

Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica

IE0523 – Circuitos Digitales II

I ciclo 2023

Tarea 02:

Descripción Estructural del Sumador de 4 Bits: Síntesis Automática

Andrés Chaves Vargas - B92198

Grupo 01

Profesor: Enrique Coen

Abril 2023

Índice

1. Descripción Arquitectónica.	1
2. Plan de Pruebas.	1
3. Instrucciones de Utilización de la Simulación.	3
4. Resultados.	4
4.1. Síntesis de Alto Nivel Con Descripción Estructural Genérica.	4
4.2. Descripción Estructural utilizando Biblioteca con Tecnología Comercial.	6
4.3. Verificación de Pruebas utilizando Descripción Estructural.	8
4.4. Verificación de Pruebas utilizando Descripción Estructural con Retardos.	8
4.5. Número de Componentes Utilizados.	9
5. Conclusiones.	9
6. Recomendaciones.	9

Resumen

En esta tarea se obtuvo la descripción estructural de un sumador de 4 bits mediante el uso de la herramienta Yosys. En conjunto con dicha herramienta se utilizaron librerías que permiten obtener diseños utilizando compuertas específicas. A su vez se analizó la simulación obtenida a partir de la descripción estructural realizada mediante la herramienta GTKwave, cuando se contemplan los retardos de las compuertas y cuando no. Se concluyó que puede existir una afectación en el resultado final del sumador si no se ajustan correctamente los retardos de las librerías utilizadas.

1. Descripción Arquitectónica.

A continuación se muestran los diagramas utilizados para cada sumador:

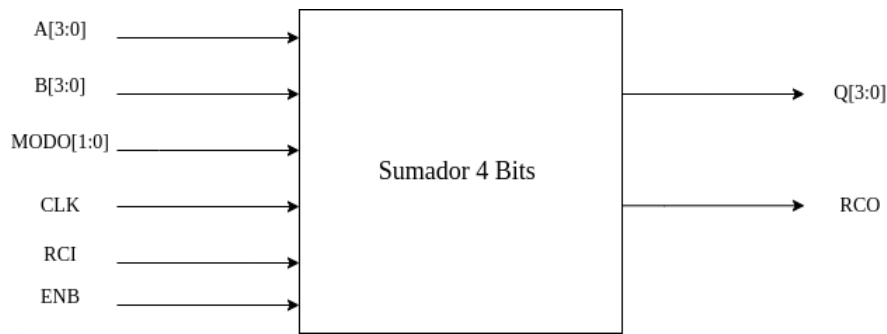


Figura 1: Diagrama Sumador 4 Bits.

En la figura anterior se muestra el bloque principal que permite realizar las operaciones esperadas.

Estado Actual	Próximo Estado			
	MODO = 00	MODO = 01	MODO = 10	MODO = 11
MODO = 00	$Q = Q$	$Q = A + B + RCI$	$Q = A - B + RCI$	$Q = 0$
MODO = 01	$Q = A + B + RCI$	$Q = A + B + RCI$	$Q = A - B + RCI$	$Q = 0$
MODO = 10	$Q = A - B + RCI$	$Q = A + B + RCI$	$Q = A - B + RCI$	$Q = 0$
MODO = 11	$S \ Q = 0$	$Q = A + B + RCI$	$Q = A - B + RCI$	$Q = 0$

2. Plan de Pruebas.

A continuación se muestra el plan de pruebas a implementar:

■ Prueba 1: Suma de 4 Bits.

Esta prueba consiste en utilizar los archivos Sumador4Bits.v probador4Bits.v y test-bench4Bits.v para sumar dos números de 4 bits. Esta prueba se divide en dos casos, en el caso 1 se realizará la suma, con $RCI = 0$. En el caso 2 habilita el valor del $RCI = 1$. Para esta prueba fueron utilizados valores al azar.

El diseño implementado sí pasó esta prueba para ambos casos ya que proporcionó los valores esperados.

Pruebas realizadas:

- Caso: $A + B$ con $RCI = 0$: $MODO = 2'b01$;
 $A = 4'b0010$ $B = 4'b0001$
 $\#10$ $A = 4'b0111$ $B = 4'b0001$
- Caso: $A + B$ con $RCI = 1$:
 $\#5$ $RCI = 1$ $\#5$ $A = 4'b0111$ $B = 4'b0011$

■ Prueba 2: Resta de 4 Bits.

Para esta prueba fueron utilizados los archivos mencionados en la prueba anterior. Esta prueba se divide en tres casos, en el caso 1 se realizó la resta con $RCI = 0$ y siendo $A > B$. En el caso 2 se mantuvo el valor del $RCI = 0$ siendo $B > A$. Finalmente para la tercera prueba se habilitó $RCI = 1$ y utilizando $B > A$. La idea de utilizar $B > A$ es observar el comportamiento del diseño al tener resultados negativos. Al implementar la prueba el diseño pasó de forma correcta la prueba realizada.

- Caso: $A > B$: $\#10$ $MODO = 2'b10$
 $RCI = 0$ $\#5$ $A = 4'b0100$ $B = 4'b0001$
- Caso: $B > A$ con $RCI = 0$:
 $\#10$ $A = 4'b0001$ $B = 4'b0100$
- Caso: $B > A$ con $RCI = 1$:
 $\#5$ $RCI = 1$ $\#5$ $A = 4'b0001$ $B = 4'b0100$

■ Prueba 3: Mantener el valor del MODO 00. Sumador 4 Bits.

La idea de esta prueba es confirmar que al asignar un valor a $MODO = 00$ se mantenga el último valor asignado a Q hasta que exista un cambio en $MODO$. Para esta prueba no se habilitó el RCI . Al realizarla el diseño mantuvo el valor de Q sin ninguna interrupción.

- Caso 1. Mantener el valor del $MODO$ 00.
 $\#10$ $MODO = 2'b01$
 $RCI = 0$
 $\#5$ $A = 4'b0100$ $B = 4'b0010$ Se realiza el cambio de $MODO$:
 $\#10$ $MODO = 2'b00$

■ Prueba 4: Mantener el valor cuando $ENB = 0$. Sumador de 4 Bits.

Al igual que para la prueba anterior, se busca que se mantenga el valor de Q al deshabilitar el ENB . El diseño pasó la prueba sin problema. Manteniendo el valor de Q .

- Caso 1. Mantener el valor cuando $ENB = 0$.

#10 MODO = 2'b10

#5 A = 4'b0100 B = 4'b0001

Se realiza el cambio de ENB:

ENB = 0

■ Prueba 5: Limpiar el Contador. Sumador de 4 Bits.

Esta es la última prueba realizada directamente al sumador de 4 bits. En ella se asigna el valor 11 a MODO, de tal forma que $Q = 0$. Y que este valor se mantenga mientras el MODO continúe sin cambiar. La prueba se pasó fue pasada por el diseño implementado.

- Caso 1. Limpiar el contador.

#10 ENB = 1

MODO = 2'b01

A = 4'b1000 B = 4'b0100

Se realiza el cambio de MODO:

#10 MODO = 2'b11

3. Instrucciones de Utilización de la Simulación.

Para poder observar la descripción estructural se creó un archivo Makefile que permite generar dicha descripción. Al utilizar el comando make en la carpeta Tarea02 que se encuentra en el repositorio de GitHub al que se puede ingresar más adelante, se puede ingresar a la herramienta Yosys o bien a la herramienta GTKwave. A continuación se muestra los comandos que debe utilizar y el resultado que debería obtener a partir de dichos comandos:

- *make one*: Le permite obtener el resultado de la descripción estructural mediante el llamado de Yosys con archivo Sumador4Bits.yys.
- *make two*: Permite abrir GTKwave con la simulación correspondiente al testbench4Bits.v que posee retardos.
- *make three*: Permite abrir GTKwave con la simulación correspondiente al testbench4BitsND.v que **no** posee retardos.

make one:

A continuación se muestra el enlace al repositorio de GitHub con los archivos necesarios:

https://github.com/andresnv12/Tareas_CircuitosII

4. Resultados.

4.1. Síntesis de Alto Nivel Con Descripción Estructural Genérica.

La síntesis realizada con descripción estructural genérica comprende los siguientes comandos que deben ser ingresados en Yosys:

- `read_verilog`
- `proc`
- `opt`
- `techmap`

A continuación se muestra el uso de los comandos *read_verilog* y *proc*:

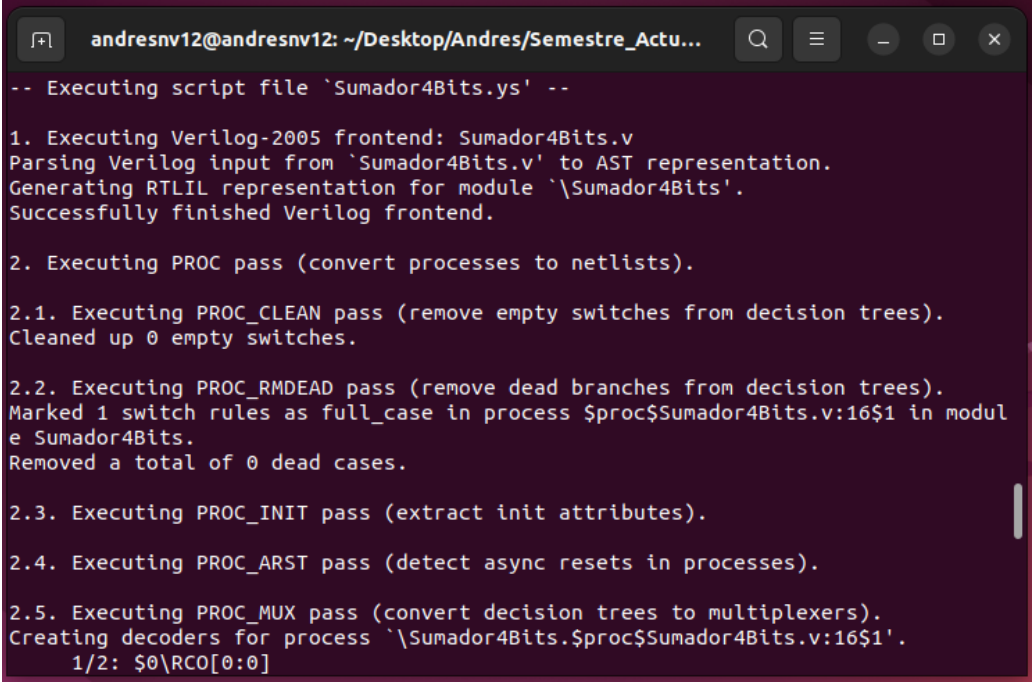
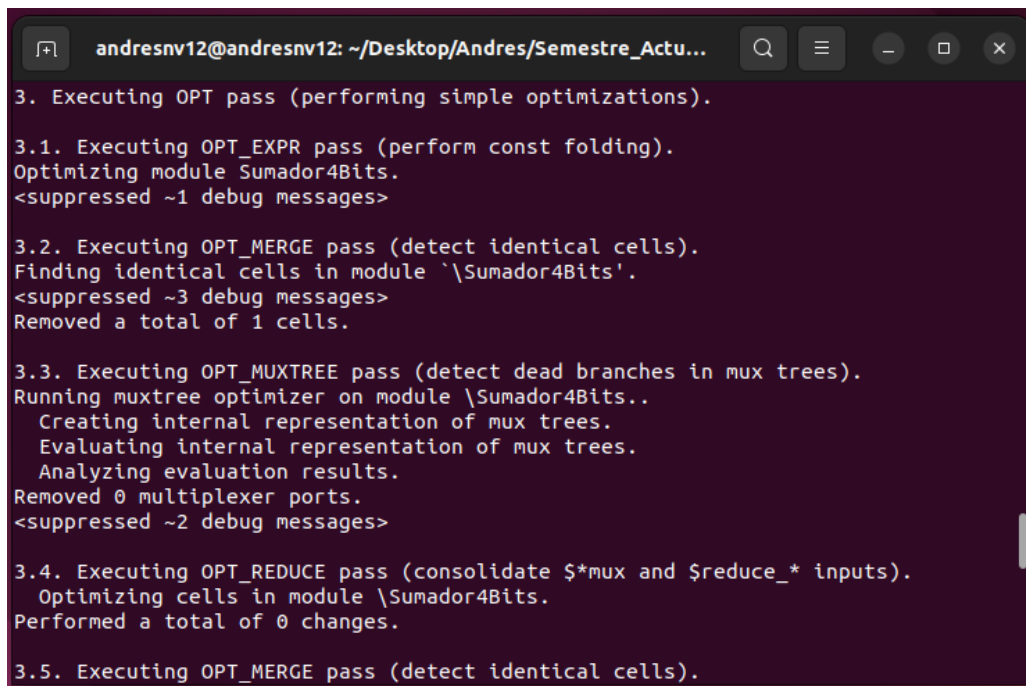
A terminal window with a dark background and light-colored text. The window title is 'andresnv12@andresnv12: ~/Desktop/Andres/Semestre_Actu...'. The terminal output shows the execution of a script file 'Sumador4Bits.yys'. It starts with '-- Executing script file `Sumador4Bits.yys` --'. Then it lists the steps: 1. Executing Verilog-2005 frontend: Sumador4Bits.v. Parsing Verilog input from `Sumador4Bits.v` to AST representation. Generating RTLIL representation for module `Sumador4Bits`. Successfully finished Verilog frontend. 2. Executing PROC pass (convert processes to netlists). 2.1. Executing PROC_CLEAN pass (remove empty switches from decision trees). Cleaned up 0 empty switches. 2.2. Executing PROC_RMDEAD pass (remove dead branches from decision trees). Marked 1 switch rules as full_case in process \$proc\$Sumador4Bits.v:16\$1 in module Sumador4Bits. Removed a total of 0 dead cases. 2.3. Executing PROC_INIT pass (extract init attributes). 2.4. Executing PROC_ARST pass (detect async resets in processes). 2.5. Executing PROC_MUX pass (convert decision trees to multiplexers). Creating decoders for process `Sumador4Bits.\$proc\$Sumador4Bits.v:16\$1`. 1/2: \$0\RC0[0:0]

Figura 2: Inicio de Yosys y Comando *proc*.

En la siguiente imagen se muestra el uso del comando *opt*:



```
andresnv12@andresnv12: ~/Desktop/Andres/Semestre_Actu...
3. Executing OPT pass (performing simple optimizations).

3.1. Executing OPT_EXPR pass (perform const folding).
Optimizing module Sumador4Bits.
<suppressed ~1 debug messages>

3.2. Executing OPT_MERGE pass (detect identical cells).
Finding identical cells in module '\Sumador4Bits'.
<suppressed ~3 debug messages>
Removed a total of 1 cells.

3.3. Executing OPT_MUXTREE pass (detect dead branches in mux trees).
Running muxtree optimizer on module \Sumador4Bits..
  Creating internal representation of mux trees.
  Evaluating internal representation of mux trees.
  Analyzing evaluation results.
Removed 0 multiplexer ports.
<suppressed ~2 debug messages>

3.4. Executing OPT_REDUCE pass (consolidate $*mux and $reduce_* inputs).
Optimizing cells in module \Sumador4Bits.
Performed a total of 0 changes.

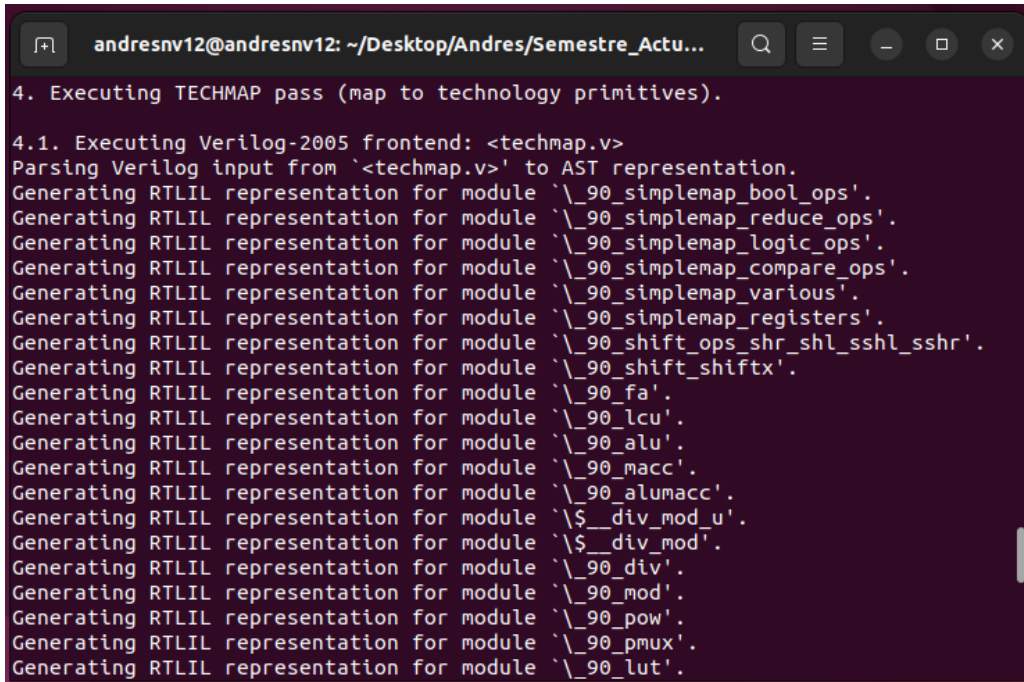
3.5. Executing OPT_MERGE pass (detect identical cells).
```

Figura 3: Uso del Comando *opt*.

Debido al gran tamaño de la descripción estructural generada, no fue posible incluir un archivo que muestre el resultado obtenido mediante el comando *show*. Sin embargo, al ingresar al siguiente link usted podrá observar el resultado obtenido al ser redireccionado al repositorio de GitHub correspondiente:

https://github.com/andresnv12/Tareas_CircuitosII/blob/master/Tarea02/Imagenes/Sumador4BitsNotechmap.svg

La siguiente figura muestra el uso del comando *techmap*:

A terminal window with a dark background and light-colored text. The window title is 'andresnv12@andresnv12: ~/Desktop/Andres/Semestre_Actu...'. The text in the terminal shows the execution of the 'techmap' command, which maps Verilog code to technology primitives. It lists various modules being processed, such as '_90_simplemap_bool_ops', '_90_simplemap_reduce_ops', and '_90_simplemap_logic_ops'.

```
4. Executing TECHMAP pass (map to technology primitives).

4.1. Executing Verilog-2005 frontend: <techmap.v>
Parsing Verilog input from '<techmap.v>' to AST representation.
Generating RTLIL representation for module '_90_simplemap_bool_ops'.
Generating RTLIL representation for module '_90_simplemap_reduce_ops'.
Generating RTLIL representation for module '_90_simplemap_logic_ops'.
Generating RTLIL representation for module '_90_simplemap_compare_ops'.
Generating RTLIL representation for module '_90_simplemap_various'.
Generating RTLIL representation for module '_90_simplemap_registers'.
Generating RTLIL representation for module '_90_shift_ops_shr_shl_sshl_sshr'.
Generating RTLIL representation for module '_90_shift_shiftx'.
Generating RTLIL representation for module '_90_fa'.
Generating RTLIL representation for module '_90_lcu'.
Generating RTLIL representation for module '_90_alu'.
Generating RTLIL representation for module '_90_macc'.
Generating RTLIL representation for module '_90_alumacc'.
Generating RTLIL representation for module '\$_div_mod_u'.
Generating RTLIL representation for module '\$_div_mod'.
Generating RTLIL representation for module '_90_div'.
Generating RTLIL representation for module '_90_mod'.
Generating RTLIL representation for module '_90_pow'.
Generating RTLIL representation for module '_90_pmux'.
Generating RTLIL representation for module '_90_lut'.
```

Figura 4: Uso del Comando *techmap*.

Al igual que el caso anterior, el resultado obtenido no puede ser incluido directamente, no obstante, puede ingresar al siguiente link: https://github.com/andresnv12/Tareas_CircuitosII/blob/master/Tarea02/Imagenes/Sum4Bitstechmap.svg para observar dicho resultado.

4.2. Descripción Estructural utilizando Biblioteca con Tecnología Comercial.

Al incluir los comandos *dfflibmap -liberty* y *abc -liberty* se puede obtener una descripción estructural con una biblioteca que utilice tecnología comercial. A continuación se muestra el uso de estos comandos:

Uso correcto del comando *dfflibmap -liberty*:


```
andresnv12@andresnv12: ~/Desktop/Andres/Semestre_Actu...
6. Executing DFFLIBMAP pass (mapping DFF cells to sequential cells from liberty
file).
cell DFF (noninv, pins=3, area=18.00) is a direct match for cell type $_DFF_P_
.
create mapping for $_DFF_N_ from mapping for $_DFF_P_.
final dff cell mappings:
DFF _DFF_N_ (.C(~C), .D( D), .Q( Q));
DFF _DFF_P_ (.C( C), .D( D), .Q( Q));
unmapped dff cell: $_DFF_NN0_
unmapped dff cell: $_DFF_NN1_
unmapped dff cell: $_DFF_NP0_
unmapped dff cell: $_DFF_NP1_
unmapped dff cell: $_DFF_PN0_
unmapped dff cell: $_DFF_PN1_
unmapped dff cell: $_DFF_PP0_
unmapped dff cell: $_DFF_PP1_
unmapped dff cell: $_DFFSR_NNN_
unmapped dff cell: $_DFFSR_NNP_
unmapped dff cell: $_DFFSR_NPN_
unmapped dff cell: $_DFFSR_NPP_
unmapped dff cell: $_DFFSR_PNN_
unmapped dff cell: $_DFFSR_PNP_
unmapped dff cell: $_DFFSR_PPN_
```

Figura 5: Uso del Comando *dfflibmap -liberty*.

Uso correcto del comando *abc -liberty*:

```
andresnv12@andresnv12: ~/Desktop/Andres/Semestre_Actu...
6. Executing DFFLIBMAP pass (mapping DFF cells to sequential cells from liberty
file).
cell DFF (noninv, pins=3, area=18.00) is a direct match for cell type $_DFF_P_
.
create mapping for $_DFF_N_ from mapping for $_DFF_P_.
final dff cell mappings:
DFF _DFF_N_ (.C(~C), .D( D), .Q( Q));
DFF _DFF_P_ (.C( C), .D( D), .Q( Q));
unmapped dff cell: $_DFF_NN0_
unmapped dff cell: $_DFF_NN1_
unmapped dff cell: $_DFF_NP0_
unmapped dff cell: $_DFF_NP1_
unmapped dff cell: $_DFF_PN0_
unmapped dff cell: $_DFF_PN1_
unmapped dff cell: $_DFF_PP0_
unmapped dff cell: $_DFF_PP1_
unmapped dff cell: $_DFFSR_NNN_
unmapped dff cell: $_DFFSR_NNP_
unmapped dff cell: $_DFFSR_NPN_
unmapped dff cell: $_DFFSR_NPP_
unmapped dff cell: $_DFFSR_PNN_
unmapped dff cell: $_DFFSR_PNP_
unmapped dff cell: $_DFFSR_PPN_
```

Figura 6: Uso del Comando *abc -liberty*.

Luego de aplicar dichos comandos, se obtiene la descripción estructural utilizando la biblioteca comercial *cmos_cells.lib*. En el siguiente link se puede observar el resultado obtenido:

https://github.com/andresnv12/Tareas_CircuitosII/blob/master/Tarea02/Imagenes/Sumador4BitsLibF.svg

4.3. Verificación de Pruebas utilizando Descripción Estructural.

Mediante el comando *write_verilog* se puede obtener un Sumador4Synth.v que realice la misma función que el archivo Sumador4Bits.v, para verificar que este archivo genera el mismo resultado, se debió editar el archivo testbench4BitsND.v para incluya la librería *cmos_cell_no_delay.lib*.

Luego, haciendo uso de un archivo Makefile se puede obtener la simulación mediante GTKwave, la cual permite observar que se obtiene el mismo resultado que cuando se incluye el archivo Sumador4Bits.v.

A continuación se muestra el resultado de la simulación realizada:

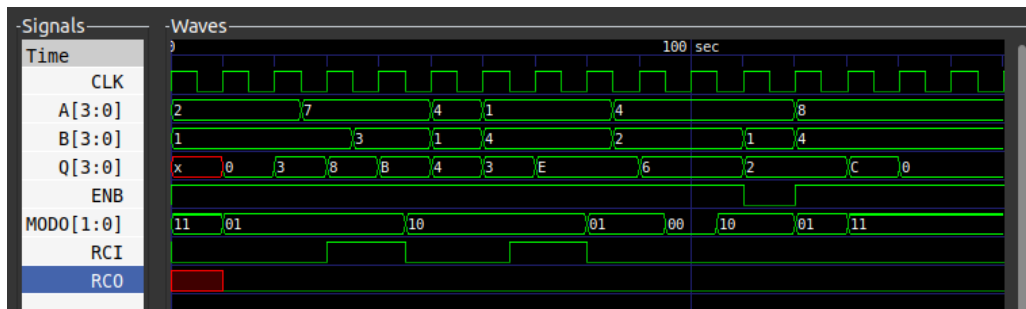


Figura 7: Simulación Realizada con Librería sin Retardo.

4.4. Verificación de Pruebas utilizando Descripción Estructural con Retardos.

Luego se corre el archivo testbench4Bits.v de tal forma que se tenga una simulación con retardo. Al utilizar el comando make es posible observar el resultado de la simulación realizada. Se puede notar que existe una pequeña diferencia con respecto a la simulación sin retardo, sin embargo, no existe una afectación ya que los retardos son relativamente pequeños, comparados con el tiempo que dura el ciclo del reloj.

A continuación se muestra el resultado de la simulación realizada:

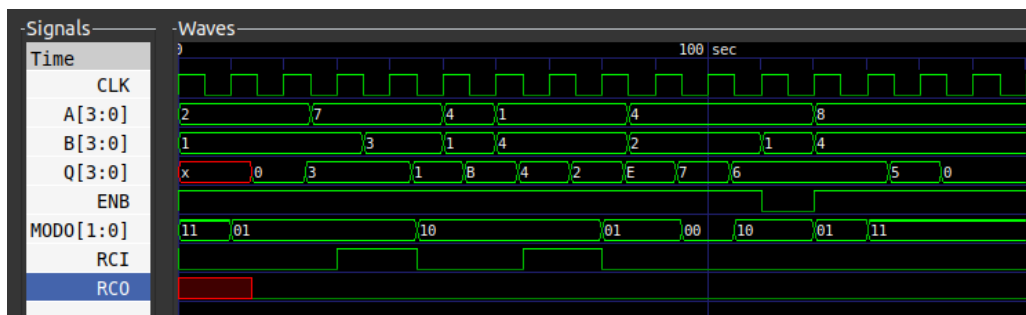


Figura 8: Simulación Realizada con Librería con Retardo.

4.5. Número de Componentes Utilizados.

A continuación se muestran los componentes utilizados para la síntesis realizada:

Tipo de Componente	Número de Componentes
NAND	58
NOR	42
NOT	25
FF	2

5. Conclusiones.

1. Mediante el uso de la herramienta Yosys es posible obtener diferentes diseños para una descripción estructural.
2. Los retardos incluidos pueden afectar el resultado de la simulación en GTKwave.

6. Recomendaciones.

- Se debe tener cuidado al añadir los retardos en las librerías para las compuertas, ya que si los retardos sobrepasan los ciclos de reloj, se tendrán lecturas erróneas en la simulación.
- Es buena idea realizar un archivo de testbench que llame una librería con retardos y otro archivo de testbench que utilice una librería sin retardos, de tal forma que no se tenga que editar un único archivo testbench.v, ya que esto puede generar errores en caso de borrar alguna línea de código.