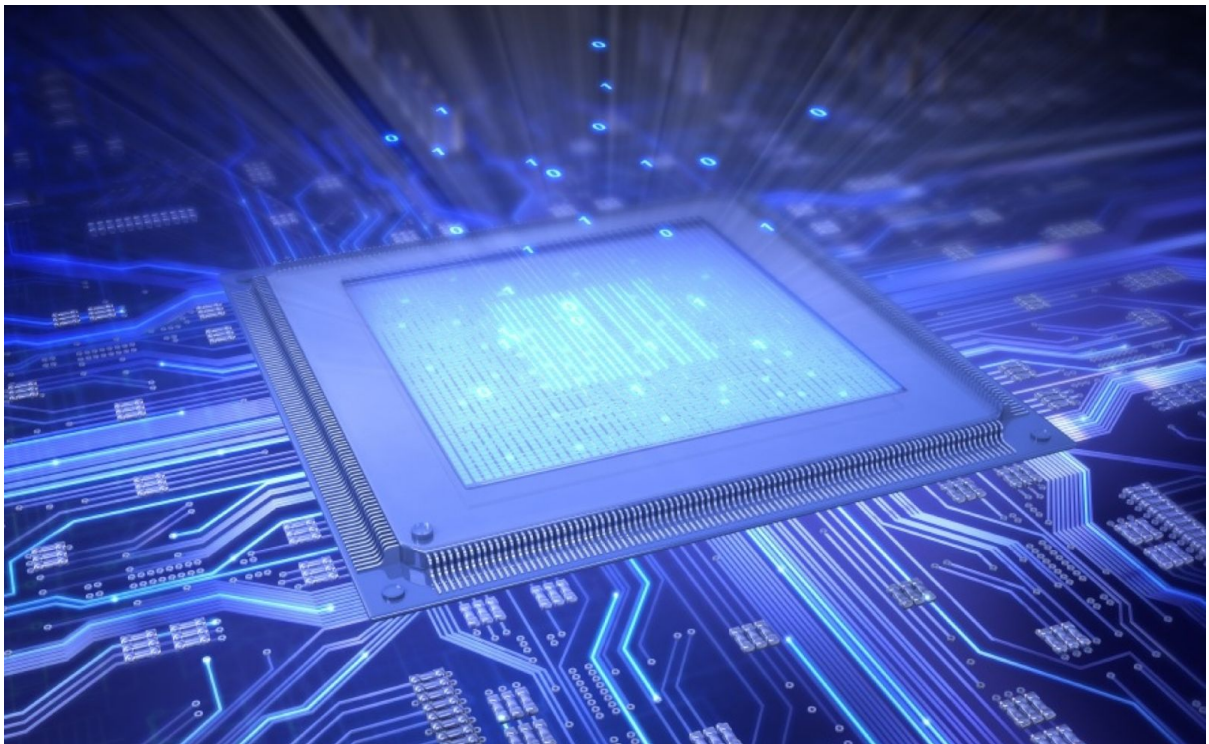


Modelado y Síntesis de Sistemas Electrónico Digitales

Apartado 1. Diseño del controlador del puerto paralelo (EPP)



Realizada por:

Guillermo Alba Sánchez

Adrián García-Vera de Lope

Índice

Modelado y Síntesis de Sistemas Electrónico Digitales	1
Apartado 1. Diseño del controlador del puerto paralelo (EPP)	1
Índice	2
Introducción	3
1.1. Comprobación de la funcionalidad del controlador del puerto paralelo	4
Modelar la entidad epp_controler	4
Simulación funcional de la entidad epp_controler	9
Simulación temporal de la entidad epp_controler	9
Informe sobre los recursos utilizados	11

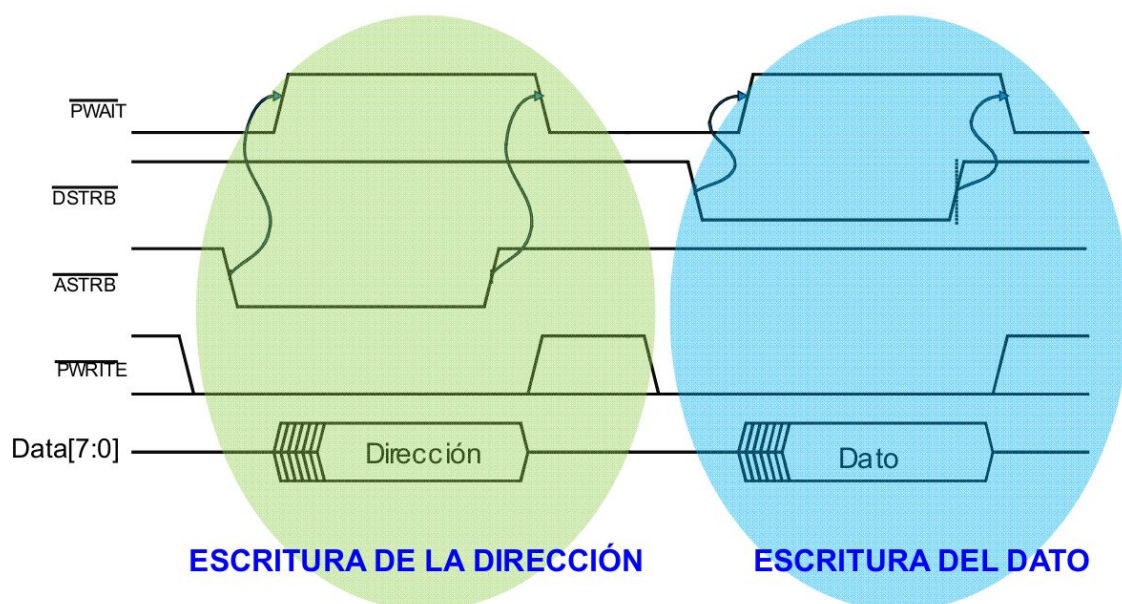
Introducción

Para este apartado se nos proporciona en clase el diseño completo para modelar la entidad *epp_controler*.

Esta entidad es la encargada de actuar como interface entre el puerto USB del ordenador y el diseño, es decir, recibe los datos enviados desde el PC con un protocolo EPP y proporciona dos salidas **DIR** (dirección) y **DATO** (dato), ambas de 8 bits y mostradas en codificación hexadecimal. Según estas salidas y la siguiente tabla se realiza la función deseada.

Dirección	Dato	Función
11x	11x	Inicialización del códec LM4550 (RESTART)
F0x	00x-FFx	Frecuencias de la nota
B0x	00x-1Fx	Volumen de la señal de audio
CAx	DDx	Tono por el canal derecho
CAx	11x	Tono por el canal izquierdo
CAx	22x	Tono por ambos canales
CAx	Resto combinaciones	Tono por ningún canal

La escritura de estos datos siguen la estructura de la siguiente imagen (representa el ciclo de escritura del protocolo EPP).



1.1. Comprobación de la funcionalidad del controlador del puerto paralelo

Para la codificación del modelo en VHDL también se nos ha proporcionado diversos archivos como es la declaración de la entidad en el archivo *epp_controler.vhd*.

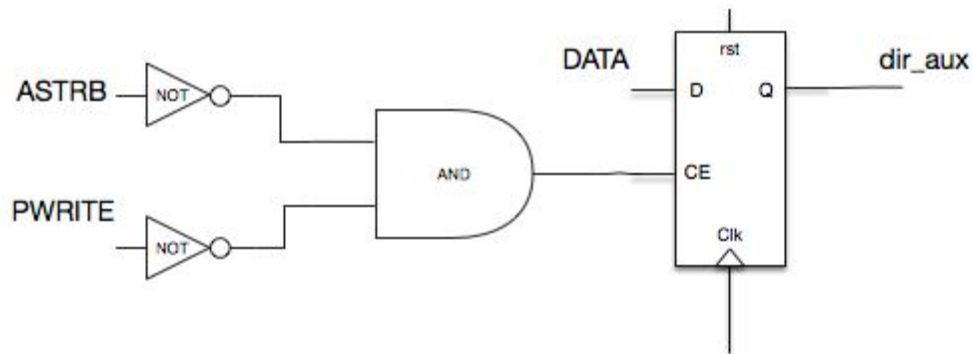
Modelar la entidad epp_controler

Para modelar esta entidad hemos seguido los diseños propuestos por nuestro profesor.

Es muy importante entender cómo funciona este componente para poder modelarlo correctamente.

Mediante el *epp_device.vhd* vamos a ir leyendo el archivo *datos.dat*. Este archivo consta de dos columnas de 8 bits cada una en hexadecimal, la primera corresponde con la dirección y la segunda con el dato, pero dichos datos se transmiten por una señal de 8 bits, por tanto no podemos enviar ambos datos al mismo tiempo, para poder solucionar este inconveniente guardaremos la dirección en una señal auxiliar y en el siguiente tick de reloj (cuando estemos leyendo la dirección) enviaremos los datos en base a un CE que crearemos y que nos permitirá enviar ambos datos a la vez.

1. Escritura de la dirección: la dirección es dependiente de **ASTRB** (está teniendo lugar una transferencia de direcciones) y de **PWRITE** (está teniendo lugar un ciclo de escritura), con estas dos señales crearemos un CE, si seguimos la documentación que se nos proporciona estas estarían en '0' para indicar que *se está escribiendo dirección*, las negamos para poder activar la salida de la puerta lógica AND y así activar el CE, lo cual nos permite obtener la señal **dir_aux**.

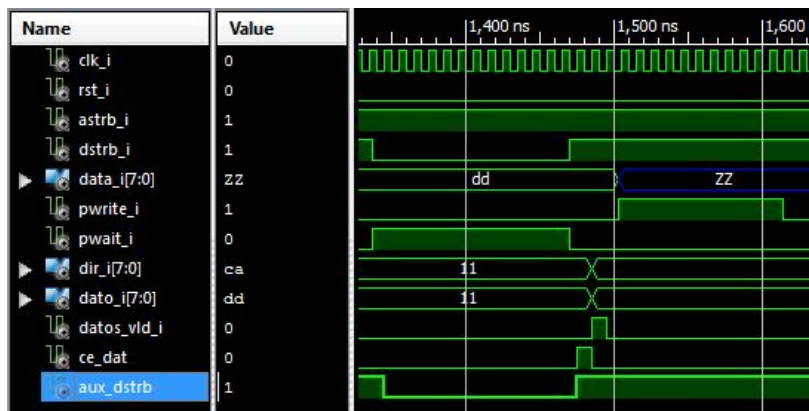
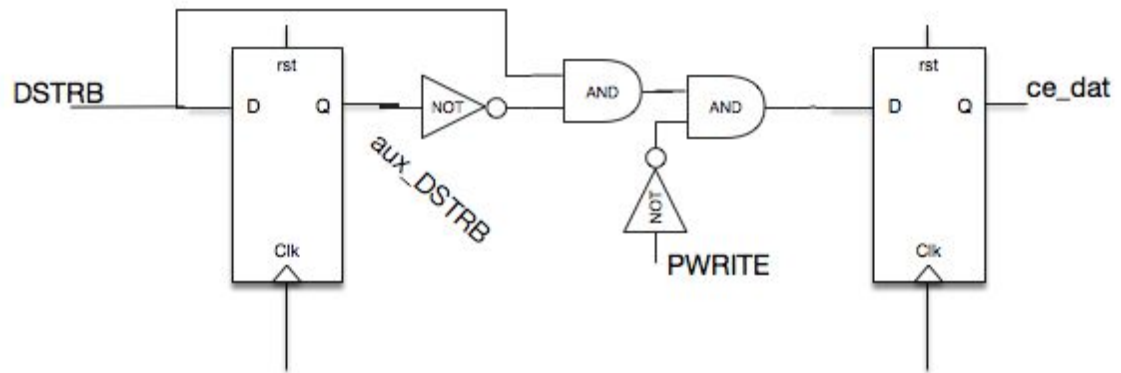


El modelo en VHDL queda así:

```
direccion : process(RST, CLK)
begin
    if RST='1' then
        dir_aux <= (others => '0');
    elsif CLK'event and CLK='1' then
        if (not ASTRB and not PWRITE) = '1' then -- CE = '1'
            dir_aux <= DATA;
        end if ;
    end if ;
end process ; -- direccion
```

El formato usado ha sido el de un biestable tipo D asíncrono, activo en flanco de subida y con CE (la condición del CE la marca la puerta lógica AND).

2. Detector de flanco de subida: en este caso el primer biestable actúa como una detección de cambio de flanco en base al actual y anterior ticks de reloj sobre la señal **DSTRB**, si además se está escribiendo (PWRITE vale '0'), habremos generado nuestro **ce_dat** que usaremos posteriormente para el envío simultáneo de las señales **DIR**, **DATO** y **DATOS_VLD**.



Aquí podemos ver las distintas señales que componen este proceso para generar el CE.

El modelo en VHDL queda así:

```

datos : process(RST, CLK)
begin
  if RST='1' then
    aux_DSTRB <= '0';
  elsif CLK'event and CLK='1' then
    aux_DSTRB <= DSTRB;
  end if ;

  if RST='1' then
    ce_dat <= '0';
  elsif CLK'event and CLK='1' then
    ce_dat <= DSTRB and not aux_DSTRB and not PWRITE;
  end if ;
end process ; -- datos

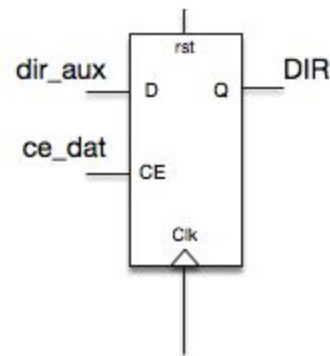
```


3. Salida de la dirección: este proceso coge la señal auxiliar **dir_aux** de 8 bits (que contiene el valor de la dirección) y lo transmite a la señal de salida **DIR** (también de 8 bits).

Para activar la transmisión de datos la señal **ce_dat** que generamos anteriormente debe estar a '1'.

El modelo en VHDL queda así:

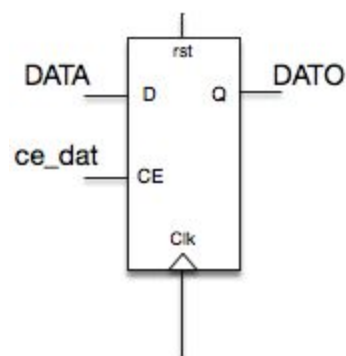
```
s_DIR : process(RST, CLK)
begin
  if RST='1' then
    DIR <= (others => '0');
  elsif CLK'event and CLK='1' then
    if ce_dat = '1' then
      DIR <= dir_aux;
    end if ;
  end if ;
end process ; -- s_DIR
```



4. Salida del dato: este proceso coge la señal de entrada **DATA** (que contiene el valor de la dirección o del dato) y lo transmite a la señal de salida **DATO**. Para escribir sólo el valor del dato usamos como CE el **ce_dat**, que sólo está activo cuando se está transmitiendo un valor de dato en la señal **DATA**.

El modelo en VHDL queda así:

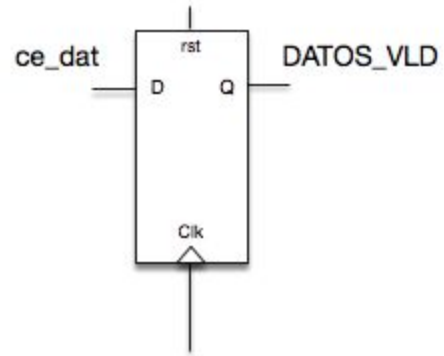
```
s_DATO : process(RST, CLK)
begin
  if RST='1' then
    DATO <= (others => '0');
  elsif CLK'event and CLK='1' then
    if ce_dat = '1' then
      DATO <= DATA;
    end if ;
  end if ;
end process ; -- s_DATO
```



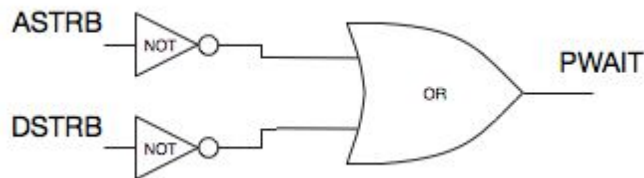
5. Comprobación de valores correctos: sirve de sincronización para las salidas anteriores indicando que hay nuevos valores en ellas.

El modelo en VHDL queda así:

```
s_DATOS_VLD : process(RST, CLK)
begin
  if RST='1' then
    DATOS_VLD <= '0';
  elsif CLK'event and CLK='1' then
    DATOS_VLD <= ce_dat;
  end if ;
end process ; -- s_DATOS_VLD
```



6. PWAIT: Este proceso activa la señal de salida **PWAIT** que indica que hay una transferencia en curso de direcciones o datos.



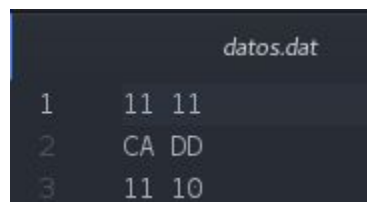
El modelo en VHDL queda así:

```
PWAIT <= not ASTRB or not DSTRB;
```


Simulación funcional de la entidad *epp_controler*

En la simulación funcional de la entidad estamos comprobando que todo nuestro modelo tiene la funcionalidad deseada.

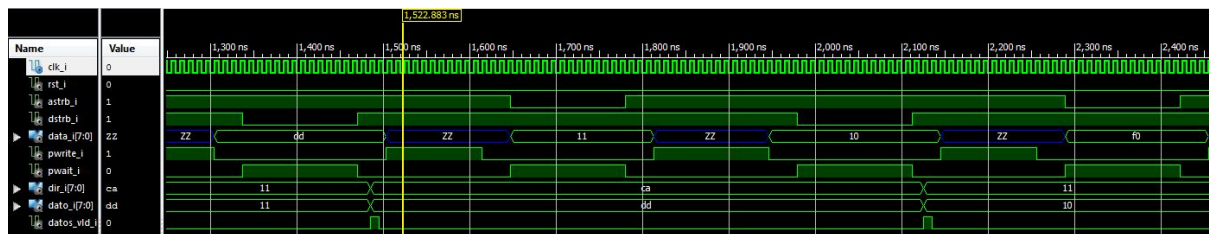
Comprobamos la salidas de las señales **DIR** y **DATO** fijándonos que los datos que está transmitiendo la señal **DATO** corresponden a las columnas del fichero datos.dat.



datos.dat		
1	11	11
2	CA	DD
3	11	10

Captura al archivo datos.dat

La señal **DATO** la hemos rellenado con los valores del archivo *datos.dat* (1º columna = valores direccion, 2º columna = valores dato), y como comprobamos en la captura de la simulación, los valores retransmitidos por la señal **DATO** son los mismos, y los valores recogidos por las señales **DIR** y **DATO** (anteriormente mencionadas) están guardando sus valores de dirección y dato respectivamente.

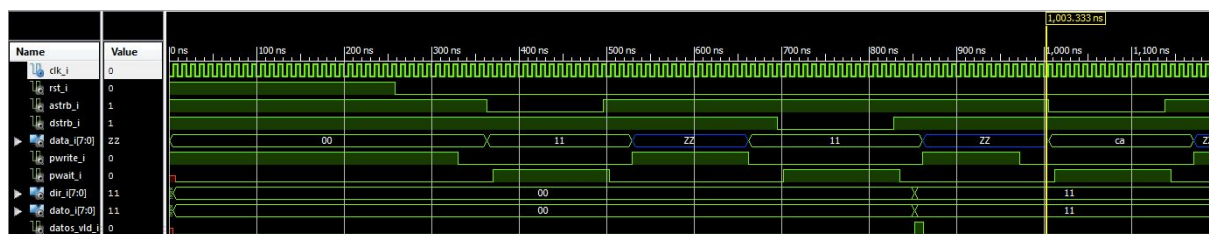


Captura de la simulación funcional de la entidad epp_controler

Simulación temporal de la entidad *epp_controler*

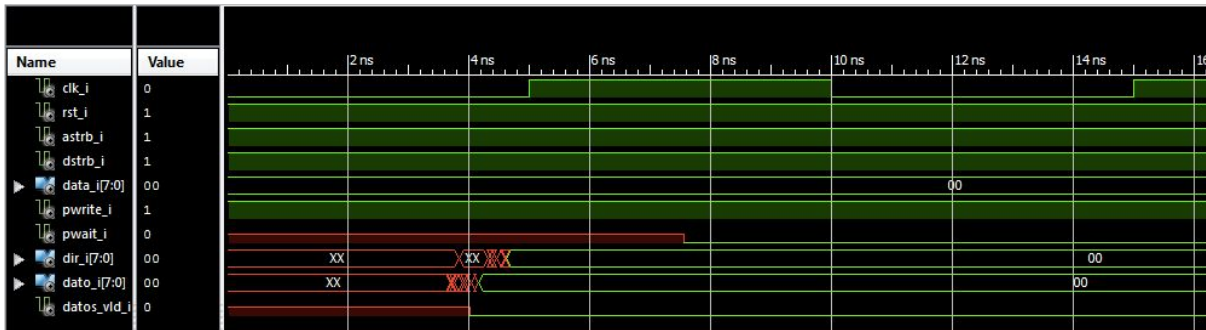
Con la simulación temporal comprobamos que la temporización del diseño cumple con las restricciones impuestas al mismo, fundamentalmente las relacionadas con la señal del reloj.

Comprobando la salida de las mismas señales que en la simulación funcional vemos que recoge perfectamente los valores de la dirección y el dato como debería de hacerlo.



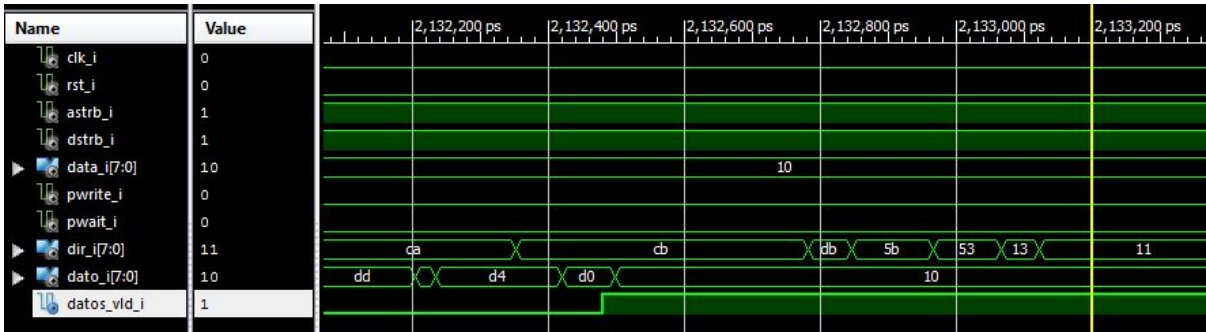
Captura de la simulación temporal

Como se puede observar en la imagen siguiente, hay un retardo causado por la simulación temporal y muy similar al que puede producir la descarga en placa.



Captura de la simulación temporal al inicio

Si hacemos zoom sobre los momentos en los que cambian los datos que se envían **DIR** y **DATO** podemos apreciar cómo se están enviando datos que parecen “basura” ya que no podemos encontrarlos en el fichero de datos, además se puede apreciar un pequeño desfase entre **DIR**, **DAT** y **DATOS_VLD**.



Captura de la simulación temporal al cambio de valores de los datos

En las siguientes imágenes vemos la diferencia del desfase de las señales en la simulación temporal (*fig 2.*) respecto a la simulación funcional (*fig 1.*), ya que en la simulación funcional todo funciona de forma ideal, y en la realidad (simulación temporal) no es así.



fig 1.

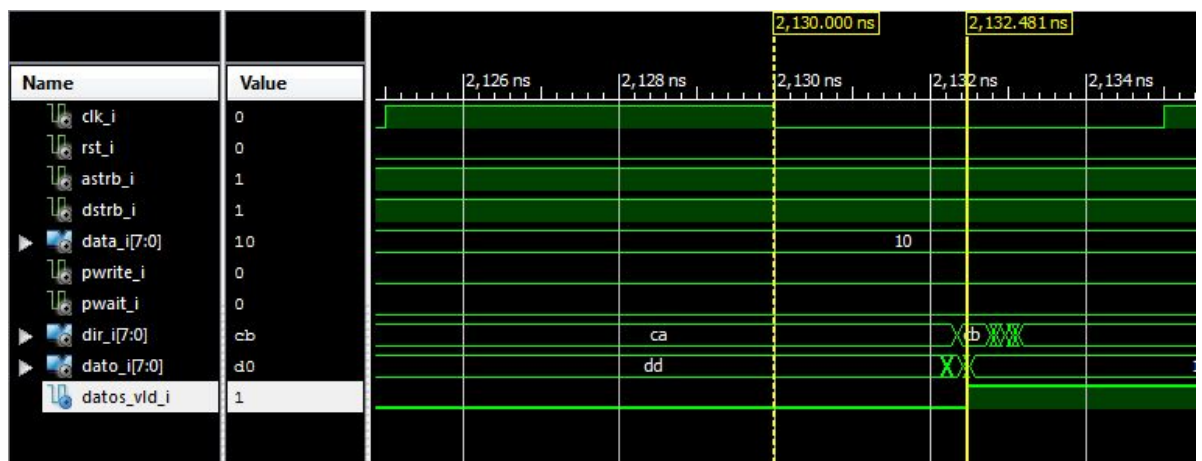


fig 2.

Aquí podemos ver la diferencia en el error que comentamos anteriormente entre la simulación funcional (fig 1.) y la temporal (fig 2.).

Informe sobre los recursos utilizados

Es condición imprescindible generar un informe sobre los recursos utilizados cuando se haga uso de la simulación temporal y posterior descarga en placa ya que de esta forma sabremos la cantidad de hardware en uso.

Como podemos observar en el informe de nuestros resultados estaremos usando muy pocos recursos de la placa, lo cual es perfecto si queremos escalar nuestro sistema.

Number of Slices Registers	43 out of 54,576 1%
Number of Slice LUTs	6 out of 27,288 1%
Number of occupied Slices	14 out of 6,8221%

