Diseño y verificación de un multiplicador secuencial con signo mediante el algoritmo de Booth modificado

Tarea 2

Integración de Sistemas Digitales

2019/2020

Guión de la práctica y bibliografía

- Consultad Tarea2 en PoliformaT
- Para la parte hardware consultad:
  - Información adjuntada en la tarea
- Para el diseño del banco de pruebas debe tenerse en cuenta
  - Bloque de teoría
  - Conceptos de Verificación
    - Lesson 4.2: Verificación de Radicador
    - Tarea 1: Verificación FIFO mediante aserciones, generación de estímulos y coberturas

Estructura de la práctica

1. **Introducción de la práctica.**
   - Objetivos
   - Estructura de la práctica
   - Funcionamiento de multiplicadores secuenciales
2. **Etapa RTL. Diseño de Componentes I. Data-path**
   - Diseño de registro de desplazamiento: para acumulador y multiplicador (**HDL**)
     genérico en su tamaño!
   - Diseño de registro para multiplicando (**HDL**)
     genérico en su tamaño!
   - Diseño del sumador restador (**HDL**)
     genérico en tamaño de operandos y resultado
3. **Etapa RTL. Diseño de componentes II.Control-path**
   - Diseño de un contador. FSM secundaria (**HDL**)
     genérico en su módulo!
   - Diseño de la FSM principal de control (**HDL**)

Estructura de la práctica

4. **Etapa RTL. Descripción del sistema y verificación funcional**
    1. Diseño Verilog del top del sistema multiplicador: control-path +data-path (**SystemVerilog**)
    2. Análisis (compilación) del diseño realizado
    3. Verificación funcional del diseño realizado

5. **Etapa Lógica. Compilación del sistema y simulación lógica**
    1. Síntesis (compilación) del sistema multiplicador.
        1. Obtención Recursos utilizados: logic-cells y flip-flops
    2. Análisis temporal estático del diseño realizado:
        1. Obtención fmax

6. **Verificación lógica BÁSICA del diseño realizado**

7. **Verificación compleja con RCSG, covergroups, program, interface, class, aserciones**

8. **Implementación hardware**

Objetivos

- Profundizar en el diseño jerárquico con HDL.
- Profundizar en el diseño de máquinas de estados mediante SystemVerilog.
- Realización de modelos configurables e independientes de la tecnología.
  - Uso de parameters de SystemVerilog.
  - No uso de instanciación de “cores” de Altera
- Consolidar conceptos de teoría sobre estructura y particionado de diseños síncronos.
- Verificación con systemverilog: RCSG, covergroups, program, interface, class, aserciones…
- Configuración de dispositivos programables del tipo SRAM.

CONTIENE ANIMACIONES

El Algoritmo the Booth

A SIGNED BINARY MULTIPLICATION TECHNIQUE

By ANDREW D. BOOTH

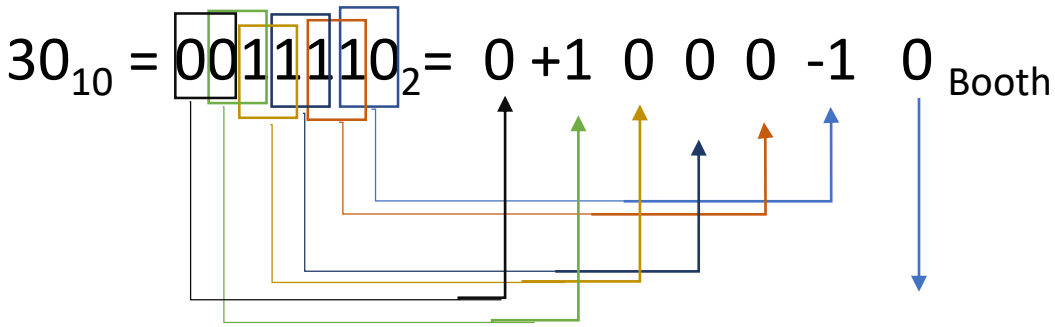
(Birkbeck College Electronic Computer Project, 21 Torrington Square, London, W.C.1)

[Received 1 August 1950]

- El algoritmo de Booth es una alternativa eficiente al clásico algoritmo de multiplicación ADD+SHIFT.
- El algoritmo de Booth consiste en realizar sumas de potencias positivas o negativas de la base en función de parejas de bits (q_i q_{i-1}).

q_i	q_{i-1}	Digito de Booth
0	0	0
0	1	+1
1	0	-1
1	1	0

- El primer paso es recodificar el multiplicador (la base) según la tabla



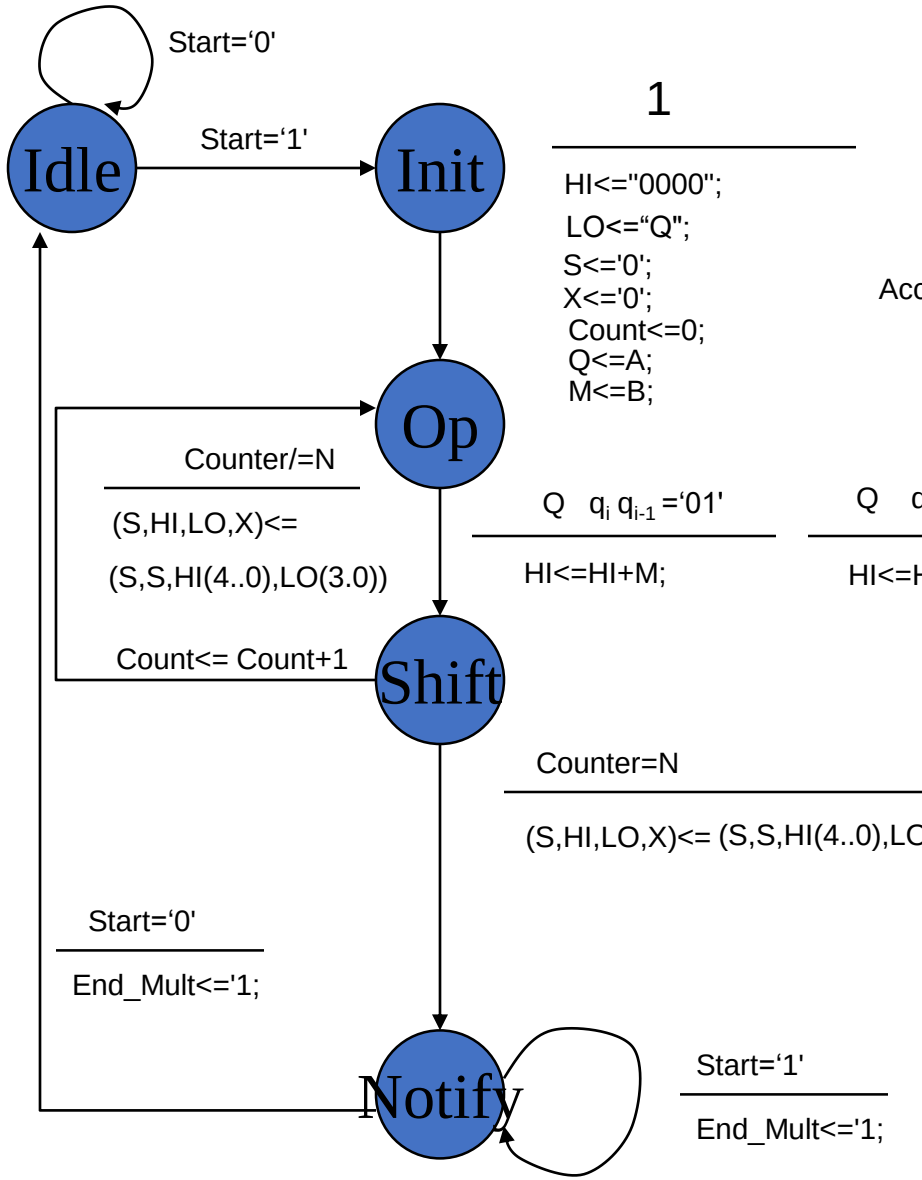
Debe suponerse un bit implícito 0 a la derecha del LSB

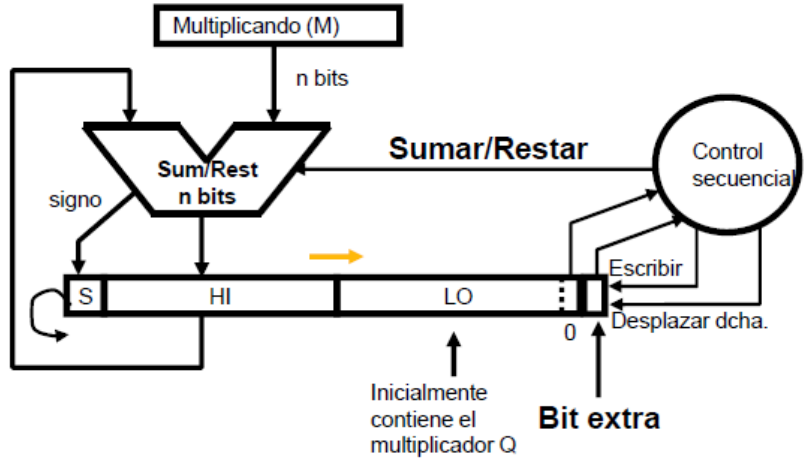
El Algoritmo the Booth

- Funcionamiento:
  - Se multiplica por el numero recodificado de manera que el multiplicando se suma o se resta (sumar el negativ0, Ca2) al resultado parcial desplazado a la izquierda.
  - El MSB del resultado parcial se extiende
  - Ejemplo

							0	1	0	1	1	0	1	(45_{10})
							0	+1	0	0	0	-1	0	(30_{Booth})
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	0	1	0	0	1	1		← Ca2 de multiplicando
0	0	0	0	0	0	0	0	0	0	0	0			
0	0	0	0	0	0	0	0	0	0	0				
0	0	0	0	0	0	0	0	0	0					
0	0	0	1	0	1	1	0	1						
0	0	0	0	0	0	0	0							
0	0	0	0	0	1	0	1	0	0	0	1	1	0	(1350_{10})

FSM Algoritmo the Booth





Ejemplo algoritmo de Booth

N=4

$M=2_{10}=0010_2$

$Q=-7_{10}=1001_2$

Ciclo	Acción	S - HI - LO - X
0	Carga de valores (LO<=Q ; HI<=0000; S,X<=0	0 0000 1001 0
1	$q_i\ q_{i-1}$ = 10 : HI <- HI –M	1 1110 1001 0
	Desplazamiento	1 1111 0100 1
2	$q_i\ q_{i-1}$ = 01: HI <- HI + M	0 0001 0100 1
	Desplazamiento	0 0000 1010 0
3	$q_i\ q_{i-1}$ = 00: HI <- HI	0 0000 1010 0
	Desplazamiento	0 0000 0101 0
4	$q_i\ q_{i-1}$ = 10: HI <- HI –M	1 1110 0101 0
	Desplazamiento	1 1111 0010 1

con extensión

```
  0 0 0 0
- 0 0 1 0
---------
  1 1 1 1 0
```

```
    1 1 1 1
+   0 0 1 0
-----------
  0 0 0 0 1
```

-14_{10}

ASM Algoritmo the Booth

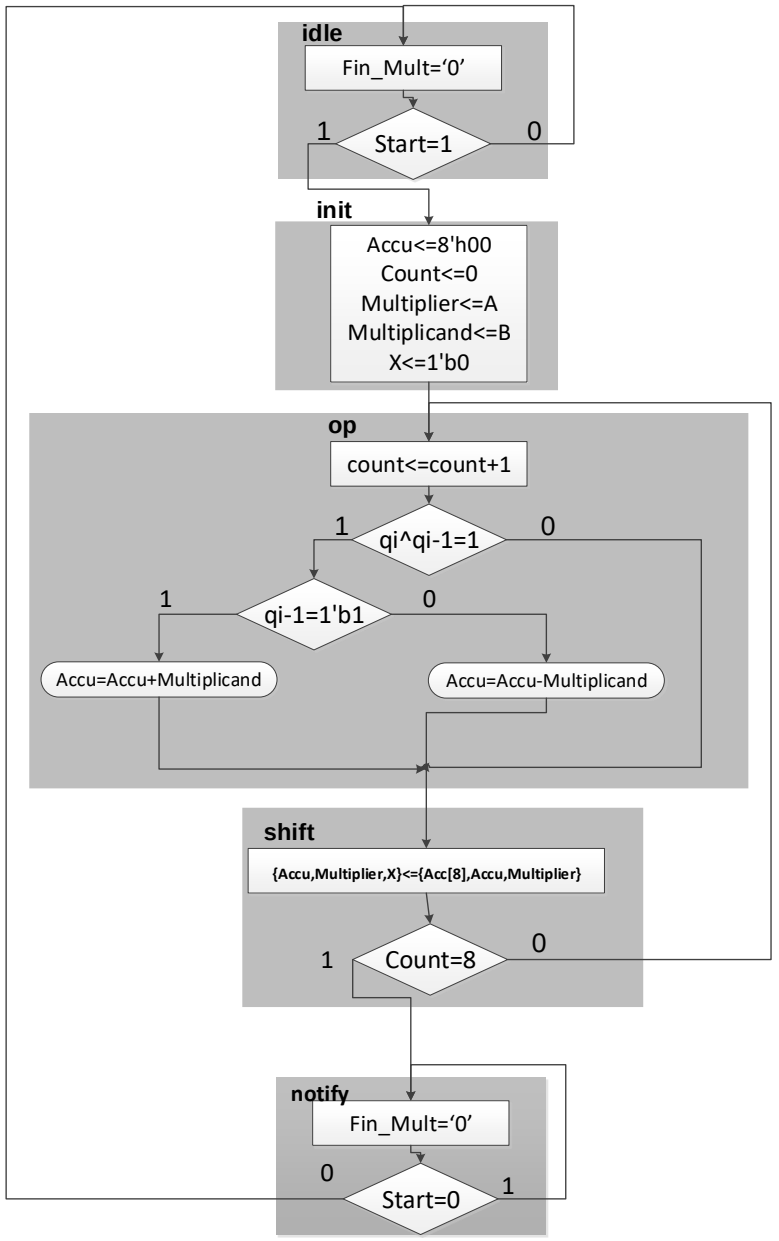
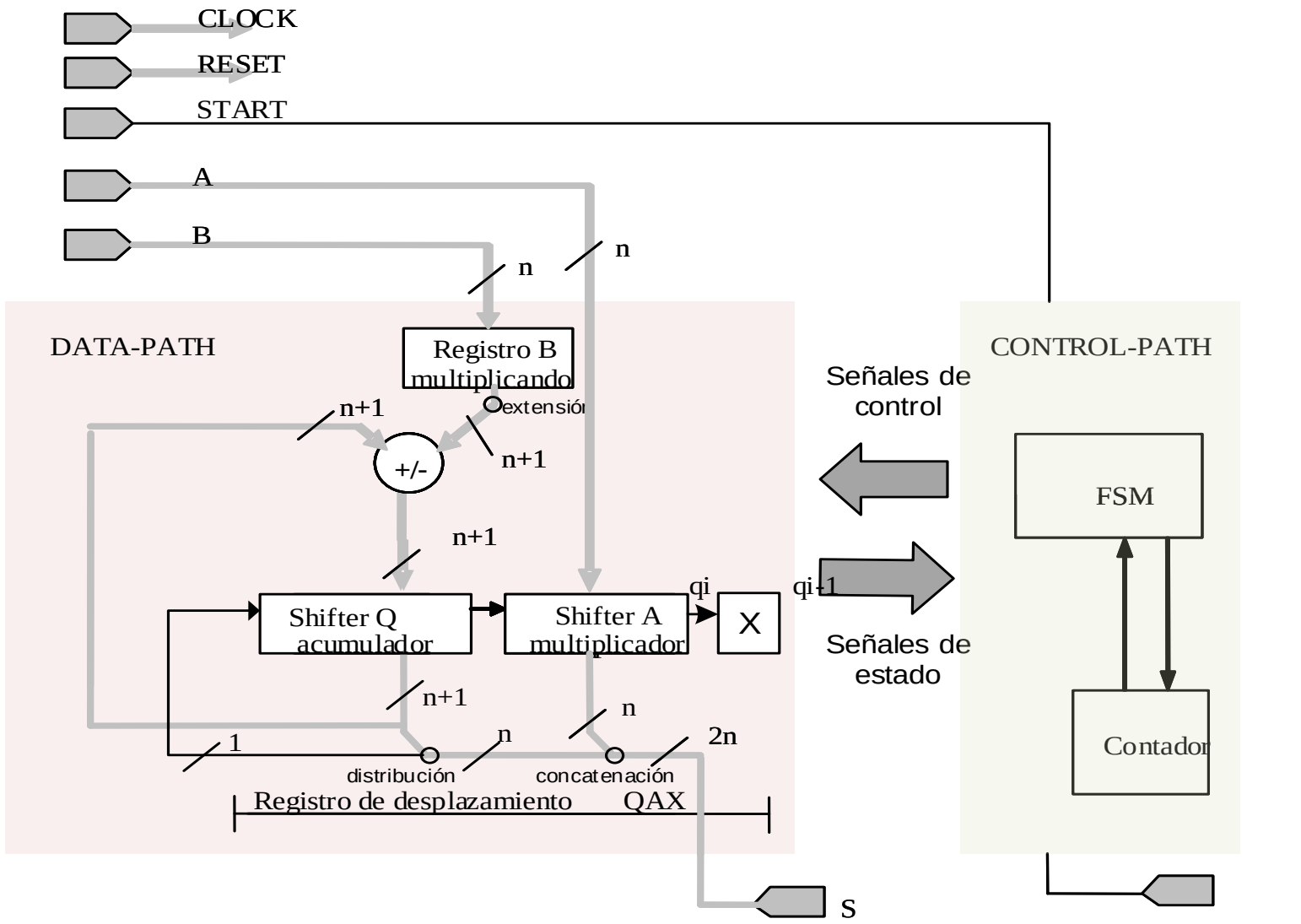


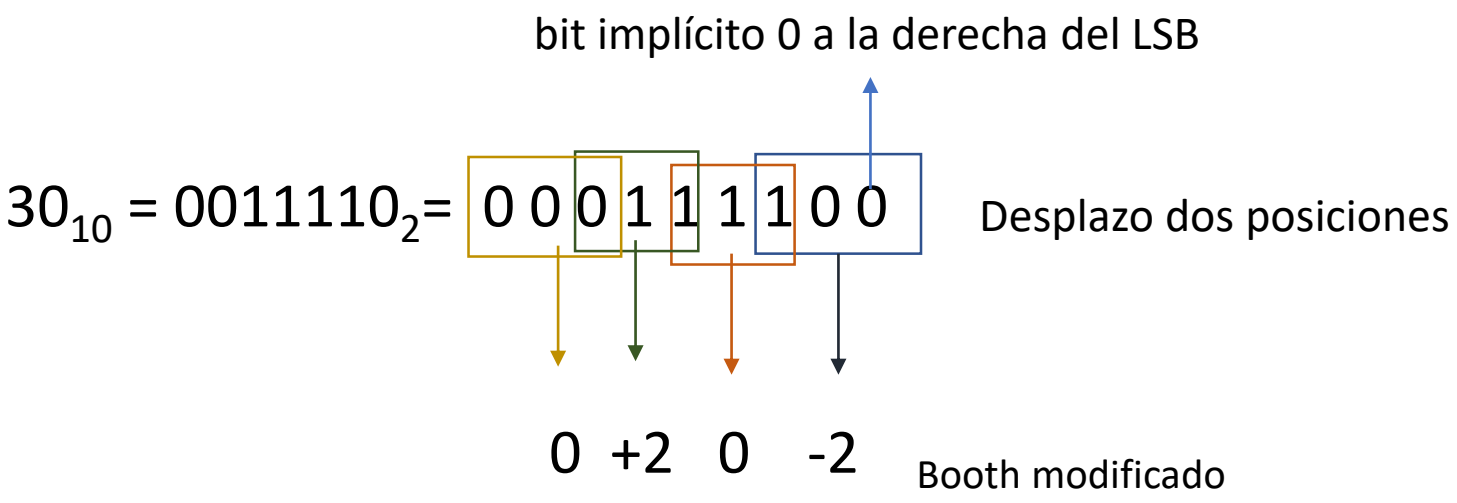
Diagrama de bloques de la solución



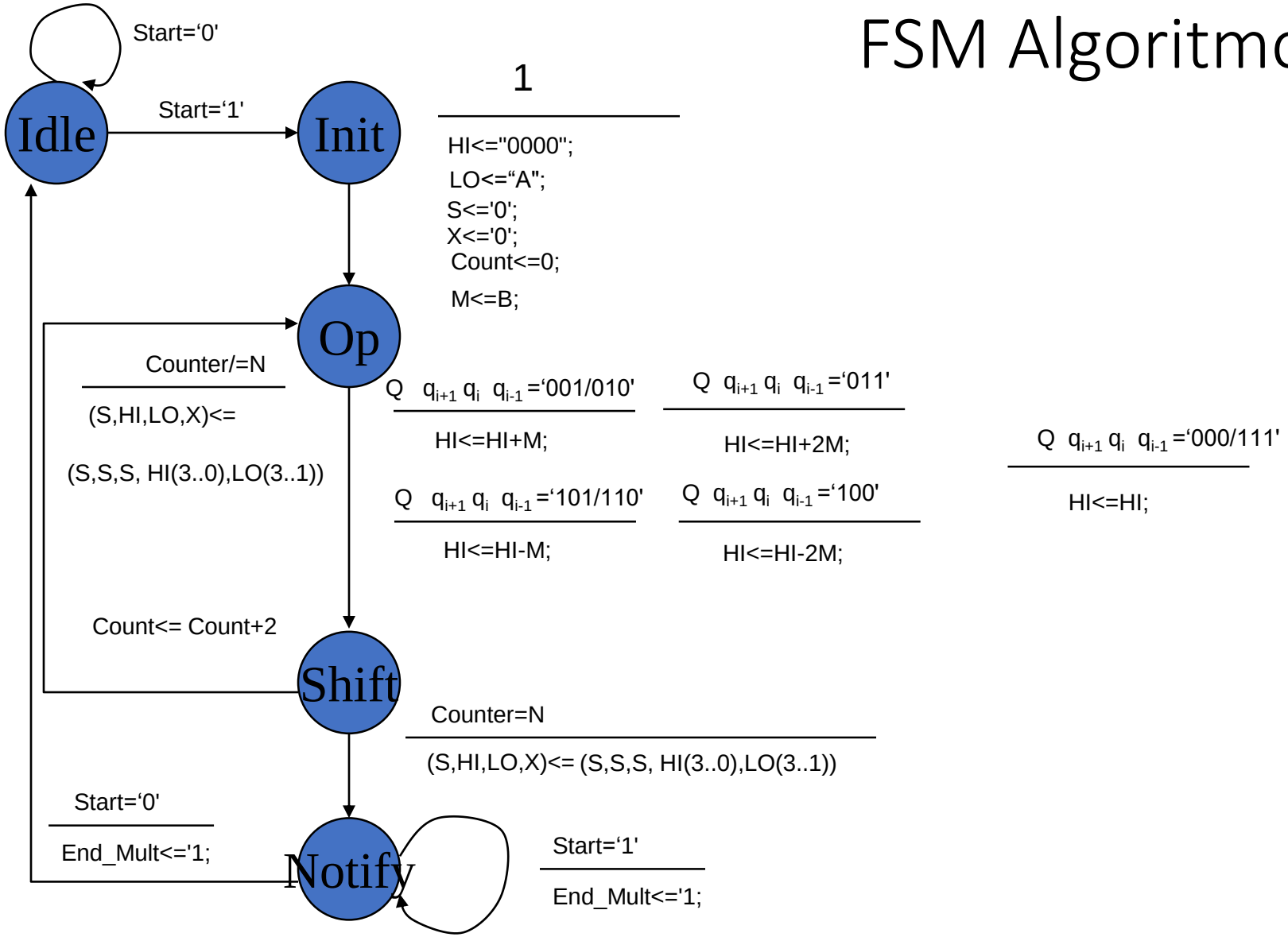
El Algoritmo the Booth modificado

- Recodificación por **parejas de bits** para reducir a la mitad el número de ciclos

q_{i+1}	q_i	q_{i-1}	Digito de Booth
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0



FSM Algoritmo the Booth modificado



Ejemplo algoritmo de Booth modificado

N= 6

B= 13_{10}= 001101_2

A= -6_{10} = 111010_2

Ciclo	Acción	S - HI - LO - X
0	Carga de valores (LO<=A ; HI<=0000; S,X<=0;M<=B)	0 000000 111010 0
1	$q_{i+1}\, q_i\, q_{i-1}$ = 100 : HI <- HI –2M con extension	1 100110 111010 0
	Desplazamiento 2 bits	1 111001 101110 1
2	$q_{i+1}\, q_i\, q_{i-1}$ = 101: HI <- HI - M	1 101100 101110 1
	Desplazamiento 2 bits	1 111011 001011 1
3	$q_{i+1}\, q_i\, q_{i-1}$ = 111: HI <- HI	1 111011 001011 1
	Desplazamiento 2 bits	1 111110 110010 1

Accu

```
  0 0 0 0 0 0
- 0 0 1 1 0 1
-----------
  1 1 0 0 1 1
- 0 0 1 1 0 1
-----------
  1 0 0 1 1 0
```

```
  1 1 1 0 0 1
- 0 0 1 1 0 1
-----------
  1 0 1 1 0 0
```

-78_{10}

ASM Algoritmo the Booth modificado

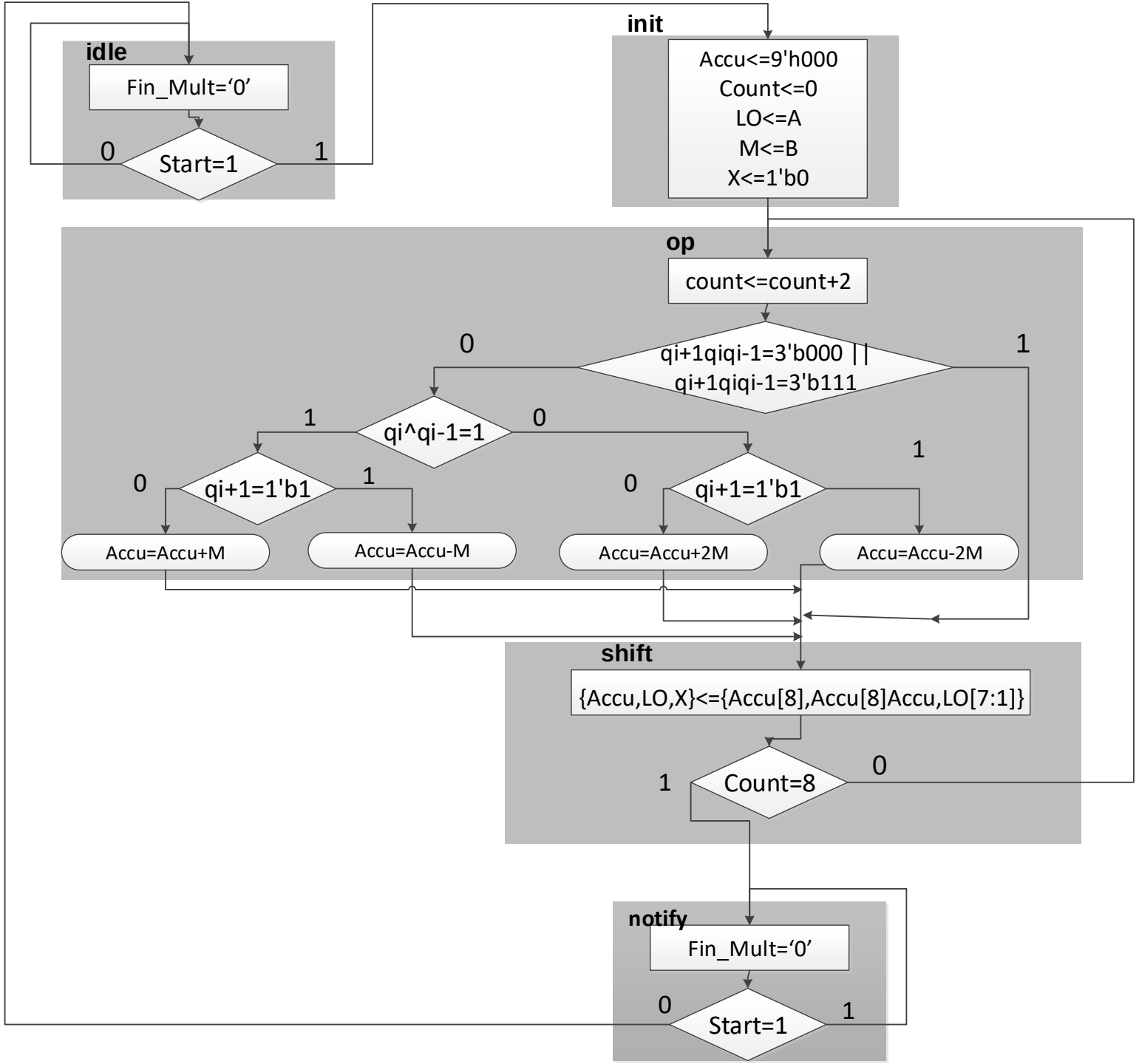
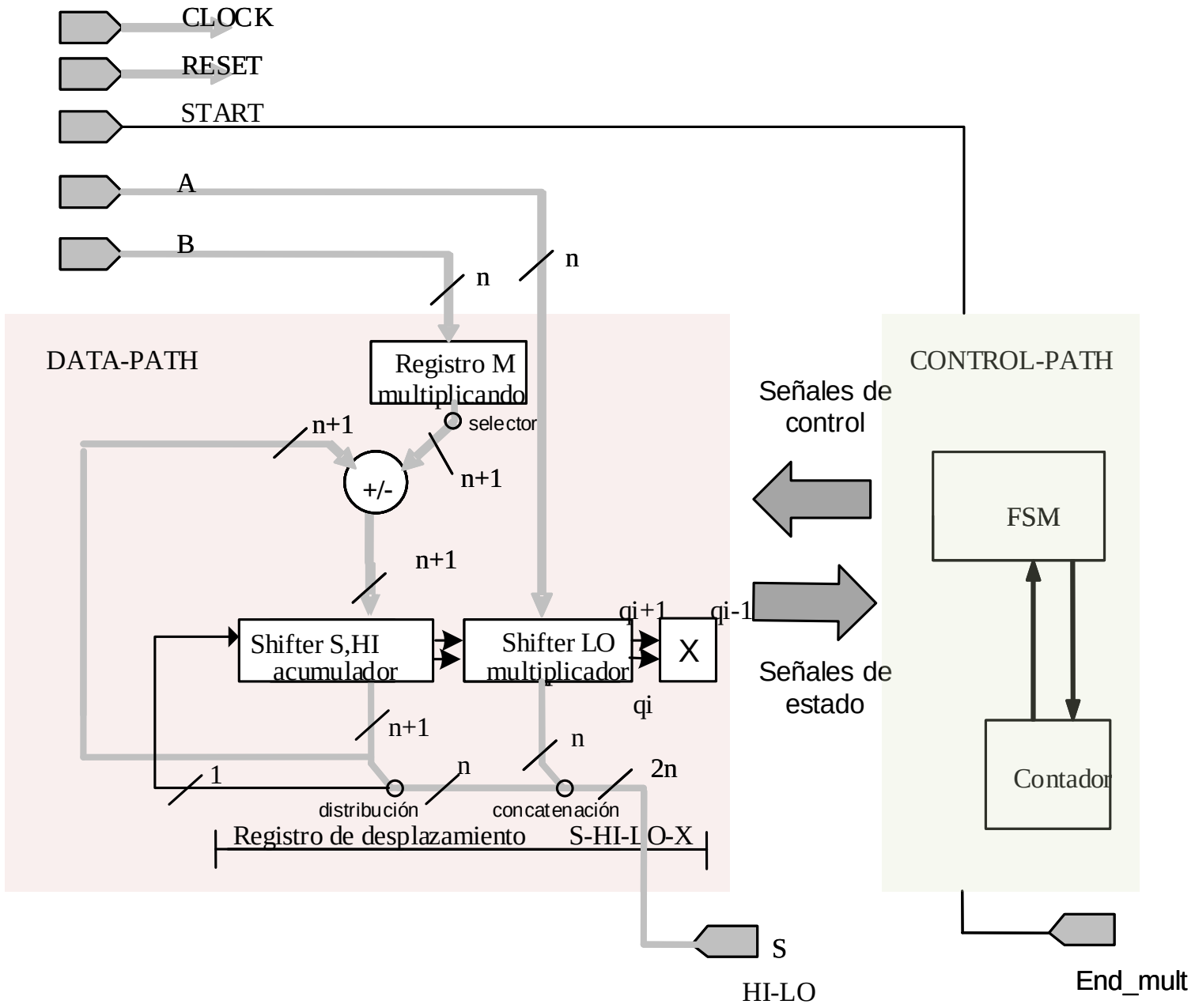
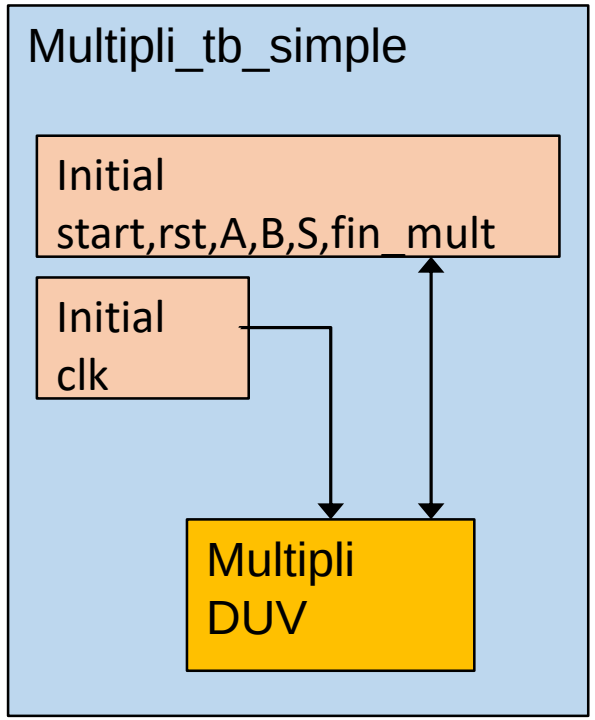


Diagrama de bloques de la solución



Verificación Básica – Nivel 0




```
`timescale 1 ns / 1 ps
module multipli_tb_simple_gateLevel;

parameter tamano=8;

reg clk;
reg [tamano-1:0] A,B;
logic fin_mult;
logic start;
reg rst;
wire [(2*tamano)-1:0] S;

initial begin clk=1'b0;
forever #50 clk=!clk;
end

initial begin
rst=1;
start= 0;
#1 rst=0;
#500 rst=1;
#100 A=8'd100; //100*2
B=8'd2;
start=1;
@ (posedge fin_mult);
@ (negedge clk);
#50 start=0;
#100 A=8'd10; //10*3
B=8'd3;
start=1;
@ (posedge fin_mult);
```

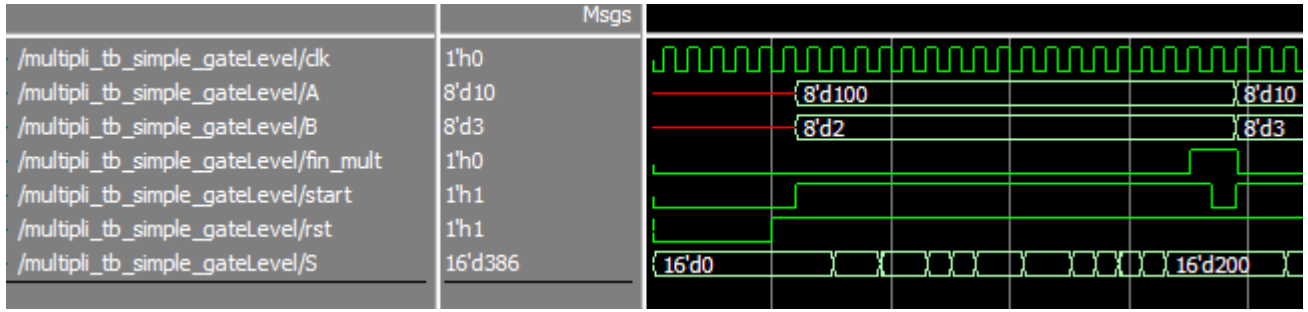
… resto de casos

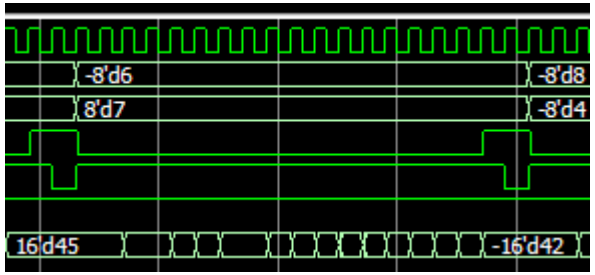
```
$stop;
end

multipli DUV
(.CLOCK(clk),
 .RESET(rst),
 .START(start),
 .A(A),
 .B(B),
 .END_MULT(fin_mult),
 .S(S)
);

endmodule
```

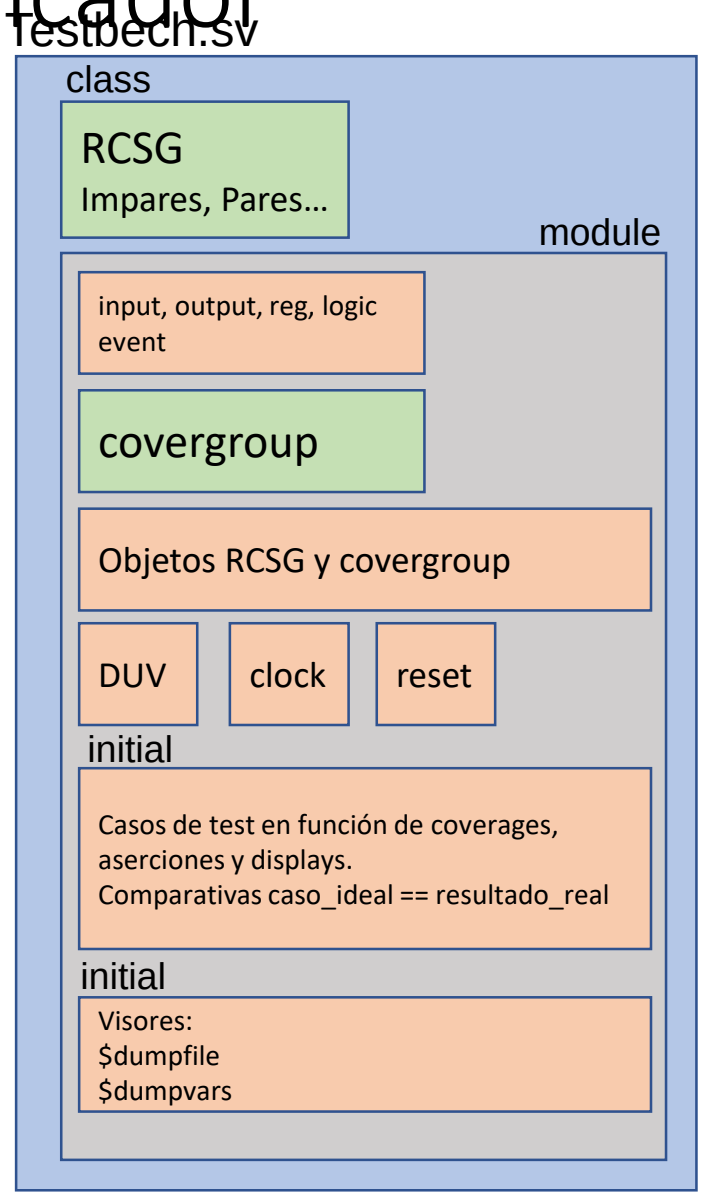
Verificación de resultados





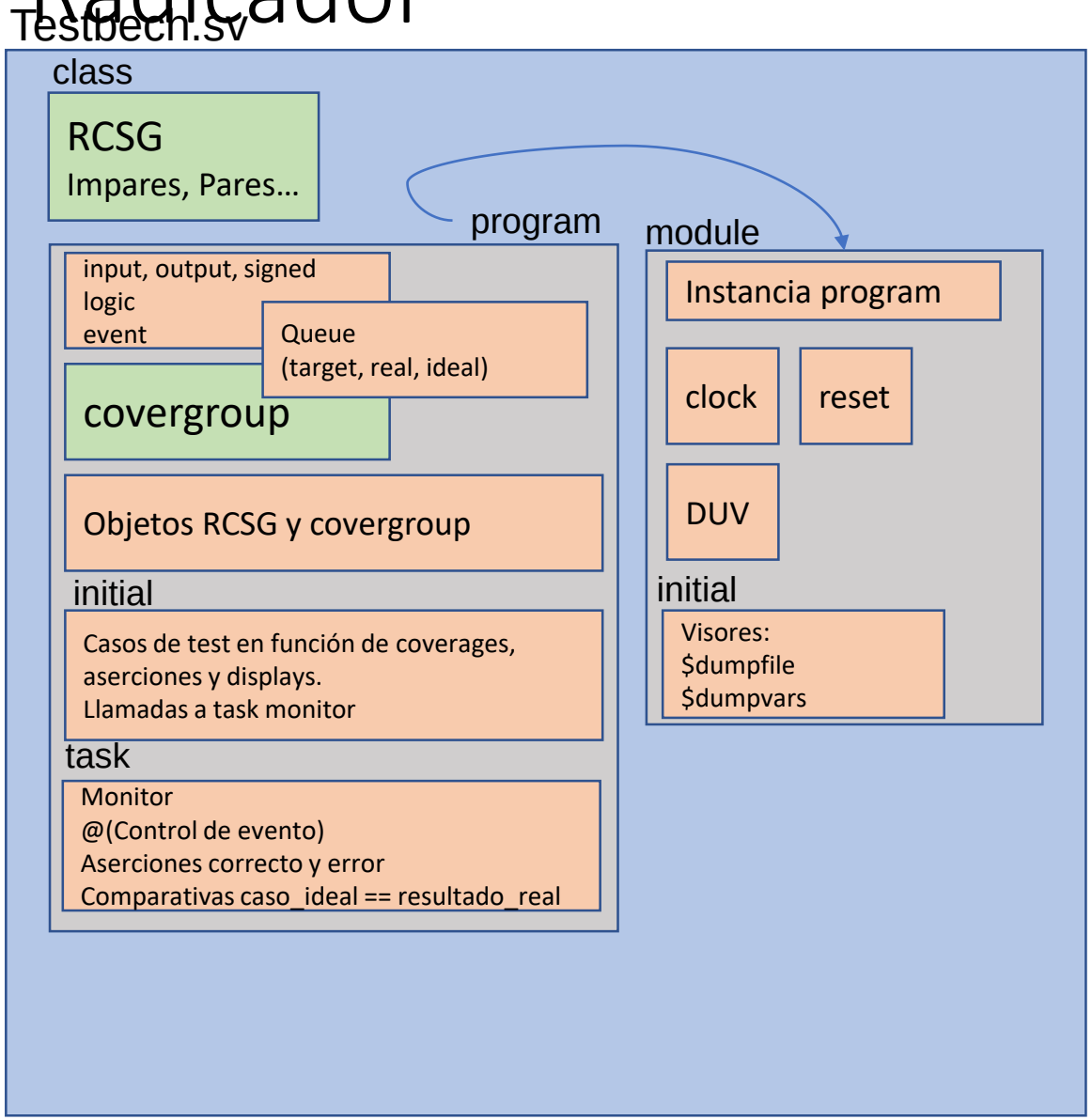
Verificación sin aserciones
Basado en visualización de formas de onda
Muchos casos sin cubrir
No automatizada
No optimizada
Peligro de mal funcionamiento

Verificación Básica II – Basada en Radicador



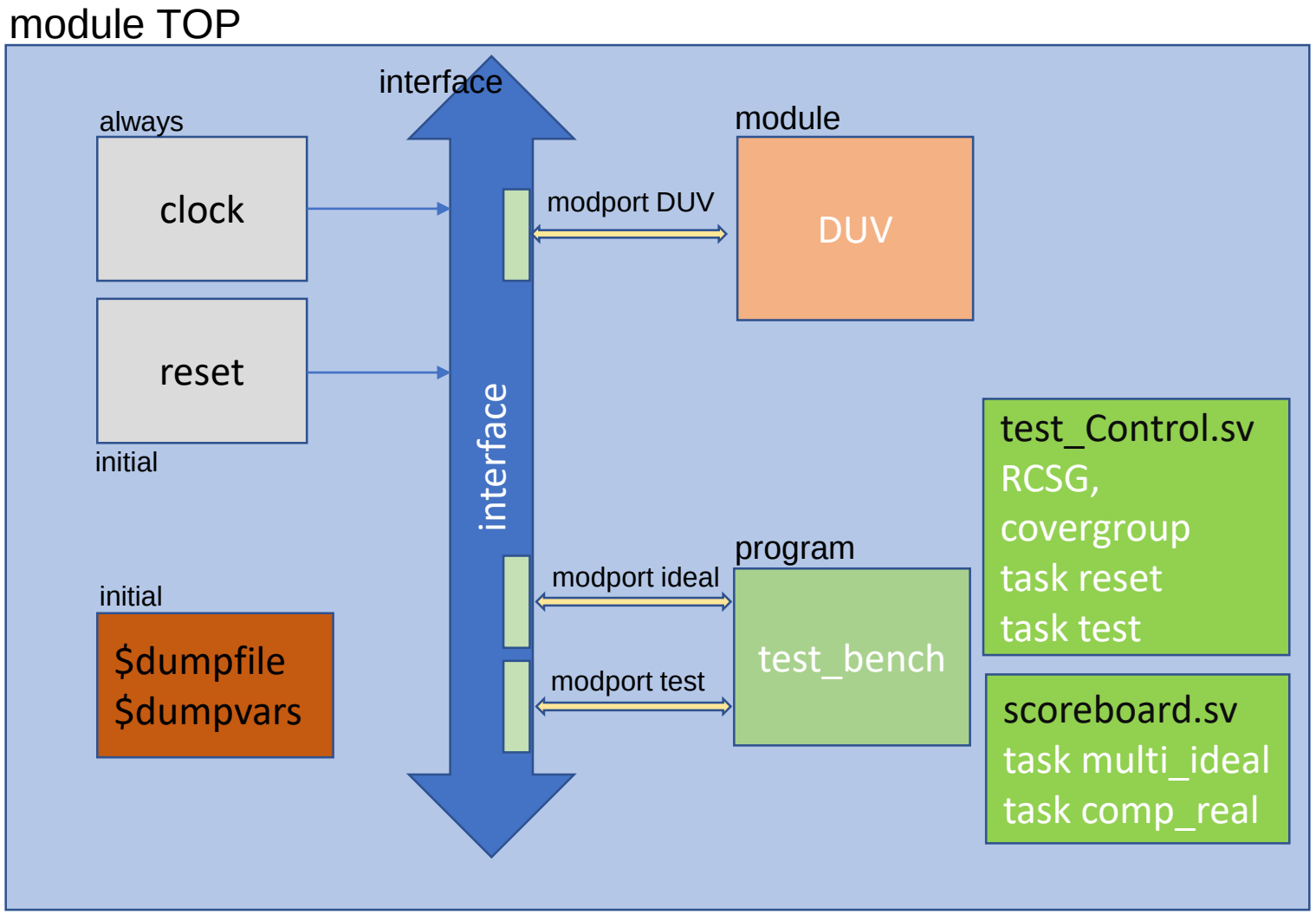
- Características
  - Fichero de test único
  - Incluye generación de estímulos aleatoria RCSG
  - Incluye covergroups
  - Instancia DUV
  - Genera clock y reset en el mismo fichero
  - Los casos de test los agrupa en un initial con aserciones y gestión de eventos para comprobación
  - La visualización de datos con initial al final del fichero
- Fichero largo y poco genérico
- Muestra las posibilidades futuras de estructurar la verificación

Verificación Media – Basada en Radicador

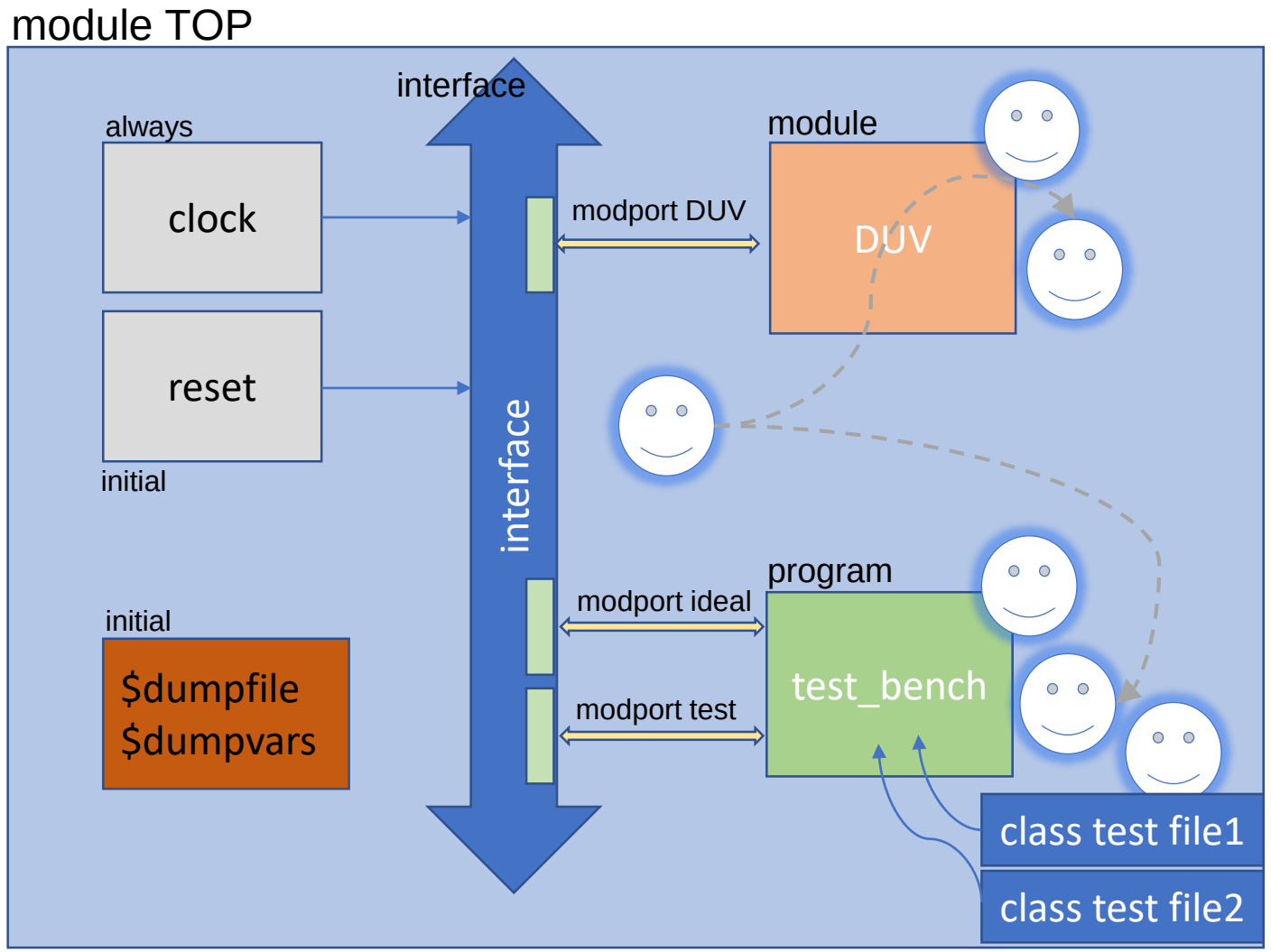


- Características adicionales
  - Agrupación de testbench en PROGRAM
  - Instancia program en module
  - Introduce colas para tratar valores ideales A*B.
  - Uso básico de colas sin embargo
- Todavía un único fichero .sv
- Escaso aprovechamiento de clases.
- No se utilizan interfaces para facilitar la conectividad

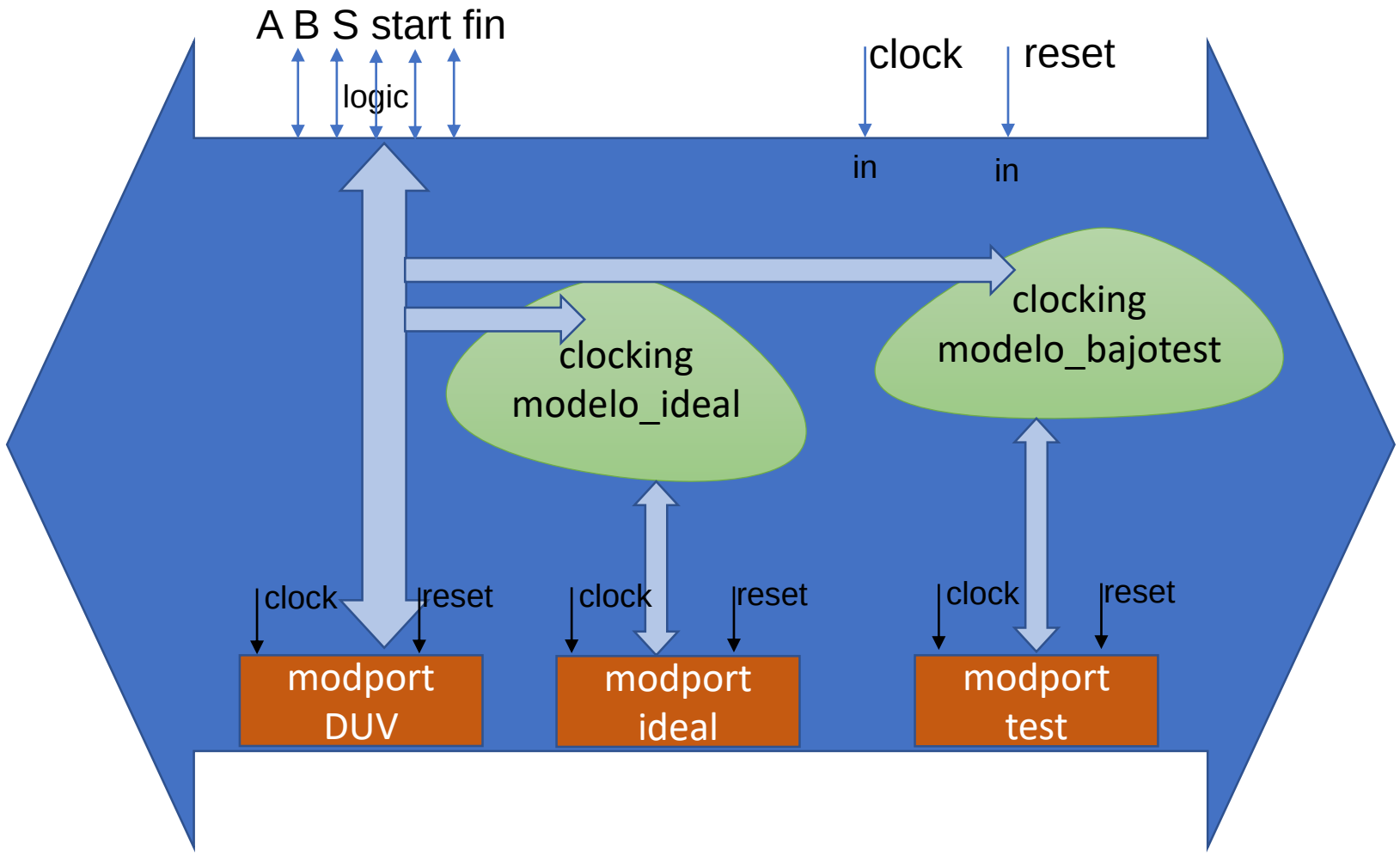
Verificación Compleja - TOP



Verificación Compleja – Equipo de Trabajo



Verificación Compleja - Interface



Verificación Compleja - Program

program

```
`includes “programas sv que definen clases e interfaces”
instancia clases de test = new(ideal, test)

initial      //lanza el test
             clases_de_test.task(); //ej…reset, check
…
end
```

Las clases de test descritas en otros ficheros importados son instanciadas (creadas) y arranca test con initial.
Las clases de test se describen a continuación

Verificación Compleja – definición de clases test