Ej. 1 — (2.0 puntos)La conjetura de Collatz establece que aplicando el algoritmo siguiente de forma iterativa tenemos una sucesión de números que siempre pasa por el número 1. El algoritmo parte de un número natural positivo, n, y aplica sucesivamente las siguientes reglas:

1.Si el número, n, es par, entonces $n = \frac{n}{2}$.

2.Si el número, n, es impar, entonces $n = 3n + 1$.

Diseñar el diagrama ASM, la ruta de datos y la tabla de salidas de la unidad de control de un sistema algorítmico que realice la comprobación empírica de la conjetura de Collatz. El sistema lee el número inicial, comprendido entre 2 y 255, de la dirección 0x00 de una memoria SRAM síncrona de 128x8 bits y guarda los valores intermedios de la sucesión en las posiciones de memorias consecutivas de dicha memoria a partir de la posicion 0x01. El sistema devolverá el número de iteraciones necesarias para llegar al valor 1. El seudo-código se incluye a continuación y a la derecha un ejemplo del contenido de la memoria tras ejecutar el algoritmo cuando el valor inicial es 12.

```
iter ← 0 ;
n ← mem(iter) ;
while n ≠ 1 do
    if n(0) =' 0' then
        n ← n/2 ;
    else
        n ← 3 · n + 1 ;
    iter++ ;
    mem(iter) ← n ;
end
```

Addr	Val
0x00	12
0x01	6
0x02	3
0x03	10
0x04	5
0x05	16
0x06	8
0x07	4
0x08	2
0x09	1
0x0A	...

El sistema tiene tres entradas y dos salidas. Las entradas son: `clk`, `rst` e `ini`. Supóngase que el valor inicial está codificado en binario puro y que está comprendido entre 2 y 255. Las salidas son `iter` y `fin`, de forma que la salida `iter` es el número de iteraciones necesarias para llegar a 1. El sistema comienza a funcionar cuando la señal `ini` se pone a 1. Cuando se complete el cálculo el sistema volverá al estado inicial, en el que la señal `fin` es igual a 1.

```
entity asm is
  port (clk   : in  std_logic;
        rst   : in  std_logic;
        ini   : in  std_logic;
        iter  : out std_logic_vector(6 downto 0);
        fin   : out std_logic
        );
end asm;
```

```
entity ram is
  port (clk   : in  std_logic;
        dina  : in  std_logic_vector(7 downto 0);
        addra : in  std_logic_vector(6 downto 0)
        wea   : in  std_logic;
        ena   : in  std_logic;
        douta : out std_logic_vector(7 downto 0)
        );
end ram;
```

En la ruta de datos se puede usar una memoria SRAM síncrona 128x8 bits, de un solo puerto y modo de escritura READ_FIRST, registros, registros con desplazamiento, contadores, una **única** ALU[(1)], y los elementos combinacionales adicionales que se consideren necesarios.

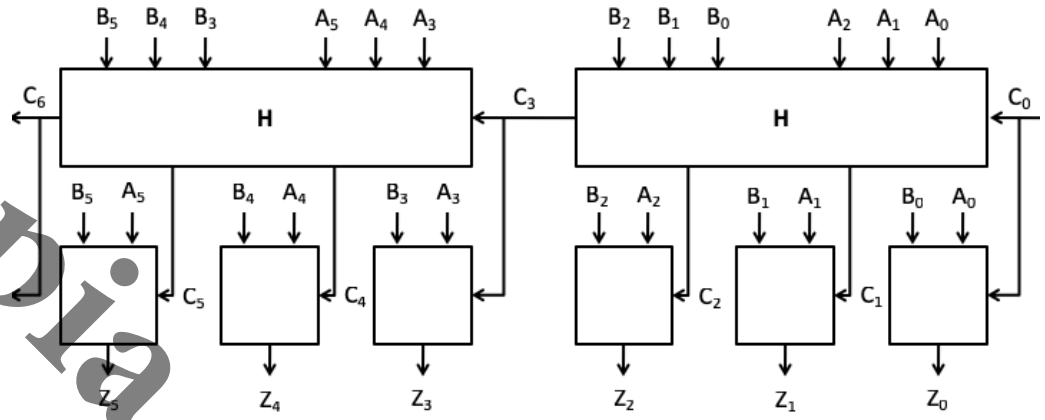
[(1)]: los códigos de operación de la ALU son: 00 (suma), 01 (resta), 10 (multiplicación) y 11 (and).

Ej. 2 — (1.50 puntos) Diseñar utilizando puertas lógicas:

1.Una red iterativa 1D que dados dos vectores de entrada, A y B, de n bits genere una salida, Z, de n bits tal que:

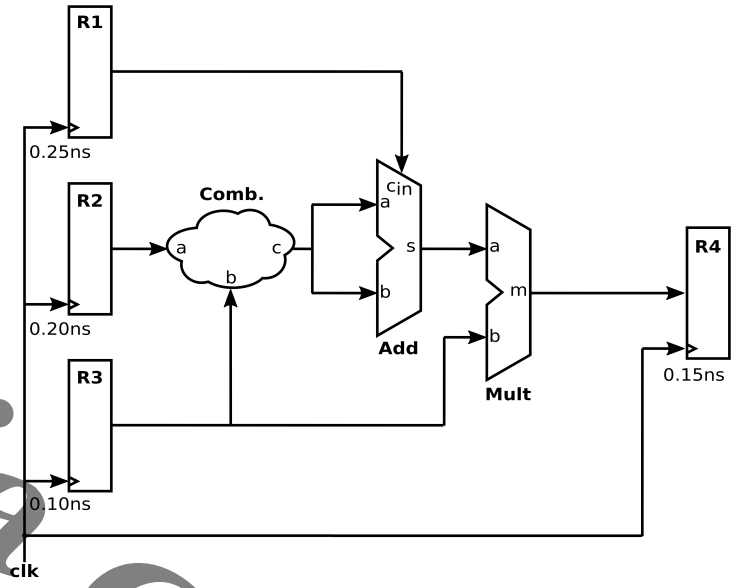
$$Z_i = \begin{cases} A_i & \text{si paridad de unos en } (A_{i-1},\ldots,A_0) \text{ es par y paridad de unos en } (B_{i-1},\ldots,B_0) \text{ es par} \\ A_i \text{ AND } B_i & \text{si paridad de unos en } (A_{i-1},\ldots,A_0) \text{ es par y paridad de unos en } (B_{i-1},\ldots,B_0) \text{ es impar} \\ A_i \text{ OR } B_i & \text{si paridad de unos en } (A_{i-1},\ldots,A_0) \text{ es impar y paridad de unos en } (B_{i-1},\ldots,B_0) \text{ es par} \\ B_i & \text{si paridad de unos en } (A_{i-1},\ldots,A_0) \text{ es impar y paridad de unos en } (B_{i-1},\ldots,B_0) \text{ es impar} \end{cases}$$

2.Una red iterativa con anticipación de operandos, tal y como se observa en la siguiente figura.



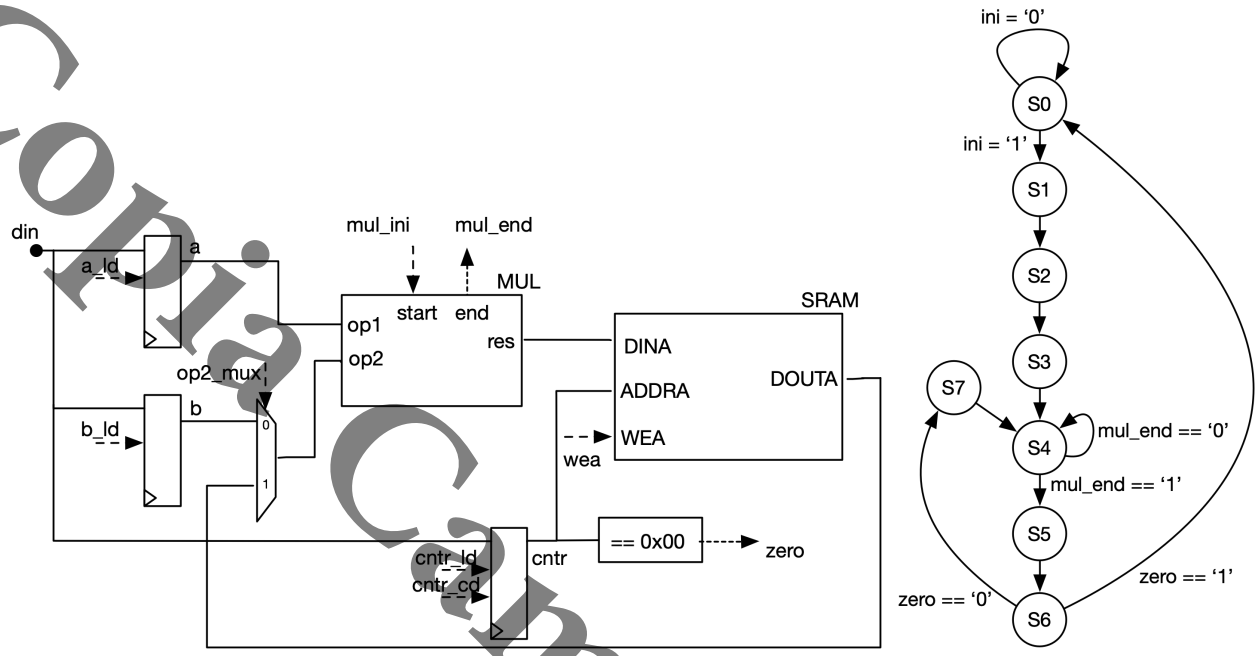
3.Suponiendo que todas las puertas de dos y tres entradas tienen un retardo t y las puertas de cuatro entradas tienen un retardo $2 \cdot t$, calcular el retardo del camino crítico en el diseño del apartado anterior para $n = 15$. En el caso de haber usado inversores, suponer que su retardo es despreciable.

Ej. 3 — (1.25 puntos) Dado el circuito de la figura, los valores de propagación de sus componentes son los siguientes: Comb(a→c) = Comb(b→c) = 1,5 ns, ADD(a→s) = ADD(b→s) = 1,00 ns, ADD(c_{in} →s) = 0,50 ns, y MUL(a→m) = MUL(b→m) = 2,50 ns. Los valores en las líneas de reloj son el retardo de propagación desde la fuente de reloj hasta la entrada de reloj. Los parámetros de los registros son: $t_{clk-2-q}$ = 0,15 ns, t_{setup} = 0,20 ns, t_{hold} = 0,10 ns.



1. Encontrar el camino crítico y justificar si el circuito puede funcionar correctamente a 200 MHz.
2. Indicar la máxima frecuencia a la que puede funcionar correctamente.
3. ¿Sería posible, mediante segmentación, que el circuito funcionase a 200 MHz? Si es el caso, indíquese dónde habría que ubicar los registros de segmentación para hacerlo posible; y compruebese si efectivametne, el circuito segmentado puede funcionar a 200 MHz. Supóngase que el retardo de reloj de el/los nuevo(s) registro(s) es de 0,10 ns.

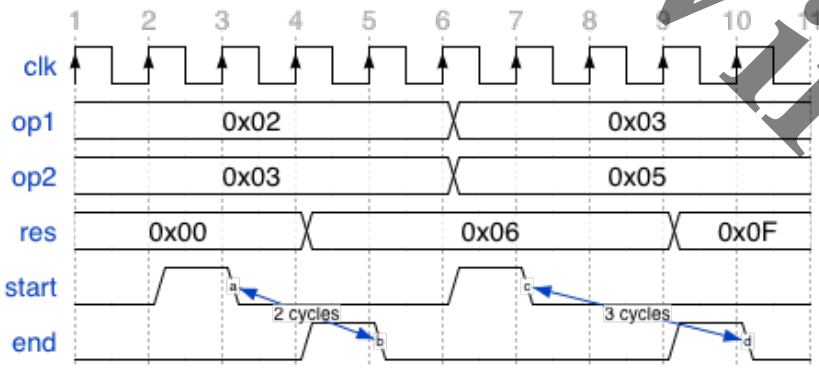
Ej. 4 — (1.25 puntos) Completar el cronograma correspondiente al siguiente sistema e indicar el contenido final de la memoria SRAM. Las entradas al sistema son `din`, `ini`, `clk` y `rst` (reset del registro, contador y MUL, a '0' en el cronograma) y `fin` es la salida; el resto de señales son internas según tabla adjunta. Considerar que la memoria es síncrona y funciona en modo READ_FIRST y ALWAYS_ENABLE. Suponer también que el módulo `cntr` es un contador descendente de 8 bits con carga paralela y que la señal `cntr_cd` es la señal de habilitación de cuenta abajo. El componente `MUL` es un ASM que realiza la multiplicación de sus dos operandos de entrada cuando su señal `start` se pone a 1. Cuando finaliza el cálculo activa la señal `end` durante un ciclo. La latencia del multiplicador es de 2 ciclos para multiplicaciones con resultado par y de 3 ciclos para multiplicaciones con el resultado impar y sus entradas y salidas están registradas. Véase abajo ejemplos de temporizacion del MUL para ambos casos.

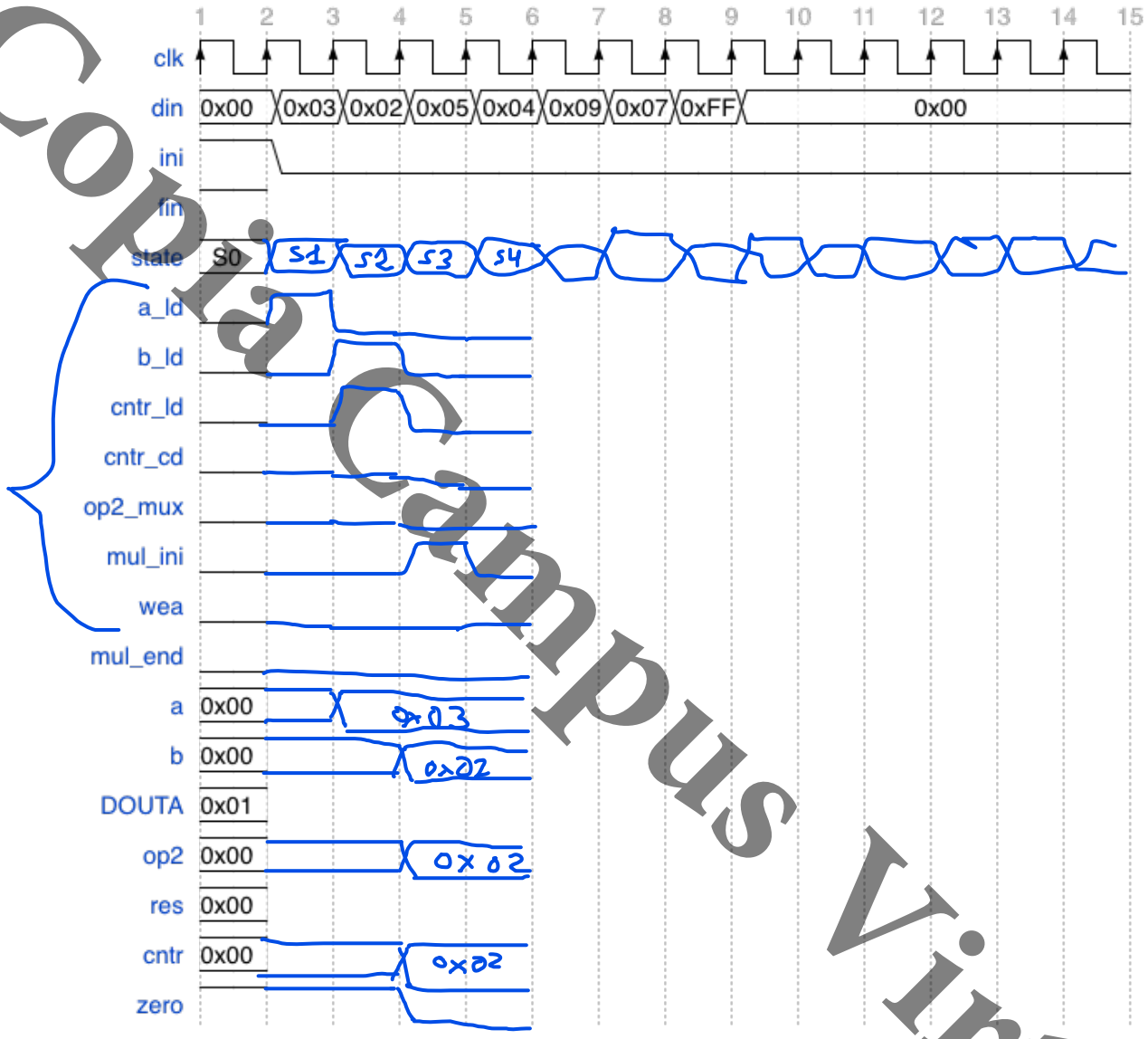


SRAM

ADDR	DATA
0x00	0x0001
0x01	0x0006
0x02	0x0004
0x03	0x0104
0x04	0xA000
0x05	0x1111

State	a_ld	b_ld	cntr_ld	cntr_cd	op2_mux	mul_ini	wea	fin
S0	0	0	0	0	0	0	0	1
S1	1	0	0	0	0	0	0	0
S2	0	1	1	0	0	0	0	0
S3	0	0	0	0	0	1	0	0
S4	0	0	0	0	0	0	0	0
S5	0	0	0	1	0	0	1	0
S6	0	0	0	0	0	0	0	0
S7	0	0	0	0	1	1	0	0





Ej. 5 — (0.5 puntos) Empleando una memoria de tamaño 16x1 bits implemente una memoria de 32x2 bits que se pueda leer y escribir. Indique claramente las conexiones de las señales de control así como la anchura y bits usados para formar los buses.