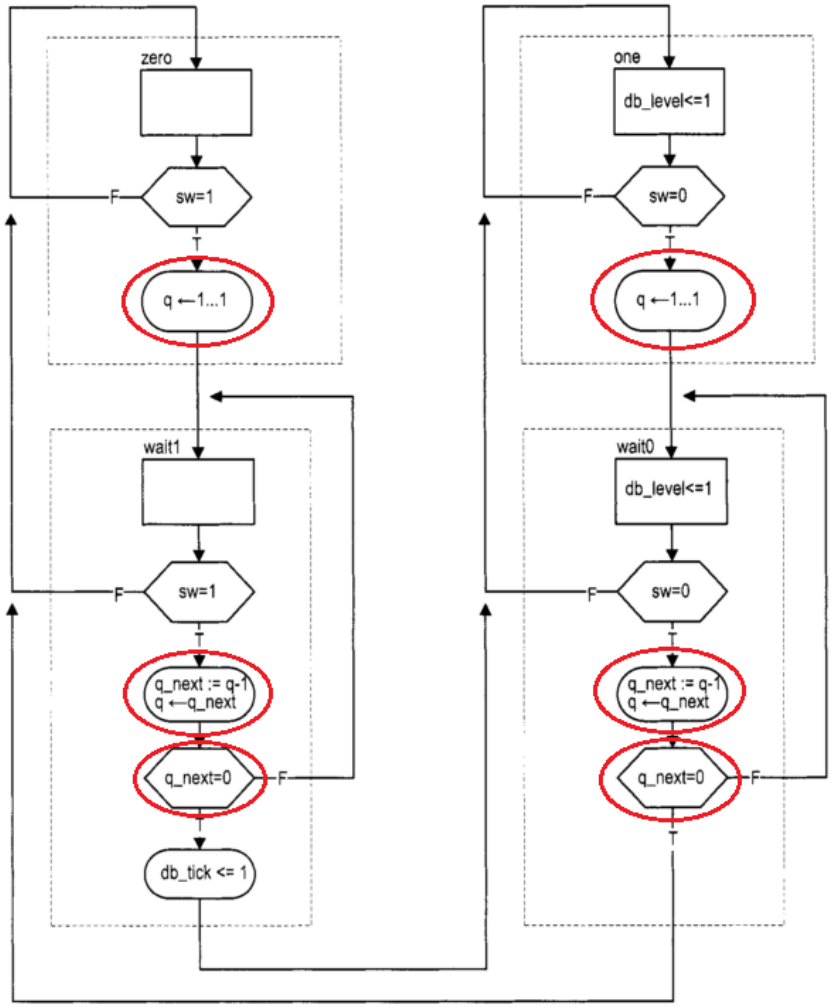
- Identificar los componentes principales del DataPath y las señales que estado asociadas (que generan y que necesitan).

Diseño: Datapath en ASMD



- Identificar los componentes principales del DataPath y las señales que estado asociadas (que generan y que necesitan).
- Contador y:
  1. Cargarlo
  2. Decrementarlo
  3. Levantar una señal de estado cuando llega a cero.

Diseño: Datapath en ASMD



- Identificar los componentes principales del DataPath y las señales que estado asociadas (que generan y que necesitan).
- Contador y:
  1. Cargarlo
  2. Decrementarlo
  3. Levantar una señal de estado cuando llega a cero.
- Entonces: contador con `q_load`, `q_dec` y que genera `q_zero`.

Código con Data Path explícito: Entidad

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity debounce is
    port(
        clk, reset: in std_logic;
        sw: in std_logic;
        db_level, db_tick: out std_logic
    );
end debounce ;

architecture exp_fsmd_arch of debounce is
    constant N: integer:=21;  -- filter of 2^N * 10ns =(aprox) 20ms
    -- Control-path:
    type state_type is (zero, wait0, one, wait1); -- estado
    signal state_reg, state_next: state_type; -- estado
    signal q_load, q_dec: std_logic;-- salidas para data path

    -- Data path
    signal q_reg, q_next: unsigned(N-1 downto 0);-- estado
    signal q_zero: std_logic; -- salidas para control path
begin
```

Código con Data Path explícito: Data path

```
-- DATA PATH : Circuito secuencial regular - CONTADOR
-- entradas: q_load, q_dec
-- salidas para control path: q_zero

-- 1) Registros del datapath
process(clk,reset)
begin
   if reset='1' then
      q_reg <= (others=>'0');
   elsif (clk'event and clk='1') then
      q_reg <= q_next;
   end if;
end process;

-- 2) next state logic
q_next <= (others=>'1') when q_load='1' else
          q_reg - 1 when q_dec='1' else
          q_reg;

-- 3) output logic
q_zero <= '1' when q_next=0 else '0';
```

Código con Data Path explícito: Control Path

```
-- CONTROL PATH : FSM con:
    -- entradas: sw, q_zero
    -- salidas p' data path : q_load y q_reg para
    -- salidas del circuito: db_level, db_tick

-- 1) Registro de estado
process(clk,reset)
begin
    if reset='1' then
        state_reg <= zero;
    elsif (clk'event and clk='1') then
        state_reg <= state_next;
    end if;
end process;
```

Código con Data Path explícito: Control Path - cont

```
-- 2) Next state logic
process(state_reg,sw,q_zero)
begin
   state_next <= state_reg;
   case state_reg is
      when zero =>
         if (sw='1') then
            state_next <= wait1;
         end if;
      when wait1=>
         if (sw='1') then
            if (q_zero='1') then
               state_next <= one;
            end if;
         else -- sw='0'
            state_next <= zero;
         end if;
      when one =>
         if (sw='0') then
            state_next <= wait0;
         end if;
      when wait0=>
         if (sw='0') then
            if (q_zero='1') then
               state_next <= zero;
            end if;
         else -- sw='1'
            state_next <= one;
         end if;
   end case;
end process;
```

```
-- 3) Output logic
process(state_reg,sw,q_zero)
begin
   q_load <= '0';
   q_dec <= '0';
   db_tick <= '0';
   case state_reg is
      when zero =>
         db_level <= '0';
         if (sw='1') then
            q_load <= '1';
         end if;
      when wait1=>
         db_level <= '0';
         if (sw='1') then
            q_dec <= '1';
            if (q_zero='1') then
               db_tick <= '1';
            end if;
         end if;
      when one =>
         db_level <= '1';
         if (sw='0') then
            q_load <= '1';
         end if;
      when wait0=>
         db_level <= '1';
         if (sw='0') then
            q_dec <= '1';
         end if;
   end case;
end process;
```

Código con Data Path implícito

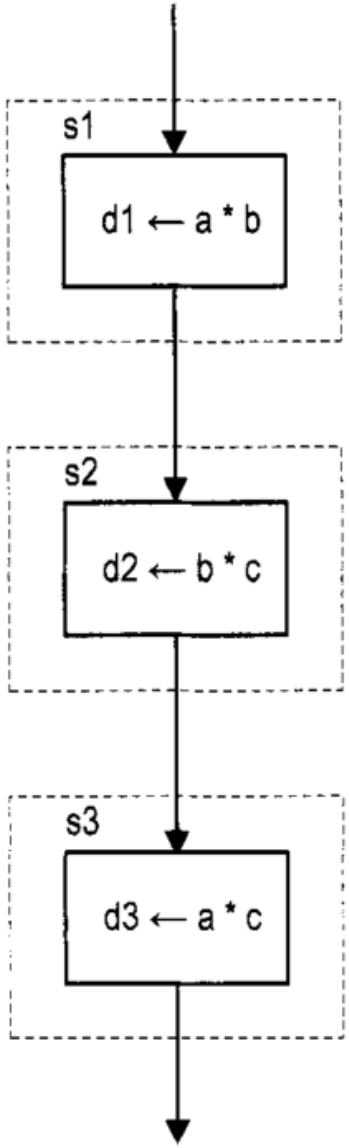
```
architecture fsmd_arch of debounce is
   constant N: integer:=21;  -- filter of 2^N * 10ns = 20ms
   type state_type is (zero, wait0, one, wait1);
   signal state_reg, state_next: state_type;
   signal q_reg, q_next: unsigned(N-1 downto 0);
begin
   -- FSMD registros de estado Y datos
   process(clk,reset)
   begin
      if reset='1' then
         state_reg <= zero;
         q_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
         q_reg <= q_next;
      end if;
   end process;
```

Código con Data Path implícito

```
-- next-state logic y
-- unidades funcionales data path + routing
process(state_reg,q_reg,sw,q_next)
begin
   state_next <= state_reg;
   q_next <= q_reg;
   db_tick <= '0';
   case state_reg is
      when zero =>
         db_level <= '0';
         if (sw='1') then
            state_next <= wait1;
            q_next <= (others=>'1');
         end if;
      when wait1=>
         db_level <= '0';
         if (sw='1') then
            q_next <= q_reg - 1;
            if (q_next=0) then
               state_next <= one;
               db_tick <= '1';
            end if;
         else -- sw='0'
            state_next <= zero;
         end if;
      when one =>
         db_level <= '1';
         if (sw='0') then
            state_next <= wait0;
            q_next <= (others=>'1');
         end if;
      when wait0=>
         db_level <= '1';
         if (sw='0') then
            q_next <= q_reg - 1;
            if (q_next=0) then
               state_next <= zero;
            end if;
         else -- sw='1'
            state_next <= one;
         end if;
   end case;
end process;
```

Comparación

- El código con datapath implícito sigue mas cercanamente el ASMD/FSMD.
- Sin embargo, confiamos en que el soft de sintesis genere el datapath automáticamente, y por lo tanto, tenemos menos control.



Ejercicio