Universidad del Valle de Guatemala

Facultad de Ingeniería

Departamento de Ingeniería Electrónica

IE2011 - Electrónica Digital I

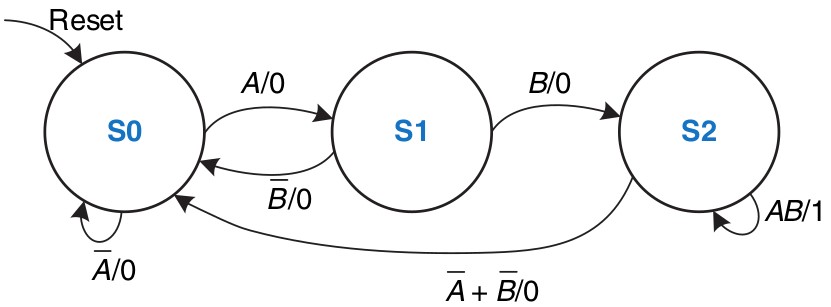
Kurt Kellner

# Laboratorio #06

Este laboratorio será trabajado de forma individual y se entregará de forma digital de acuerdo a la fecha de entrega en Canvas. Haga los ejercicios **SIN** usar una calculadora (a menos que se le indique lo contrario). Deberá identificar su entrega con su nombre, carné y sección.

## Ejercicio 01

Implemente la siguiente Máquina de Estados Finitos



**Figure 1:** Diagrama Ejercicio 01

Su solución debe incluir: Clock Reset

1. Caja Negra

A

B Y

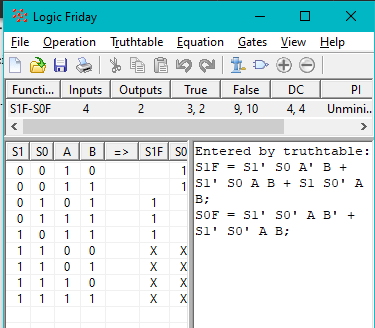
1. Tabla de transiciones de estado sin codificar

|  |  |  |  |
| --- | --- | --- | --- |
| ACTUAL | A | B | SIGUIENTE |
| S0 | 0 | 0 | S0 |
| S0 | 0 | 1 | S0 |
| S0 | 1 | 0 | S1 |
| S0 | 1 | 1 | S1 |
| S1 | 0 | 0 | S0 |
| S1 | 0 | 1 | S0 |
| S1 | 1 | 0 | S2 |
| S1 | 1 | 1 | S2 |
| S2 | 0 | 0 | S0 |
| S2 | 0 | 1 | S0 |
| S2 | 1 | 0 | S0 |
| S2 | 1 | 1 | S2 |

1. Tabla de transiciones de estado codificada

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S1 | S0 | A | B | S1’ | S0’ |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |

1. Screenshots de Logic Friday resolviendo las tablas

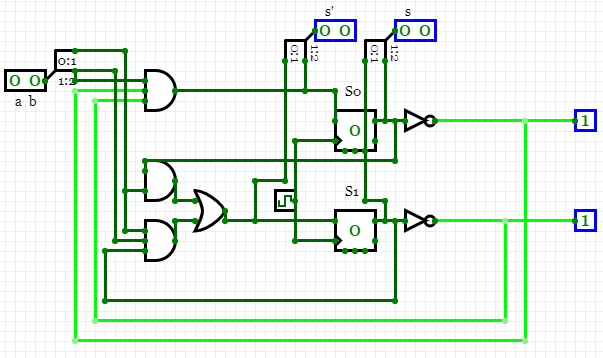


1. Ecuaciones booleanas

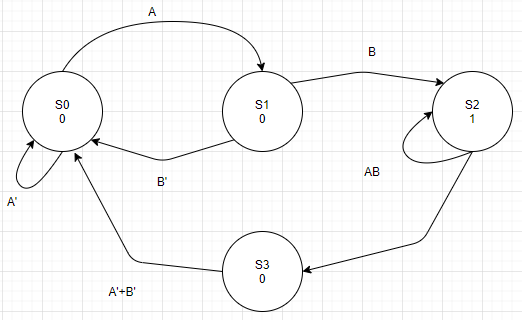
S1f = S0 B + S1 A B;

S0f = S1' S0' A

1. Implementación completa en CircuitVerse.



## Ejercicio 02

Re-dibuje el diagrama del Ejercicio 01 como una FSM de Moore. **Únicamente haga el diagrama**.

i

## Ejercicio 03

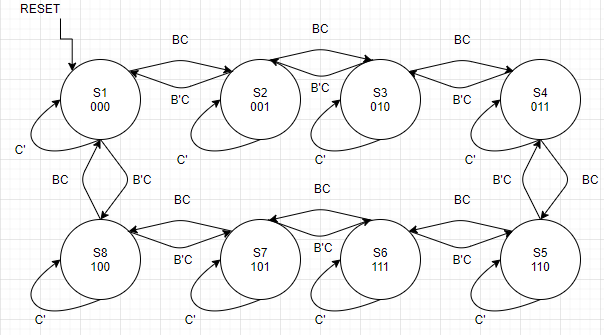
Diseñe una FSM de Moore que represente un contador Gray de 3 bits módulo 8. Un contador cambia de valor en el flanco de reloj. La escala de Gray tiene la característica que sólo cambia un bit a la vez. Un contador módulo *N* cuenta de 0 hasta *N-1* y luego vuelve a empezar. El conteo debe ser de la siguiente forma:

|  |  |
| --- | --- |
| Número | Código Gray |
| 0 | 000 |
| 1 | 001 |
| 2 | 011 |
| 3 | 010 |
| 4 | 110 |
| 5 | 111 |
| 6 | 101 |
| 7 | 100 |

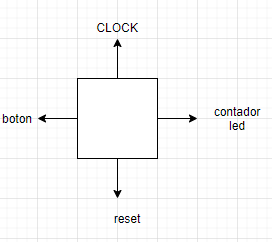
Adicionalmente su contador debe tener una entrada de 1 bit (*UP*/*DOWN*) que indique la dirección de conteo (hacia arriba o hacia abajo). Si la entrada = 1 entonces el contador aumenta. Si es 0 entonces el contador disminuye.

Su solución debe incluir:

1. Diagrama de transiciones de estado (implementado en una herramienta como <https://app.diagrams.net/>)



1. Caja Negra



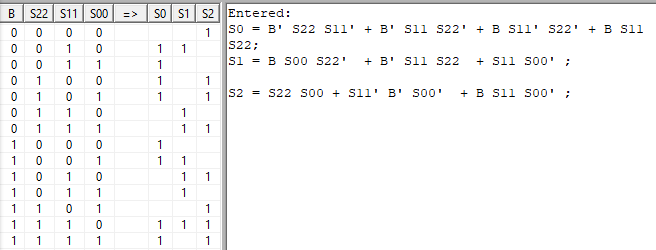
1. Tabla de transiciones de estado sin codificar

|  |  |  |  |
| --- | --- | --- | --- |
| Actual | Clock | Boton | Siguiente |
| S0 | 0 | X | S0 |
| S0 | 1 | 0 | S7 |
| S0 | 1 | 1 | S1 |
| S1 | 0 | X | S1 |
| S1 | 1 | 0 | S0 |
| S1 | 1 | 1 | S2 |
| S2 | 0 | X | S2 |
| S2 | 1 | 0 | S1 |
| S2 | 1 | 1 | S3 |
| S3 | 0 | X | S3 |
| S3 | 1 | 0 | S2 |
| S3 | 1 | 1 | S4 |
| S4 | 0 | X | S4 |
| S4 | 1 | 0 | S3 |
| S4 | 1 | 1 | S5 |
| S5 | 0 | X | S5 |
| S5 | 1 | 0 | S4 |
| S5 | 1 | 1 | S6 |
| S6 | 0 | X | S6 |
| S6 | 1 | 0 | S5 |
| S6 | 1 | 1 | S7 |
| S7 | 0 | X | S7 |
| S7 | 1 | 0 | S6 |
| S7 | 1 | 1 | S0 |

1. Tabla de transiciones de estado codificada

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| B | S2 | S1 | S0 | S0’ | S1’ | S2’ |
| 0 | 0 | 0 | 0 | X | X | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | X |
| 0 | 0 | 1 | 1 | 1 | X | X |
| 0 | 1 | 0 | 0 | 1 | X | 1 |
| 0 | 1 | 0 | 1 | 1 | X | 1 |
| 0 | 1 | 1 | 0 | X | 1 | X |
| 0 | 1 | 1 | 1 | X | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | X | X |
| 1 | 0 | 0 | 1 | 1 | 1 | X |
| 1 | 0 | 1 | 0 | X | 1 | 1 |
| 1 | 0 | 1 | 1 | X | 1 | X |
| 1 | 1 | 0 | 1 | X | X | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | X | 1 |

1. Screenshots de **Logic Friday** resolviendo las tablas



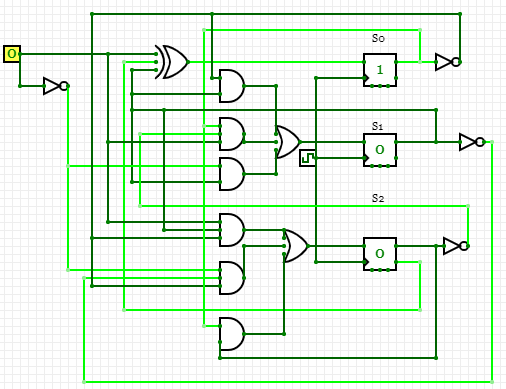
1. Ecuaciones booleanas

S0 = B' S22 S11' + B' S11 S22' + B S11' S22' + B S11 S22;

S1 = B S00 S22' + B' S11 S22 + S11 S00' ;

S2 = S22 S00 + S11' B' S00' + B S11 S00' ;

1. Implementación completa en CircuitVerse.



## Ejercicio 04

Lea la sección 4.5.4 (toda la sección) de su libro (página 205) y explique en sus propias palabras qué es **nonblocking assignment**, cuál es la diferencia entre **non-blocking** y **blocking** assignment y en qué situaciones debe utilizarse cada uno. Incluya ejemplos (no es necesario hacer un testbench).

Ya que intente leer el libro y no entendí nada :’c investigue en internet y según <https://www.nandland.com/articles/blocking-nonblocking-verilog.html#:~:text=In%20Verilog%2C%20if%20you%20want,in%20the%20same%20always%20block.> Por lo que entendí nonblocking assignment nos sirve para poder crear códigos que se basen en un funcionamiento secuencial.

De esta manera nuestra diferencia es que usar nonblocking assignment no deja que una función se realice si la anterior no ha terminado de ejecutarse mientras que blocking assignment nos permite que toda la lógica pueda ser ejecutada al mismo tiempo.

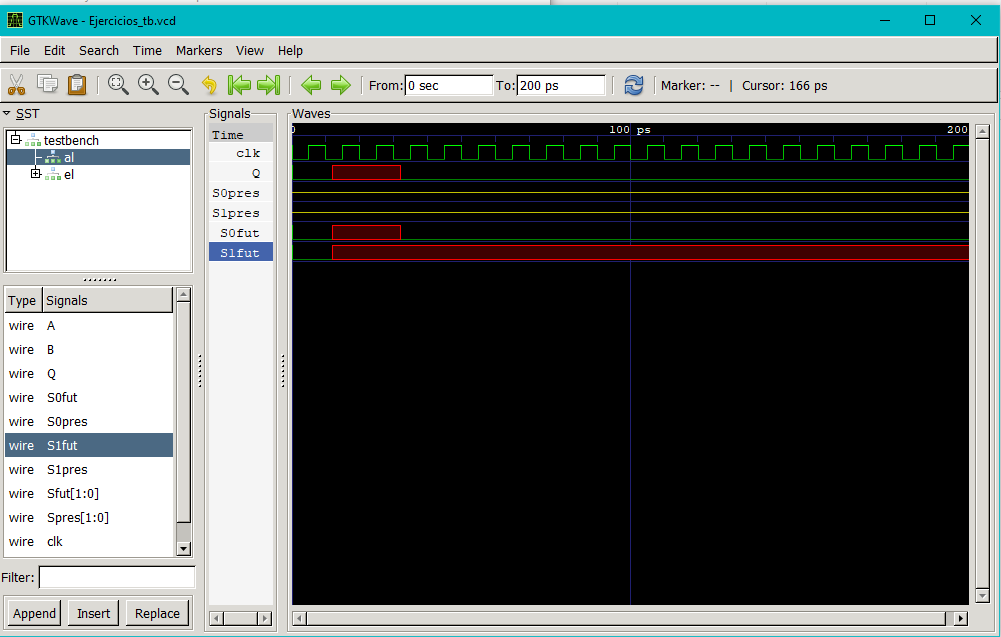
## Ejercicio 05

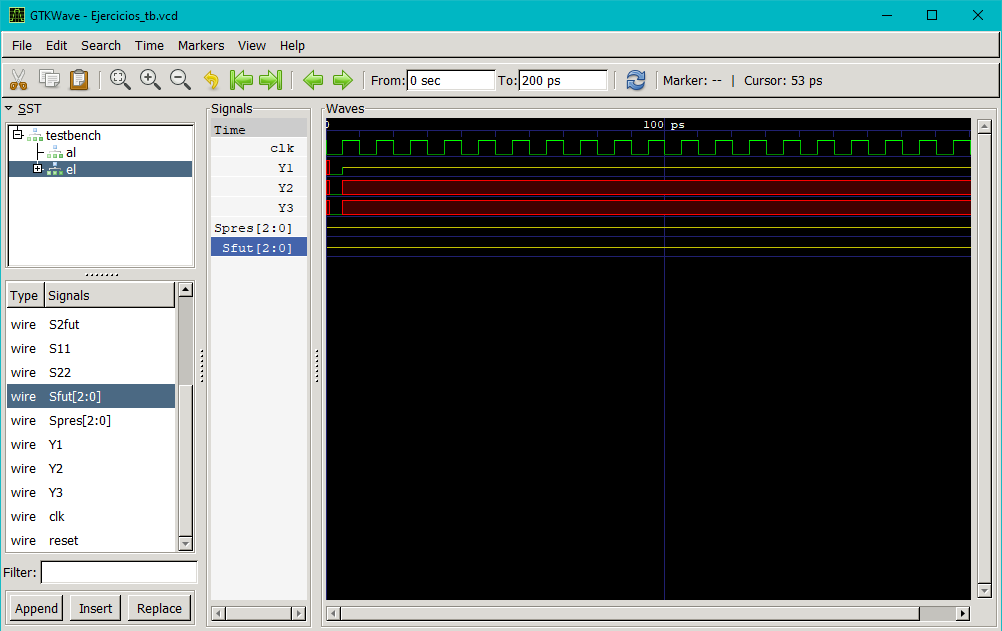
Implemente un Flip Flop tipo D de 4 bits con un *reset* asíncrono y un *set* síncrono en Verilog. Diseñe un *testbench* para probar su código.

## Ejercicio 06

Implemente los ejercicios 01 y 03 en Verilog. Recuerde que Verilog sólo es una descripción en código de su circuito. Diseñe un testbench para probar su máquina completa. Incluya ***screenshots*** de su diagrama de timing en su entrega en Canvas. Recuerde que debe subir su código a su repositorio en GitHub y agregar el link del repo en su entrega en Canvas.

[**https://github.com/dar17320/LabsDigital/tree/Final**](https://github.com/dar17320/LabsDigital/tree/Final)





ii