Lógica Secuencial en VHDL (II)

UNIVERSIDAD NEBRIJA

Paralelismo

- **Dos tipos de paralelismo:**
  - **Paralelismo espacial**
    - Duplicar el hardware permite realizar varias tareas a la vez
  - **Paralelismo temporal**
    - La tarea se divide en varias etapas
    - También llamado pipelining
    - Por ejemplo, una línea de ensamblado.

Definiciones de Paralelismo

- **Token:** Grupo de entradas procesadas para producer un grupo de salidas
- **Latencia:** Tiempo que tarda un token en pasar del inicio al final
- **Throughput:** Número de tokens producidos por unidad de tiempo

El paralelismo incrementa el throughput

Ejemplo de paralelismo

- Ben Bitdiddle cocina galletas para celebrar que ha aprobado Sistemas Digitales
- 5 minutos para preparar la masa para 1 bandeja (Roll)
- 15 minutos para cocinarlas (Bake)
- ¿Cuál es la latencia y el throughput sin paralelismo?

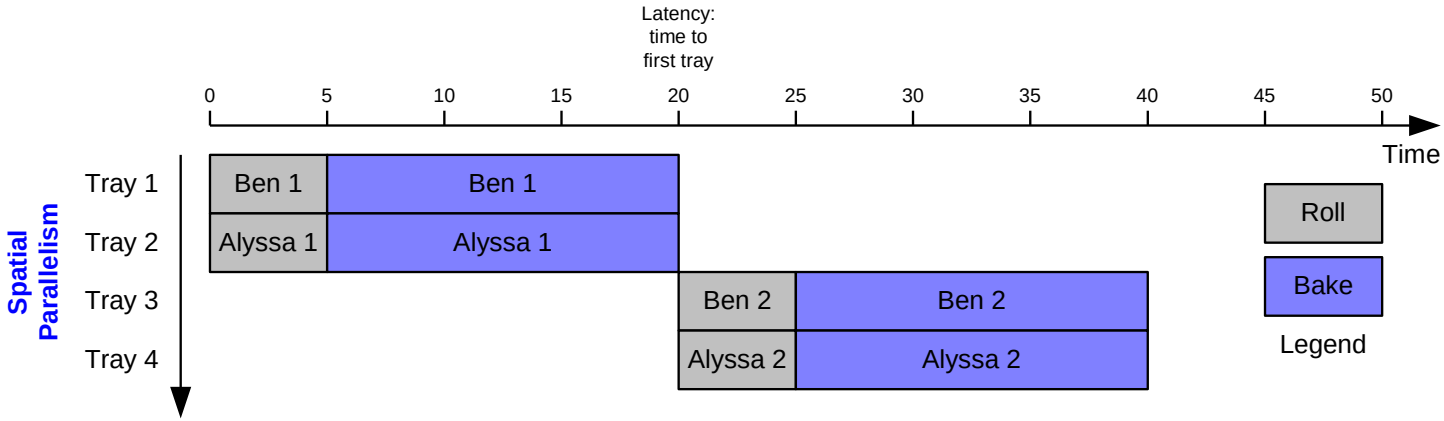
Latencia = 5 + 15 = 20 minutos = **1/3 hora**

Throughput = 1 bandeja / 1/3 hora= **3 bandejas/hora**

Ejemplo de paralelismo

- ¿Cuál es la latencia y el throughput si Ben usa paralelismo?
  - **Paralelismo espacial:** Ben pide a Allysa P. Hacker que le ayude, usando ella su propio horno.
  - **Paralelismo temporal:**
    - Dos etapas: amasado y cocinado (Roll / Bake)
    - Ben usa dos bandejas
    - Mientras la primera hornada se está cocinando, él amasa la segunda hornada y así sucesivamente.

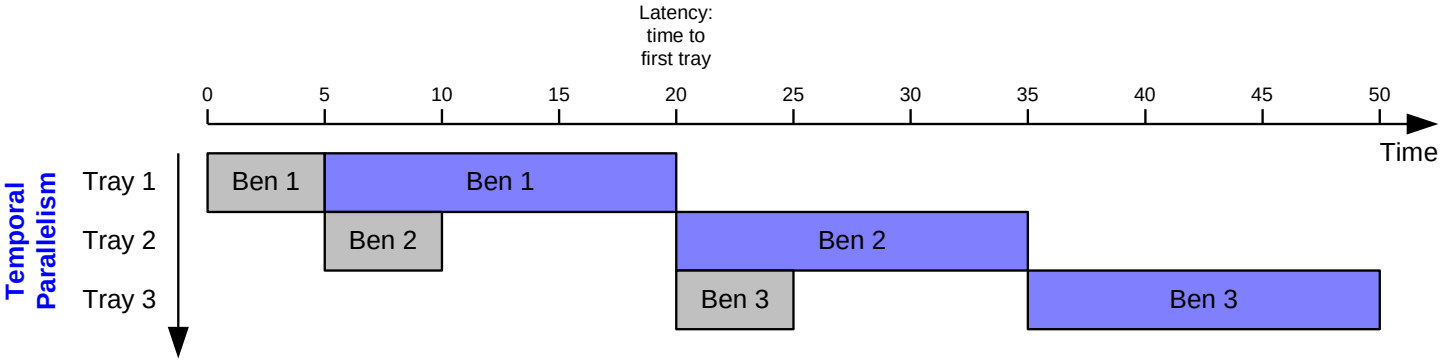
Paralelismo espacial



Latencia = 5 + 15 = 20 minutos = **1/3 hora**

Throughput = 2 bandejas/ 1/3 hora = **6 bandejas/hora**

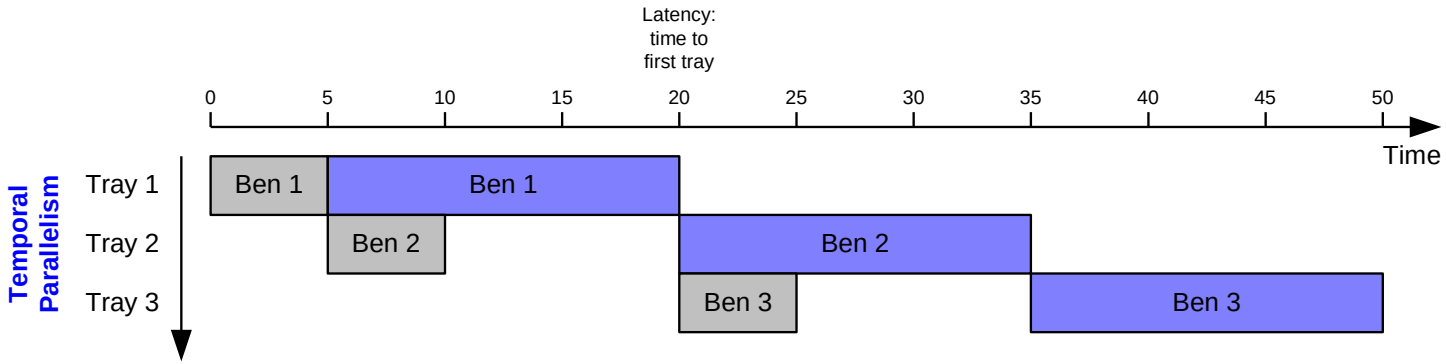
Temporal Parallelism



Latency = ?

Throughput = ?

Paralelismo temporal



Latencia = 5 + 15 = 20 minutos = **1/3 hora**

Throughput = 1 bandeja/ 1/4 hora= **4 bandejas/hora**

Usando ambas técnicas, el throughput sería **8 bandejas/hora**

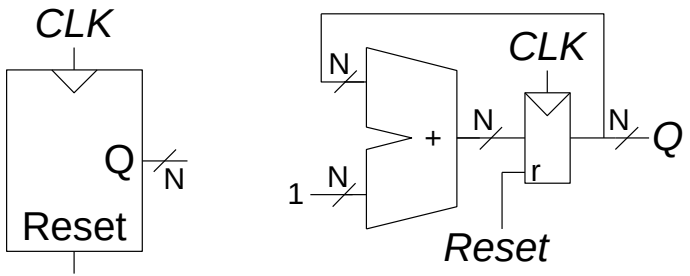
Diseño de lógica secuencial síncrona

- Romper las rutas cíclicas **insertando registros**
- Los registros contienen el **estado** del sistema
- El estado cambia en el flanco del reloj: Sistema **sincronizado** con el reloj
- **Las reglas** de composición de este tipo de circuitos son:
  - Cada elemento del circuito es o un registro o un circuito combinacional
  - Al menos un elemento del circuito es un registro
  - Todos los registros reciben la misma señal de reloj
  - Cada ruta cíclica contiene al menos un registro.
- Dos tipos de circuitos secuenciales síncronos usados habitualmente:
  - Finite State Machines (FSMs)
  - Pipelines

Contadores

- Incrementa en cada flanco de reloj
- Usados para recorrer números de forma cíclica. Por ejemplo,
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- Usos:
  - Display de un reloj digital
  - Contador de programa: mantiene la posición de la instrucción que se ejecuta

Symbol **Implementation**



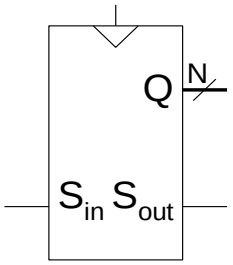
Implementación VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD_UNSIGNED.ALL;

entity counter is
  generic(N: integer := 8);
  port(clk, reset: in  STD_LOGIC;
       q:          out STD_LOGIC_VECTOR(N-1 downto 0));
end counter;

architecture synth of counter is
begin
  process(clk, reset) begin
    if reset='1' then
       q <= (OTHERS => '0');
    elsif rising_edge(clk) then
       q <= q + '1';
    end if;
  end process;
end synth;
```

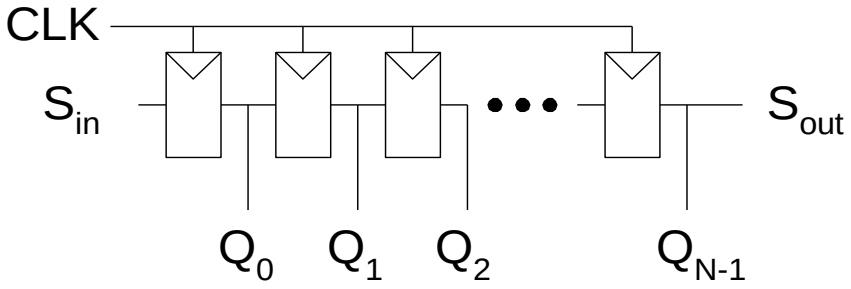
Shift Registers

- Desplazan un nuevo bit en cada flanco de reloj
- Entra y sale un bit en cada flanco.
- *Serial-to-parallel converter*: convierte entrada serie (S_{in}) en salida paralela ($Q_{0:N-1}$)

Symbol:



Implementation:

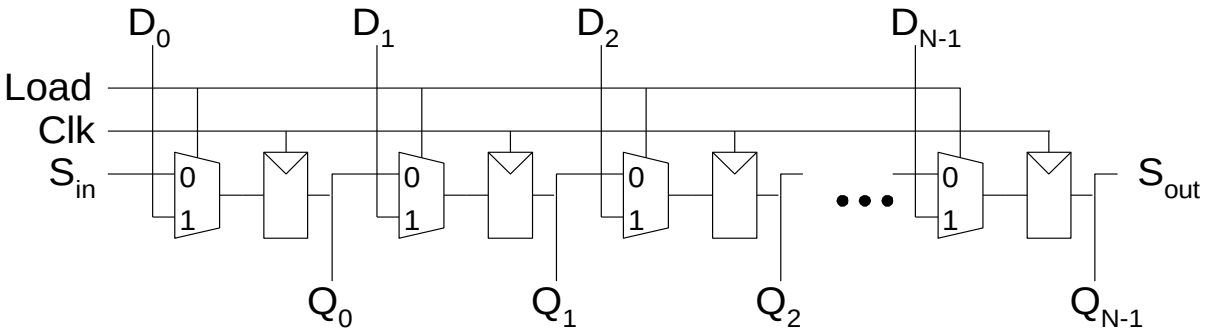


Implementación VHDL

```
entity shiftreg is
  generic(N: integer := 8);
  port(clk, reset: in  STD_LOGIC;
       load, sin:  in  STD_LOGIC;
       d:          in  STD_LOGIC_VECTOR(N-1 downto 0);
       q:          out STD_LOGIC_VECTOR(N-1 downto 0);
       sout:       out STD_LOGIC);
end shiftreg;

architecture synth of shiftreg is
begin
  process(clk, reset) begin
    if reset='1' then
       q <= (OTHERS => '0');
    elsif rising_edge(clk) then
      if load='1' then
         q <= d;
      else
         q <= q(N-2 downto 0) & sin;
      end if;
    end if;
  end process;

  sout <= q(N-1);
end synth;
```

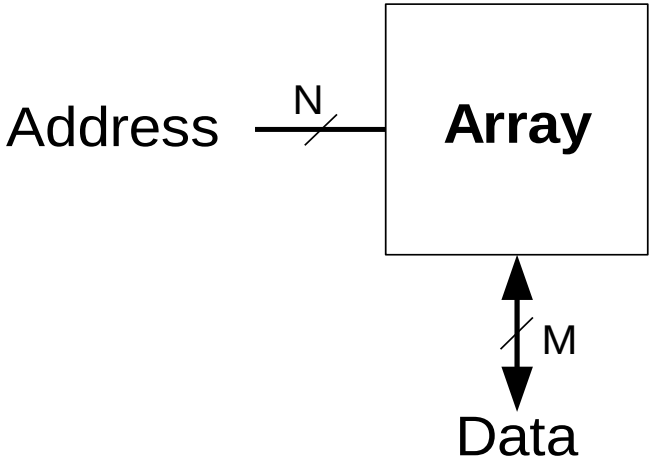
Shift Register con carga en paralelo

- Cuando *Load* = 1, actúa como un registro de *N*-bits normal
- Cuando *Load* = 0, actúa como un shift register
- Puede actuar como un *serial-to-parallel converter* (S_{in} to $Q_{0:N-1}$) o un *parallel-to-serial converter* ($D_{0:N-1}$ to S_{out})



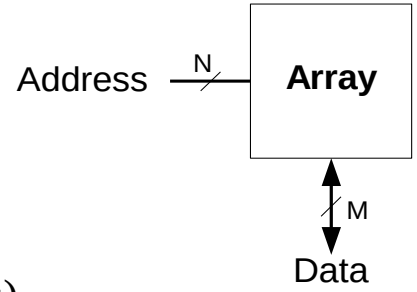
Arrays de memoria

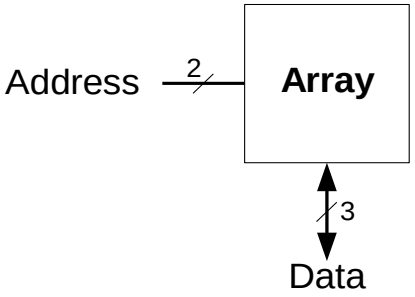
- Almacenan grandes cantidades de datos de forma eficiente
- 3 tipos comunes:
  - Dynamic random access memory (DRAM)
  - Static random access memory (SRAM)
  - Read only memory (ROM)
- Un valor de datos de *M*-bits se lee/escribe en cada dirección única de *N*-bits

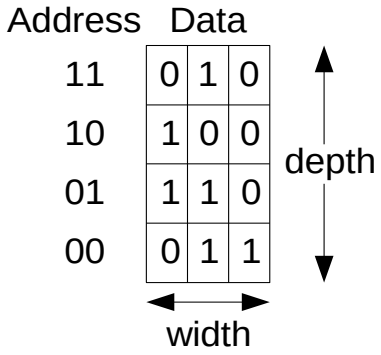


Arrays de memoria

- Array de dos dimensiones de celdas de bits
- Cada celda almacena 1 bit
- N bits de direcciones y M bits de datos:
  - 2^N filas y M columnas
  - **Depth/profundidad:** número de filas (número de palabras)
  - **Width/anchura:** número de columnas (tamaño de palabra)
  - **Tamaño del array:** depth × width = $2^N \times M$

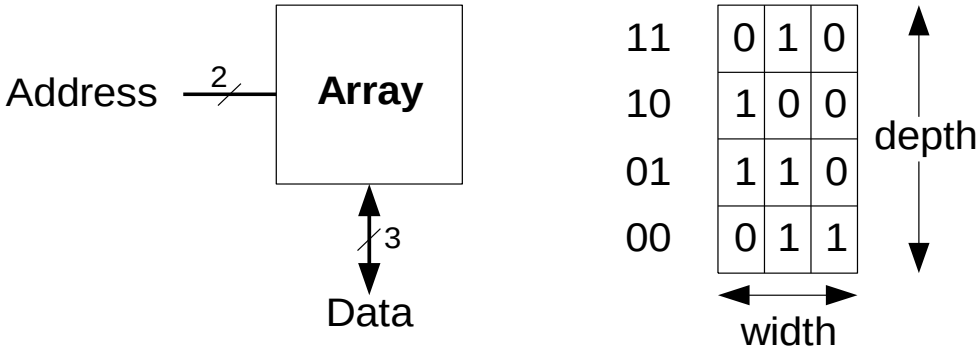




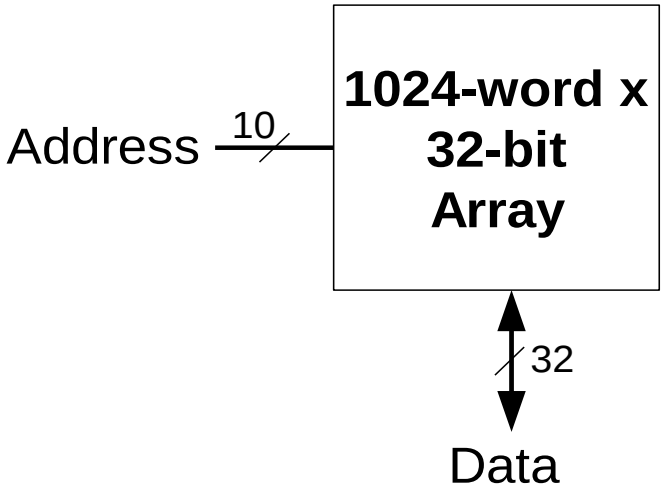


Ejemplo de arrays de memoria

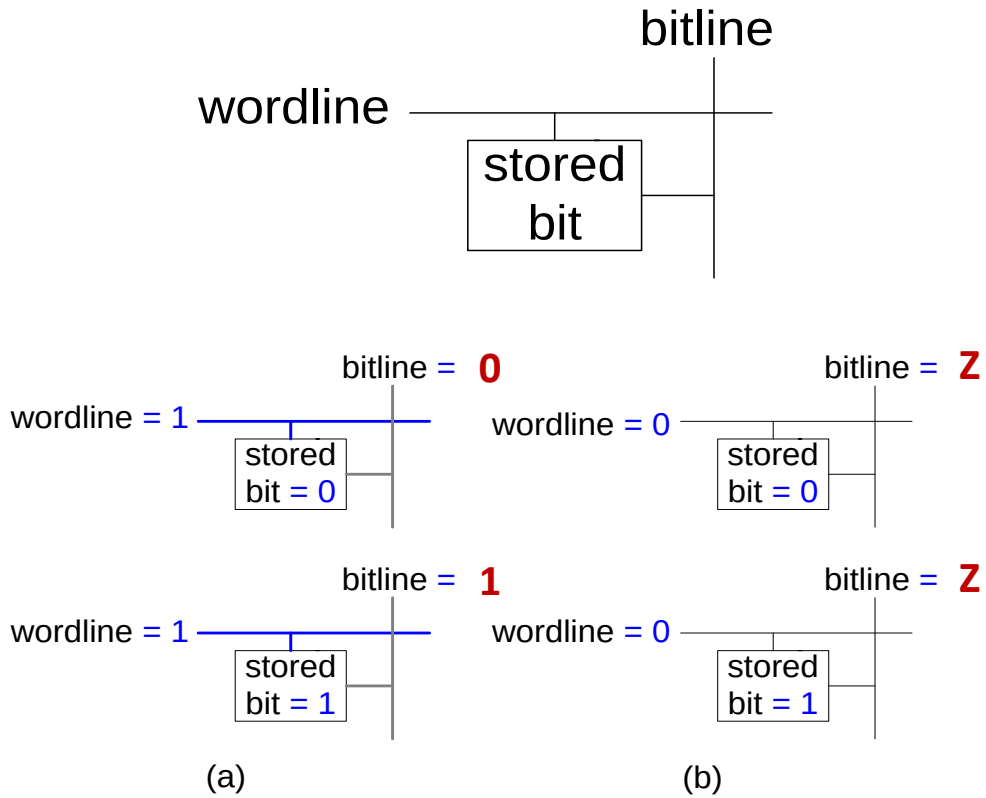
- $2^2 \times$ 3-bits
- Número de palabras: 4
- Tamaño de palabra: 3-bits
- Por ejemplo, la palabra de 3-bits almacenada en la dirección 10 es 100



Arrays de memoria

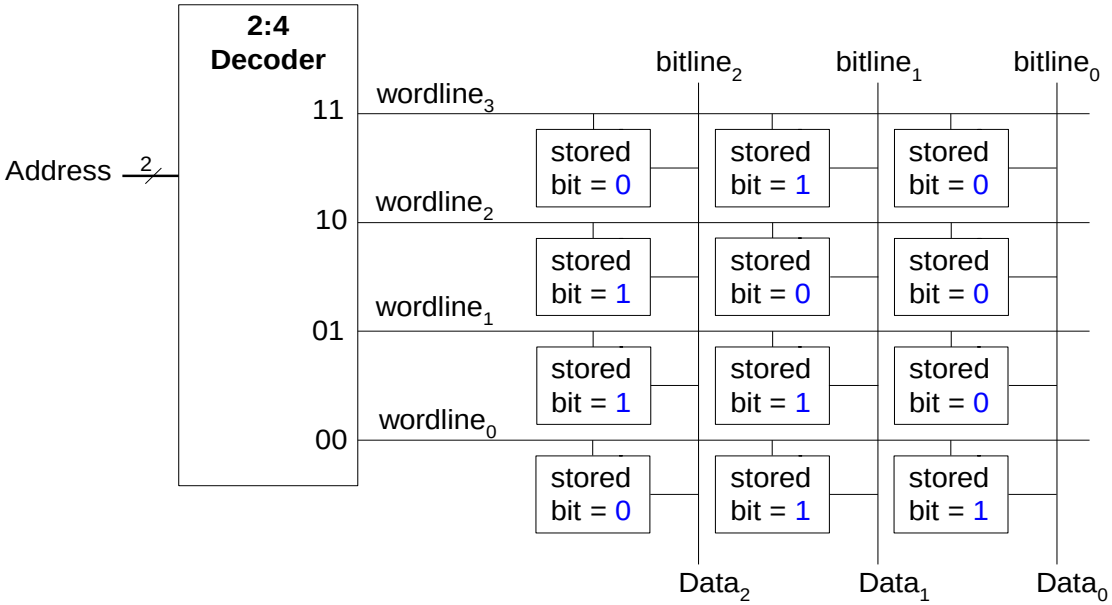


Celdas de bits de arrays de memoria



Arrays de memoria

- Wordline:
  - Como un enable
  - Se lee/escribe una sola línea de la memoria
  - corresponde a una dirección única
  - Solo un wordline en ALTO a la vez



Tipos de Memoria

- Random access memory (RAM): **volátil**
- Read only memory (ROM): **no volátil**

RAM: Random Access Memory

- **Volátil:** pierde sus datos cuando se interrumpe la alimentación
- Escritura y lectura rápida
- La memoria principal de tu ordenador

Se ha llamado históricamente así pues se puede acceder con la misma facilidad a cualquier palabra (frente a las memorias de acceso secuencial como las cintas)

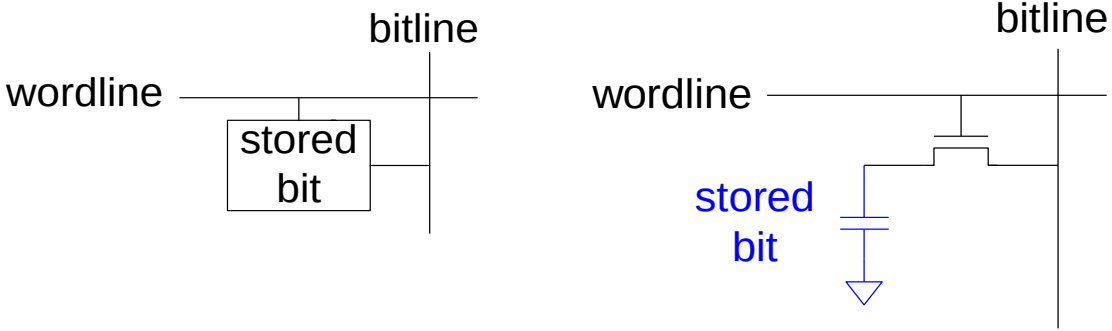
Tipos de RAM

- DRAM (Dynamic random access memory)
- SRAM (Static random access memory)

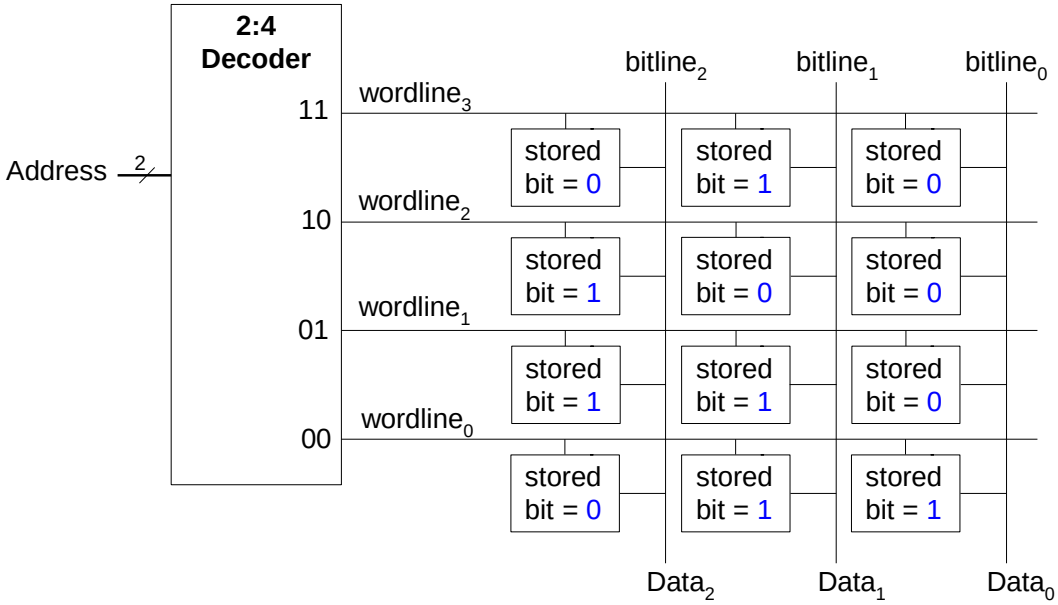
- Difieren en cómo almacenan los datos:
  - DRAM usa una capacitancia
  - SRAM usa inversores acoplados

DRAM

- *Dynamic* porque los valores necesitan ser refrescados periódicamente y también despues de leerlos para evitar que se borren:
  - Fuga de carga en el capacitor hace que se degrade el valor
  - La lectura destruye el valor guardado.



Recordatorio arrays de memoria



Implementación VHDL

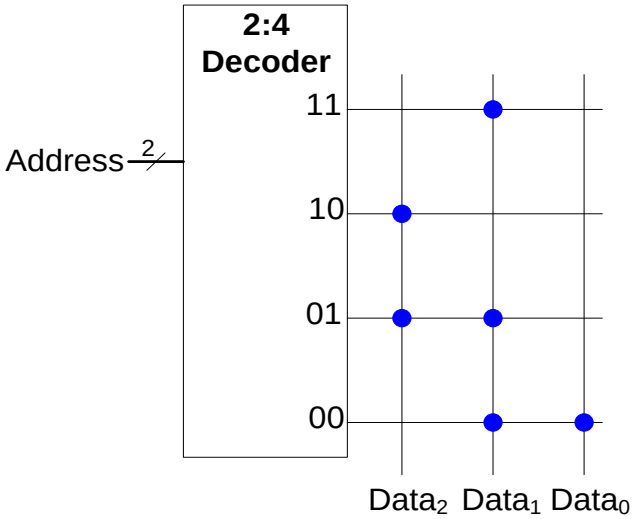
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD_UNSIGNED.ALL;

entity ram_array is
  generic(N: integer := 6; M: integer := 32);
  port(clk,
       we:   in  STD_LOGIC;
       adr:  in  STD_LOGIC_VECTOR(N-1 downto 0);
       din:  in  STD_LOGIC_VECTOR(M-1 downto 0);
       dout: out STD_LOGIC_VECTOR(M-1 downto 0));
end ram_array;

architecture synth of ram_array is
  type mem_array is array((2**N-1) downto 0)
       of STD_LOGIC_VECTOR(M-1 downto 0);
  signal mem: mem_array;
begin
  process(clk) begin
    if rising_edge(clk) then
      if we='1' then mem(TO_INTEGER(adr)) <= din;
      end if;
    end if;
  end process;

  dout <= mem(TO_INTEGER(adr));
end synth;
```

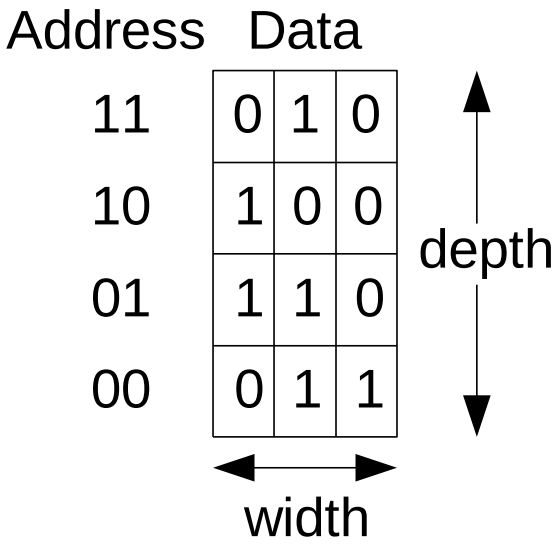
Se definen tipos propios con type.
Array es un tipo que existe para construir vectores y matrices.

ROM: Read Only Memory

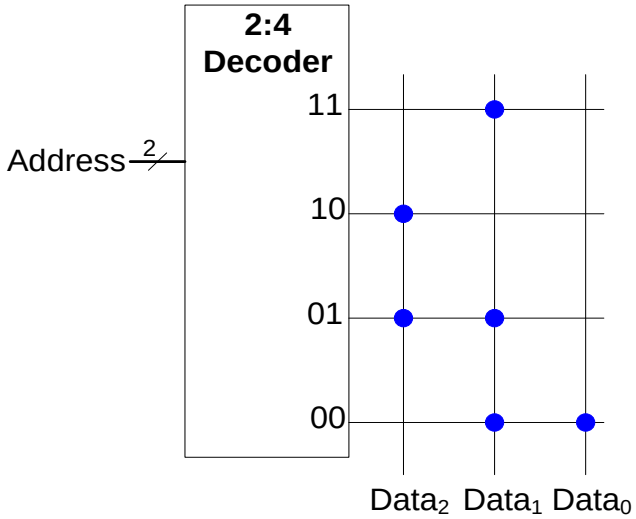
- **No volátil:** retiene la información cuando se apaga la alimentación
- Lee rápidamente, pero la escritura es imposible o más lenta, o limitada en número de veces.
- La memoria flash usada en cámaras, móviles y otros dispositivos es un tipo de ROM

ROM: Dot Notation





ROM Logic



$$Data_2 = A_1 \oplus A_0$$

$$Data_1 = \overline{A_1} + A_0$$

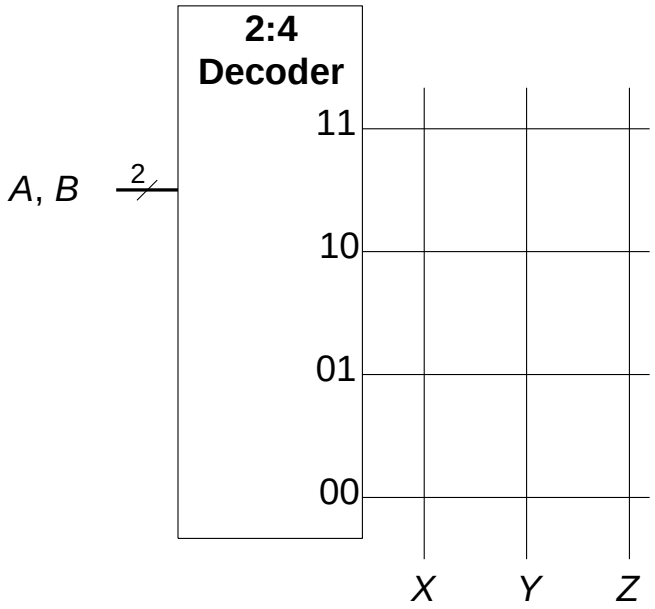
$$Data_0 = \overline{A_1}\,\overline{A_0}$$

Implementación VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity rom is
  port(adr:  in  STD_LOGIC_VECTOR(1 downto 0);
       dout: out STD_LOGIC_VECTOR(2 downto 0));
end rom;

architecture synth of rom is
begin
  process(all) begin
    case adr is
      when "00" => dout <= "011";
      when "01" => dout <= "110";
      when "10" => dout <= "100";
      when "11" => dout <= "010";
       when others => dout <= "UUU";
    end case;
  end process;
end synth;
```

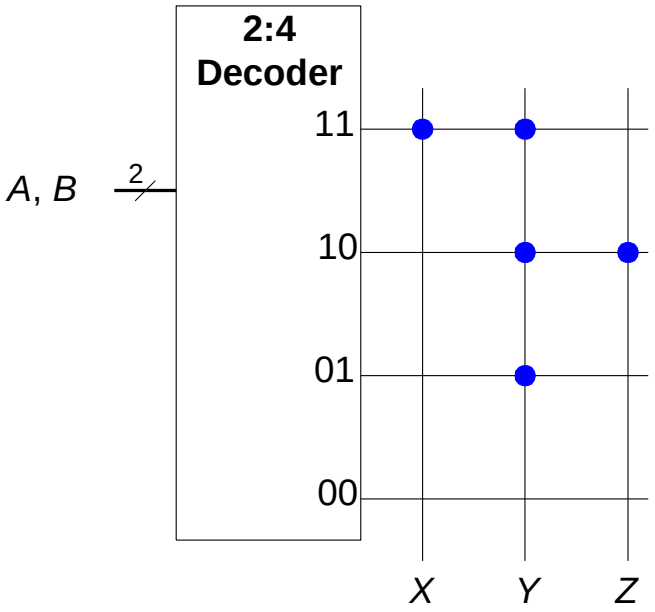
Ejemplo: Lógica con ROMs

- Implementa las siguientes funciones lógicas usando una ROM de $2^2 \times 3$-bits:
  - $X = AB$
  - $Y = A + B$
  - $Z = A\overline{B}$

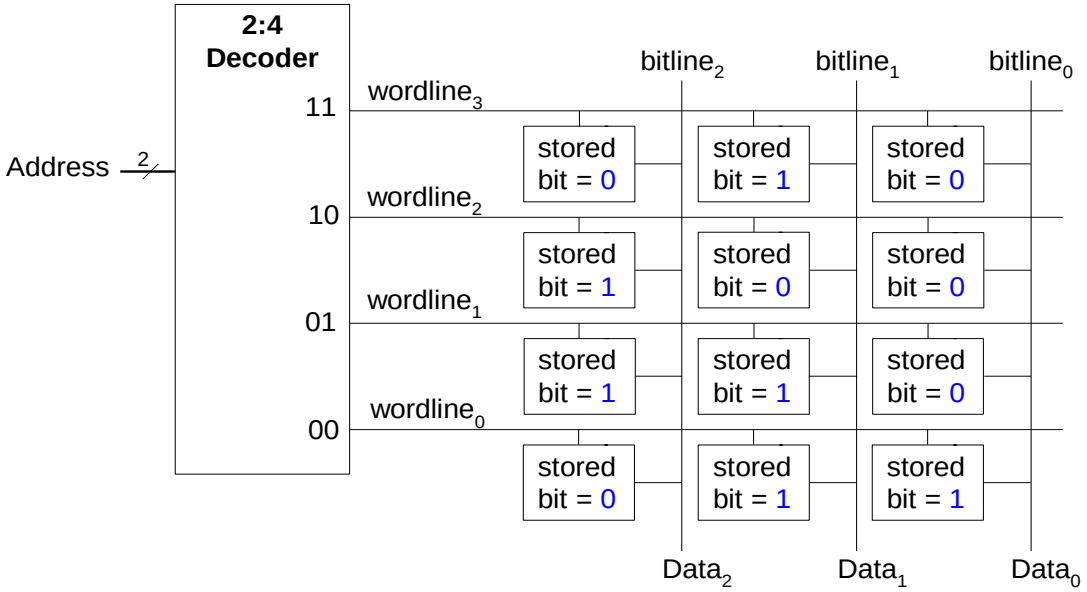


Ejemplo: Lógica con ROMs

- Implementa las siguientes funciones lógicas usando una ROM de $2^2 \times 3$-bits:
  - $X = AB$
  - $Y = A + B$
  - $Z = A\,\overline{B}$



Lógica con cualquier array de memoria



$$Data_2 = A_1 \oplus A_0$$

$$Data_1 = \overline{A_1} + A_0$$

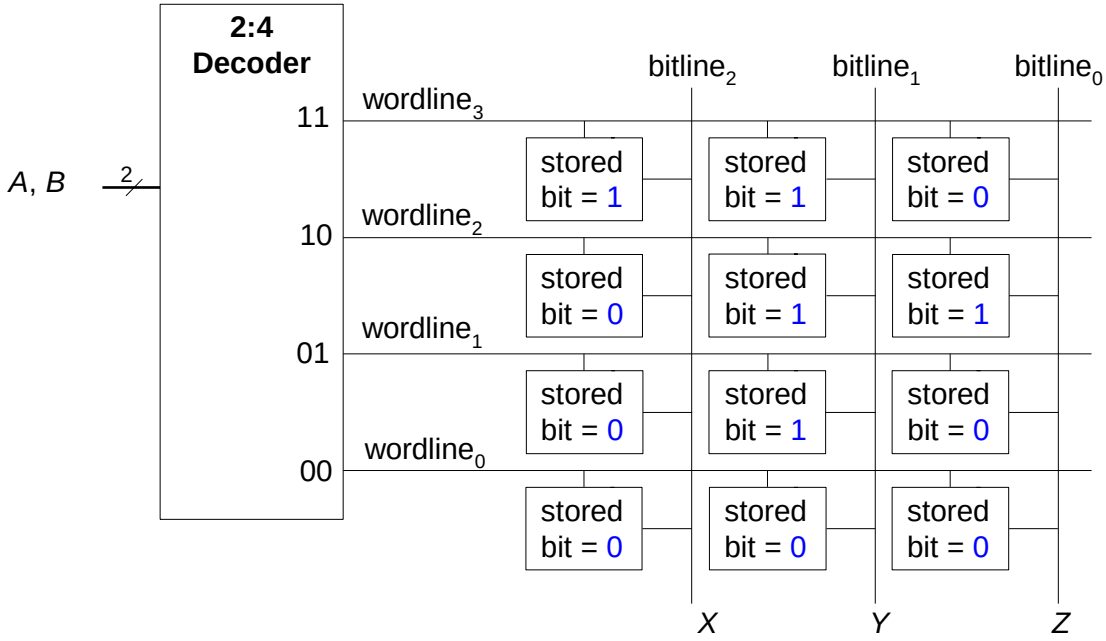
$$Data_0 = \overline{A_1}\,\overline{A_0}$$

Lógica con cualquier array de memoria

- Implementa las siguientes funciones usando un array de memoria de $2^2 \times 3$-bits:
  - $X = AB$
  - $Y = A + B$
  - $Z = A\overline{B}$

Lógica con cualquier array de memoria

- Implementa las siguientes funciones usando un array de memoria de $2^2 \times 3$-bits:
  - X = AB
  - Y = A + B
  - Z = A $\overline{B}$

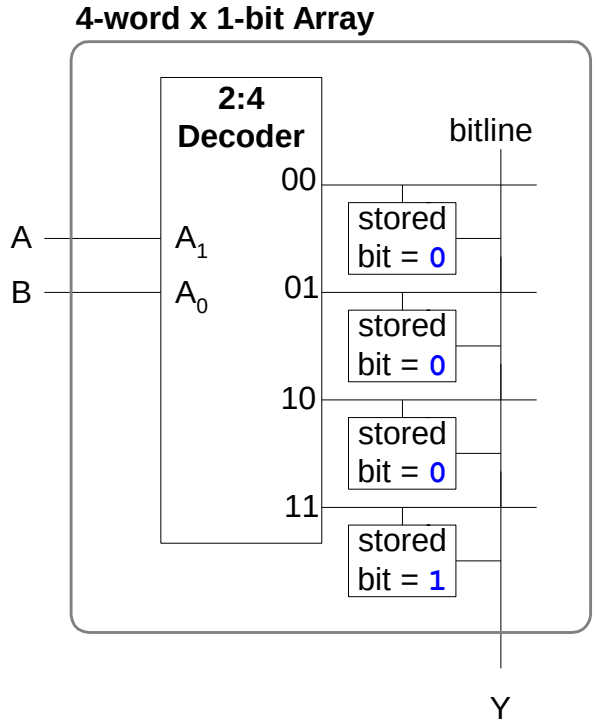


Lógica con cualquier array de memoria

- Llamados *lookup tables* (LUTs): encuentra la salida como combinaciones de las entradas (direcciones)

Truth Table

A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



Implementación VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity lut is
  port(adr:  in  STD_LOGIC_VECTOR(1 downto 0);
       dout: out STD_LOGIC);
end lut;

architecture synth of lut is
begin
  process(all) begin
    case adr is
      when "00" => dout <= "0";
      when "01" => dout <= "0";
      when "10" => dout <= "0";
      when "11" => dout <= "1";
     when others => dout <= "X";
    end case;
  end process;
end synth;
```

Memorias Multipuerto

- **Puerto:** pareja dirección/datos
- Memoria de 3 puertos
  - 2 puertos de lectura (A1/RD1, A2/RD2)
  - 1 puerto de escritura (A3/WD3, WE3 habilita la escritura)
- **Register file:** pequeña memoria multipuerto