

IE-0523 Circuitos Digitales II

Tarea 2: Creación / Pruebas  
Diseño conductual: Multiplexor con memoria  
Mux 2:1 & Flip Flops

Timna Belinda Brown Ramírez  
B61254  
timna.brown@ucr.ac.cr

I-2020

---

## Tabla de contenidos

|  |           |
|--|-----------|
| <b>1. Enunciado</b>  | <b>4</b>  |
| <b>2. Indicaciones adicionales por medio de ZOOM</b>               | <b>6</b>  |
| <b>3. Restricción de páginas</b>                                   | <b>6</b>  |
| <b>4. Descripción arquitectónica o diagrama del circuito</b>       | <b>6</b>  |
| <b>5. Plan de Pruebas</b>  | <b>7</b>  |
| 5.1. Descripción . . . . .   | 7         |
| 5.2. Código - probador.v . . . . .                                 | 7         |
| <b>6. Instrucciones de utilización de la simulación (Makefile)</b> | <b>8</b>  |
| 6.1. Descripción . . . . .   | 9         |
| 6.2. Código - BancoPruebasConductual.v . . . . .                   | 9         |
| 6.3. Makefile . . . . .  | 10        |
| <b>7. Ejemplos de los resultados</b>                               | <b>10</b> |
| 7.1. Toma de datos 1 . . . . .                                     | 10        |
| 7.2. Toma de datos 2 . . . . .                                     | 11        |
| 7.3. Toma de datos 3 . . . . .                                     | 11        |
| <b>8. Análisis y conclusiones</b>                                  | <b>11</b> |
| 8.1. Análisis . . . . .  | 11        |
| 8.2. Conclusiones . . . . .  | 11        |



## Índice de figuras

|    |                                  |    |
|----|----------------------------------|----|
| 1. | Makefile ejecutado . . . . .     | 9  |
| 2. | Toma de datos número 1 . . . . . | 10 |
| 3. | Toma de datos número 2 . . . . . | 11 |
| 4. | Toma de datos número 3 . . . . . | 11 |

# 1. Enunciado

IE-0523 Circuitos Digitales II  
I Ciclo 2020  
Prof. Jorge Soto

18 de marzo de 2020

## Tarea #2 Multiplexor con memoria (Entrega 26 de marzo)

*Al igual que en la Tarea #1 tome el tiempo que demora en hacer cada una de las cosas solicitadas: búsqueda de información, diseño, elaboración de las pruebas, ejecución de las simulaciones, etc.*

### Evaluación

- |                               |     |
|-------------------------------|-----|
| 1. Funcionamiento del diseño: |     |
| a. Simulación                 | 40% |
| b. Pruebas                    | 40% |
| 2. Documentación              | 10% |
| 3. Makefile                   | 10% |

### Trabajo a realizar sobre el dispositivo a diseñar

Para esta tarea se deben completar los siguientes puntos:

1. Realice un diagrama del circuito propuesto.
2. Escriba una descripción **conductual** del multiplexor usando Verilog. Esta descripción servirá como una especificación detallada y formal del funcionamiento del dispositivo diseñado. Además servirá como patrón para hacer las pruebas sobre los diseños estructurales que se vayan a hacer. La descripción en Verilog deberá tener al menos un módulo de banco de pruebas, un módulo probador, y un módulo con la descripción conductual. Use Icarus Verilog para hacer esto.  
  
Recuerde que esta descripción conductual deberá ser sintetizada en el futuro. No se permite el uso de bloques *initial* u operadores “#” en la descripción conductual (en el probador sí se pueden usar). Tenga cuidado con los *latches*. No utilice *negedge clk* en ninguna parte de la tarea.
3. Defina un plan de pruebas para garantizar el funcionamiento del diseño. Desarrolle las pruebas y automatizárlas mediante el *Makefile*. Diseñe un archivo banco de pruebas y un archivo probador. Utilice como guía el probador del ejemplo Alarma, respetando las reglas de sincronizar con el reloj y utilizar asignaciones no bloqueantes.

### Especificaciones

Estructura del bloque conductual:

```
module mux (  
    input clk,  
    input reset_L,  
    input selector,  
    input [1:0] data_in0,  
    input [1:0] data_in1,  
    output reg [1:0] data_out  
);
```

El bloque consiste en un multiplexor de 2:1 con dos entradas y una salida de datos de 2 bits, además de un flip-flop a la salida para almacenar el resultado. Todo esto es parte de la descripción conductual, dentro del “*module mux*”.

Recuerde aplicar `reset_L` a los flip-flops para borrar su estado inicial no determinado. Cuando el bloque está en reset (`reset_L == 0`), los flops mostrarán valor de 0. Cuando no está en reset (`reset_L == 1`), mostrará el resultado del multiplexor.

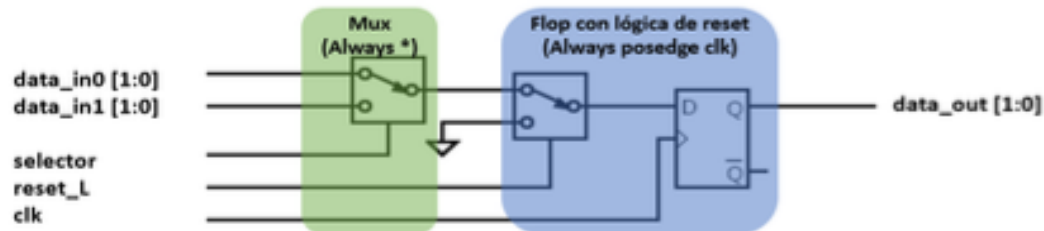


Figura 1. Esquemático del diseño.

### Consejos para el diseño

- Recuerde que esta tarea se trata de un diseño **conductual**, en el que se utilizan bloques *always* con lógica *if/else*, entre otros.
- Utilice como guía el ejemplo de Alarma (descripción, probador y testbench).
- Cuando se habla de que un dato debe pasar sin retrasos de un punto al otro, tiene que utilizar lógica combinacional. Utilice un `always @(*)` o un `assign` (el retardo combinacional de las compuertas se desprecia en esta tarea).
- Cuando debe almacenar o sostener un dato, tiene que utilizar lógica secuencial (flip-flops). Utilice un `always @(posedge clk)`.
- Recuerde el paralelismo en el diseño de hardware, distintas lógicas o distintos bloques `always` pueden ejecutarse en el mismo tiempo de simulación.
- Dos *drivers* o asignadores no pueden manipular la misma señal. Por ejemplo, dos bloques `always` no pueden tratar de asignar un valor a la misma señal. Esto provocaría un corto en el circuito.
- La lógica combinacional en un bloque `always @(*)` siempre debe tener un valor por defecto al inicio del bloque.

### Propuesta de Plan de Pruebas Mínimo

Construya un probador que genere como mínimo el estímulo siguiente sobre las entradas del circuito. Además, en la captura se observa cómo deben reaccionar las salidas ante los cambios en las entradas.

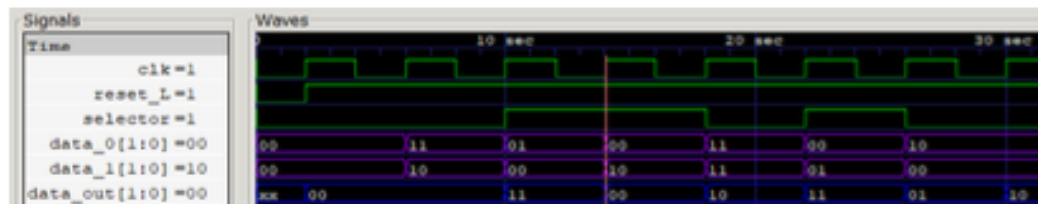


Figura 1. Captura de entradas y salidas esperadas del multiplexor.

## Guía para el reporte

Se debe entregar en forma electrónica un documento que incluya los siguientes puntos en a lo sumo 10 páginas de longitud:

1. **Distribución del tiempo invertido en la tarea.**
2. **Descripción arquitectónica o diagrama del circuito.**
3. **Plan de Pruebas:** Descripción de las pruebas realizadas.
4. **Instrucciones de utilización de la simulación:** Describir cómo correr las pruebas mediante el *makefile*.
5. **Ejemplos de los resultados:** Una descripción de los resultados más importantes acompañados de los diagramas temporales de la simulación (GTKWave) o cualquier otra salida que demuestre claramente el comportamiento descrito. No es necesario incluir una muestra exhaustiva de resultados, sino que los más representativos del diseño. El punto es mostrarle al lector los comportamientos más sobresalientes para formarle una idea clara del funcionamiento del diseño.
6. **Análisis y conclusiones:** Analice los resultados y enumere los inconvenientes durante la tarea.

## 2. Indicaciones adicionales por medio de ZOOM

El profesor indicó que intentáramos no gastar mucho tiempo en el reporte, dado que las tareas por el COVID-19 así como el curso fue modificado respecto a entregas lo que genera que los trabajos se acumulen así como la materia. Considerando esta indicación, los tiempos fueron sumados en su totalidad y la especificación de instalación considerando diferentes sistemas operativos en algunas secciones no se denota todos los comandos utilizados para dicha instalación.

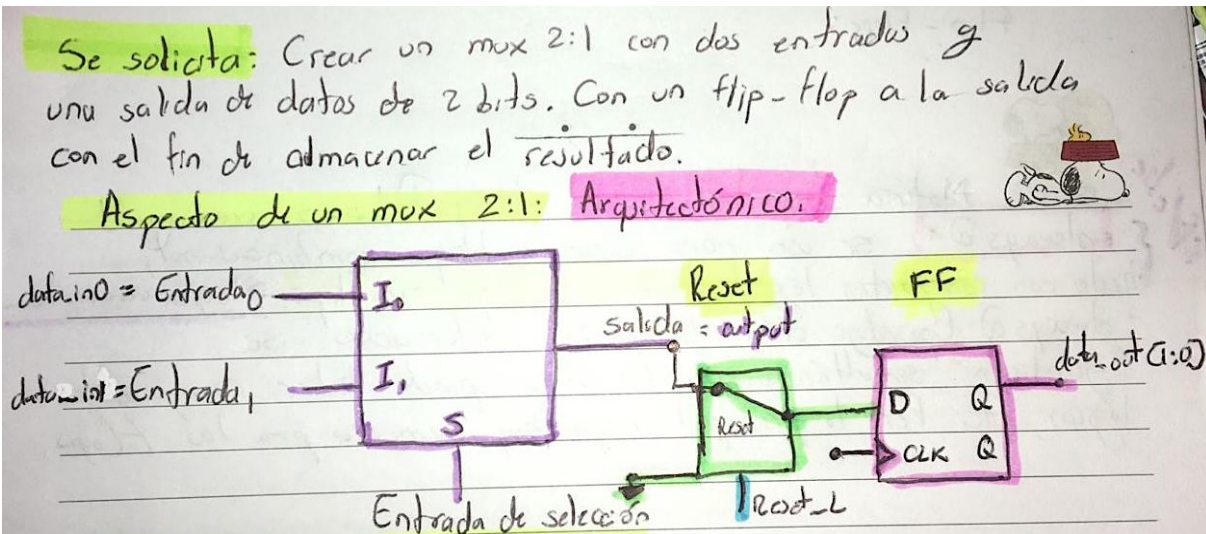
## 3. Restricción de páginas

Dado que en las instrucciones se menciona: "Se debe entregar en forma electrónica un documento que incluya los siguientes puntos en a lo sumo 10 páginas de longitud", se considera que las diez páginas se contabilizan a partir de la sección **Descripción arquitectónica o diagrama del circuito** por lo que se adjunta el enunciado y algunas indicaciones de por ZOOM con el fin de que el trabajo sea lo más claro posible.

## 4. Descripción arquitectónica o diagrama del circuito

- Tiempo total: aproximadamente 3 horas

Con el fin de comprender como se describe arquitectónicamente un diseño de HDL (Hardware Description Language) se buscó información pertinente en las presentaciones realizadas por el profesor, específicamente la presentación 1 y 2. Del mismo modo, se vieron diferentes tutoriales en YouTube, así como documentos en internet [1].



## 5. Plan de Pruebas

- Tiempo total: aproximadamente 19 horas

### 5.1. Descripción

Considerando las indicaciones, el diseño se enfoca en la sección conductual de un multiplexor 2:1 donde 2:1 significa, con dos entradas de datos y una entrada de selección. Dicho esto, consideramos que el funcionamiento de un multiplexor es permitir el paso de la entrada de acuerdo al estado en que se encuentra la entrada selectora. Por este motivo, el probador incluye 3 pruebas realizadas. Las cuales suceden mediante el cambio en el estímulo lo que permite poseer una idea el comportamiento de las señales. La página de XILINX fue utilizada con el fin de entender algunos errores obtenidos durante el desarrollo de la tarea [2].

### 5.2. Código - probador.v

```
// Se define el modulo tester
module probador(

input [1:0] data_out_probador ,
output reg [1:0] data_in0_probador ,
output reg [1:0] data_in1_probador ,
output reg reset_L_probador ,
output reg clk_probador ,
output reg selector_probador
);

initial
begin

// Definiendo el dumpfile NOMBRE.DEL.ARCHIVO.ESCOGENCIA.PERSONAL.vcd), o bien
// conocido por variable change dump, este archivo contiene
// informacion sobre simulador utilizado, escala de tiempo, fecha de creacion,
// definiciones de variables, y cambios de valor.
$dumpfile("mux2_1_con_Flip_flop.vcd");
```

```
// Mensaje que se imprime en consola una vez
$display ("\\t\\t\\tclk_probador,\\treset_L_probador,
\\tselector_probador,\\tdata_in0_probador, \\tdata_in1_probador,\\tdata_out_probador");
// Mensaje que se imprime en consola cada vez que un elemento de la lista cambia
$monitor($time,"\\t%\\t%\\t\\t%\\t\\t%\\t\\t%\\t\\t%", clk_probador,
reset_L_probador, selector_probador, data_in0_probador, data_in1_probador,
data_out_probador);
$dumpvars;

// Esto pasa el primer ciclo de reloj... Definiendo valores iniciales ....
data_in0_probador = 2'b0;
data_in1_probador = 2'b0;
selector_probador = 0;
reset_L_probador = 0;

// Se realiza la siguiente prueba #1

repeat(2) begin
    @(posedge clk_probador);
// Una vez que pasa el primer ciclo de reloj las senales
// Con el fin de que las senales pasen por todos sus valores posibles
{data_in0_probador, data_in1_probador} <= {data_in0_probador,
data_in1_probador} + 2'b1;
{reset_L_probador, selector_probador} <= {reset_L_probador, selector_probador} + 1;

// Se realiza la siguiente prueba #2
    @(posedge clk_probador);
// Una vez que pasa el primer ciclo de reloj las senales
// Con el fin de que las senales pasen por todos sus valores posibles
{data_in0_probador, data_in1_probador} <= {data_in0_probador,
data_in1_probador} + 2'b1;
{reset_L_probador, selector_probador} <= {reset_L_probador, selector_probador} + 1;

// Se realiza la siguiente prueba #3
    @(posedge clk_probador);
// Una vez que pasa el primer ciclo de reloj las senales
// Con el fin de que las senales pasen por todos sus valores posibles
{data_in0_probador, data_in1_probador} <= {data_in0_probador,
data_in1_probador} + 2'b1;
{reset_L_probador, selector_probador} <= {reset_L_probador, selector_probador} + 1;
end
// Termina de almacenar senales
$finish;
end // end del bloque initial
// Para el reloj
initial clk_probador <= 0;
// Hace "toggle" cada 2*10ns
always #2 clk_probador <= ~clk_probador;
endmodule
```

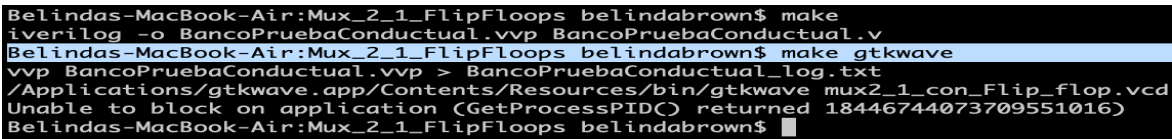
- Tiempo total: aproximadamente 5 minutos.



## 6.1. Descripción

Con el fin de implementarlo, ejecute lo siguiente mediante la consola o terminal de su computadora posicionándose en en la dirección absoluta del archivo popularmente conocido como PATH.

1. Simule el banco de pruebas conductual, dado que este contiene el módulo del multiplexor con memoria en donde se aplica el reset e incluye el módulo de probador en donde se encuentran las pruebas realizadas, esto se sucede mediante la indicación: `$make`
2. Dadas las instrucciones, requerimos visualizar el comportamiento de las señales dadas las pruebas realizadas. Esto se realiza mediante la instrucción: `$make gtwave`.



```
Belindas-MacBook-Air:Mux_2_1_FlipFloops belindabrown$ make
iverilog -o BancoPruebaConductual.vvp BancoPruebaConductual.v
Belindas-MacBook-Air:Mux_2_1_FlipFloops belindabrown$ make gtwave
vvp BancoPruebaConductual.vvp > BancoPruebaConductual_log.txt
/Applications/gtkwave.app/Contents/Resources/bin/gtkwave mux2_1_con_Flip_flop.vcd
Unable to block on application (GetProcessPID()) returned 18446744073709551016)
Belindas-MacBook-Air:Mux_2_1_FlipFloops belindabrown$
```

Figura 1: Makefile ejecutado

## 6.2. Código - BancoPruebasConductual.v

```
// escala          unidad temporal (valor de uno) / precision
'timescale        1ns          / 100ps
// incluye de archivos de verilog
// Pueden omitirse y llamarse desde el testbench
#include "mux2_1_conductual.v"
#include "probador.v"

module BancoPruebas; // Testbench

wire [1:0] data_out_banco_conduc , data_in0_banco_conduc , data_in1_banco_conduc ;
wire reset_L_banco_conduc , clk_banco_conduc , selector_banco_conduc ;

// Descripcion conductual de alarma
mux2_1 a_cond( .data_out          (data_out_banco_conduc),
               .data_in0          (data_in0_banco_conduc),
               .data_in1          (data_in1_banco_conduc),
               .reset_L           (reset_L_banco_conduc),
               .clk               (clk_banco_conduc),
               .selector          (selector_banco_conduc)
);

// Probador: generador de senales y monitor
probador prob(.data_out_probador          (data_out_banco_conduc),
              .data_in0_probador           (data_in0_banco_conduc),
              .data_in1_probador           (data_in1_banco_conduc),
              .reset_L_probador            (reset_L_banco_conduc),
              .clk_probador                (clk_banco_conduc),
              .selector_probador           (selector_banco_conduc)
);

endmodule
```

### 6.3. Makefile

```
#Verilog

TESTBENCH = BancoPruebaConductual
SRCS      = $(TESTBENCH).v
VCD = mux2_1_con_Flip_flop

.PHONY: all
all: $(TESTBENCH).vvp

%.vvp: %.v
    iverilog -o $@ $^
#simulate the dump

#To generate these two files I need this
$(VCD).vcd $(TESTBENCH)_log.txt: $(TESTBENCH).vvp
# this is how it does it
    vvp $^ > $(TESTBENCH)_log.txt

#target phony
.PHONY: gtkwave
gtkwave: $(VCD).vcd
    /Applications/gtkwave.app/Contents/Resources/bin/gtkwave $^

.PHONY: clean
clean:
    rm -rf $(TESTBENCH).vvp $(VCD).vcd $(TESTBENCH)_log.txt *.out *.syn.v
```

## 7. Ejemplos de los resultados

Dado que con ver el comportamiento de las señales, lo que se desea analizar es ... se toman las siguientes imágenes visualizadas mediante gtkwave. Estas serán comentadas en la sección de **Análisis y conclusiones**

### 7.1. Toma de datos 1

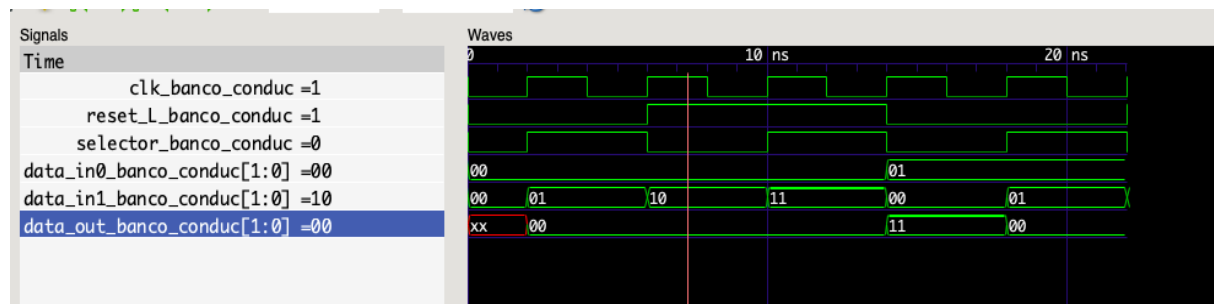


Figura 2: Toma de datos número 1

## 7.2. Toma de datos 2

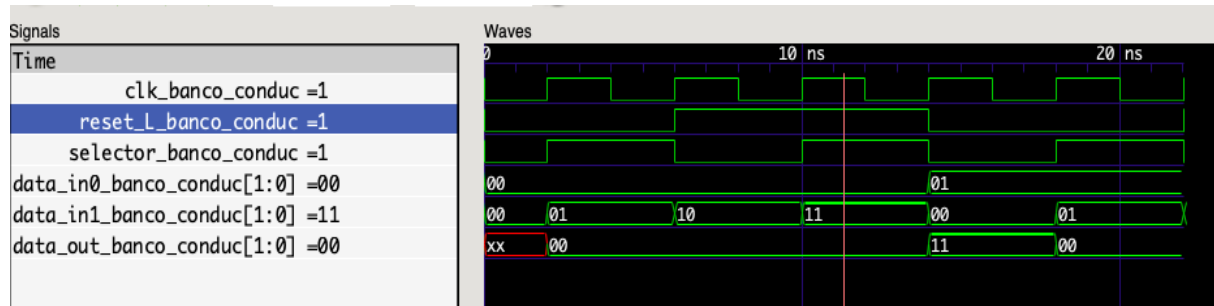


Figura 3: Toma de datos número 2

## 7.3. Toma de datos 3

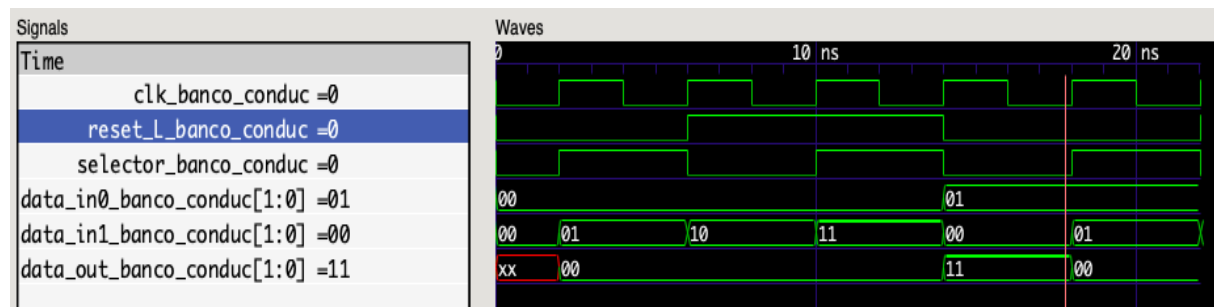


Figura 4: Toma de datos número 3

## 8. Análisis y conclusiones

- Tiempo total: aproximadamente 4 horas

### 8.1. Análisis

Cuando se realizan pruebas para el diseño del multiplexor 2:1 con memoria, lo que se espera es funcione siguiendo la siguiente lógica: si el selector posee un valor de cero, la salida debe contener lo que presenta la entrada cero. Por otro lado, si el selector posee un valor de uno, la salida debe mostrar el valor de la entrada uno.

### 8.2. Conclusiones

Considerando el análisis realizado se puede concluir los siguientes puntos:

- Como la información del multiplexor pasa por un flop el data\_out responde al selector y la señal de entrada del ciclo anterior.
- La función del reset es fijar la salida del flop en cero. Su función de reseteo ideal es al principio.

## 9. Confección del reporte

- Tiempo total: aproximadamente 3 horas

Fue realizado en L<sup>A</sup>T<sub>E</sub>X mediante el interpretador en línea Overleaf. Contiene imágenes recortadas de lo solicitado por el profesor.

## Referencias

- [1] Universidad de Valladolid. *5-Descripción basadas en algoritmos*. <https://forums.xilinx.com/t5/Simulation-and-Verification/Verilog-syntax-seems-correct-but-I-still-get-an-error/td-p/491096>, Accesado el 22 de abril del 2020.
- [2] XLINX. *Verilog syntax seems correct but I still get an error*. <https://forums.xilinx.com/t5/Simulation-and-Verification/Verilog-syntax-seems-correct-but-I-still-get-an-error/td-p/491096>, Accesado el 20 de abril del 2020.