IE0523 - Circuitos Digitales II: Proyecto #1: Capa Física PCIE

WILLY VILLALOBOS, B17170
SEBASTIÁN BLANCO ,B41005
JOSÉ ANDRÉS MATAMOROS, B17170
Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
4 de junio de 2018

***Resumen*—El este documento se describe el proceso realizado para diseñar la capa PHY del protocolo de PCIE simplificada, donde se tienen datos de un byte que se envían a un receptor que debe leer esos mismos datos con la misma tasa de transferencia de datos. En la sección de descripción arquitectónica se tiene el diseño realizado tanto para la capa Tx como la Rx del diseño con sus entradas y salidas. En la parte del plan de pruebas se tiene las pruebas realizadas al diseño tanto a su descripción estructural como su descripción conductual.**

I. TIEMPO EMPLEADO

- Investigación inicial del problema: 8 horas
- Implementación de la capa física de PCIe: 36 horas
- Pruebas y ensayos: 20 horas
- Redacción de informe y presentación: 4 horas
- Tiempo total invertido: **68:00:00**

II. ESTUDIO DE MERCADO

Cuadro I
TABLA DE VERDAD DEL REGISTRO

XIO1100 Texas Instrument	PX1011B NXP
CLK referencia 100 MHz - 125MHz.	Single Lane 2.5 Gbits/s.
Bajo consumo de potencia.	Frecuencia máxima 250 MHz.
PCI Express 1.1 Compliant.	Bajo consumo de potencia.
Un costo de $9.7 la unidad.	Costo por unidad de $7.96.

III. DESCRIPCIÓN ARQUITECTÓNICA

La capa de PHY de el protocolo PCIE contiene dos grandes módulos, La etapa Tx y la etapa Rx:

- Etapa Tx: La etapa Tx es la encargada de transmitir los comandos y datos ala etapa Rx mediante un puerto serial. En esta estapa se tienen tres grandes módulos:
  - Multiplexor de Comandos: el multiplexor de comandos esta encargado de escoger cual comando se pasa a la siguiente etapa mediante las lineas de selección, este puede propagar 10 comandos diferentes de un byte cada uno:
    - TLP = FF
    - COM = BC
    - PAD = C7
    - SKP = AC
    - STP = AA
    - SDP = E5
    - END = F6
    - EDB = DF
    - FTS = A8
    - IDL = AE

    La última entrada del multiplexor es la entrada del byte de datos de que se quieren enviar. El multiplexor cuenta con 4 lineas de selección que son controladas por el módulo de contador que hace que estas lineas cambien con una frecuencia de 1 MHz para poder hacer la distribución de datos con esta frecuencia. Las salidas de este módulo son:
    - Outmux: Salida que propaga el dato seleccionado según el valor del bus de selección del multiplexor.
- Byte Stripping: Los datos de vienen de la etapa anterior con la frencuencia de 1 Mhz se reparten con la misma frecuencia. Estos datos se reparten en cuatro canales de un byte cada uno donde los valores de los 4 canales se actualizan con una frecuencia de 250 kHz. Este tiene cuatro salidas que los los cuatro bytes que entrarian a la estapa Paralel-Serie.
- Etapa Paralelo-Serie (p2s): En esta etapa se toman los cuatro bytes de os cuatro canales y se separann en sus ocho bits, donde estos se mandan por una unica salida de un bit serial con una frecuencia de 2 MHz, cada canal tiene su propia linea serial para un bus serial de 4 bits de salida.
- Etapa Serie-Paralelo: Esta etapa tiene como entrada el bus de 4 bits de entrada, donde cada bit se va guardando en cuatro diferentes registros, donde después de que se guardan ocho de los bits que vienen de los canales seriales en el registro este pone el valor de los ocho bits como salida para la siguiente etapa, esto pasa para los cuatro registros. El valor delos 4 bytes de salida se refrescan con una frecuencia de 250 kHz
- Byte Joining: de los 4 bytes de los cuatro canales de entrada que vienen con una frecuencia de 250 kHz, se van acomodando 1 de cada canal en la salida con una frecuecia de 1 Mhz, este proceso es controlado por un mutiplexor con 2 lineas de selecion, las cuales son las que se aumentan su valor con esta frecuencia.

En la figura 1 se muestra el diagrama de bloques del registro implementado, con sus respectivas entradas y salidas de datos y modalidades. Adicionalmente, el cuadro I muestra el funcionamiento para las entradas y salidas.

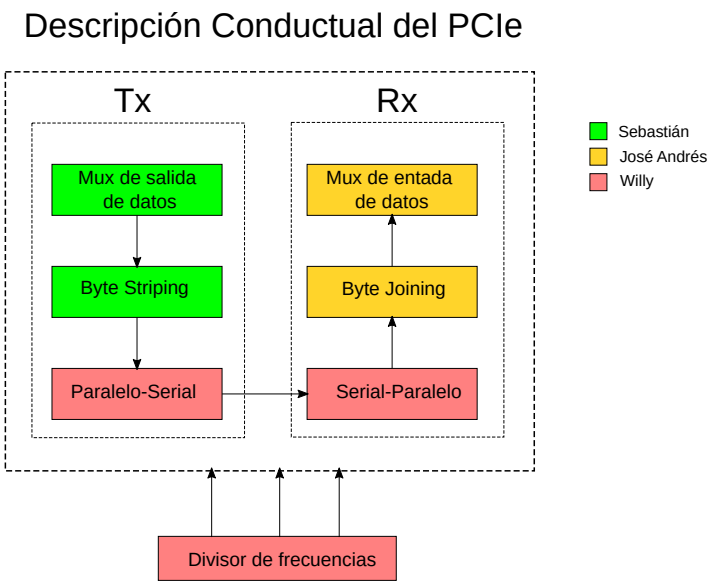

Figura 1. Diagrama de bloques conductual PHY PCIe.

IV. Plan de pruebas

En este caso se sigue el plan de pruebas sugerido en el enunciado, el cual se resume a lo siguiente:

- Rotación a la izquierda.
- Rotación a la derecha.
- Rotación circular a la izquierda.
- Rotación circular a la derecha.
- Carga en paralelo.

Para lograr estas pruebas, se implementa el testbench de la figura 2 para interconectar los distintos módulos.

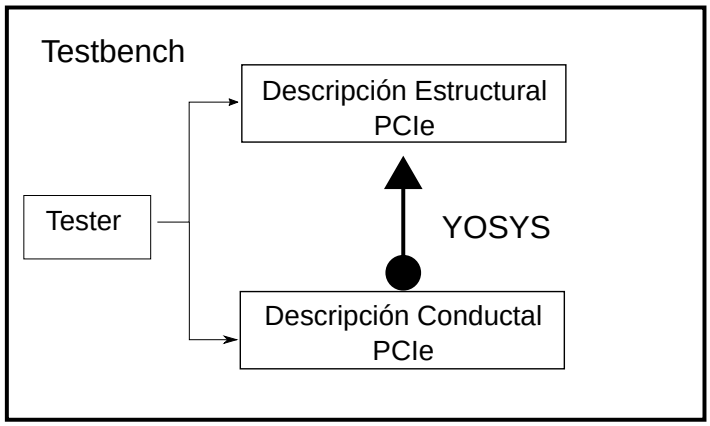

Figura 2. Diagrama de conexión de bloques del testbench para interconectar los diferentes módulos y probadores.

V. Instrucciones de utilización de la simulación

Se cita el README file el cual contiene instrucciones para la correcta utilización del Makefile con la compilación, síntesis y pruebas del registro [1]. Ahora el Makefile permite la compilación, síntesis y visualización de los datos cuando se usa el comando make, pero se siguen conservando los pasos intermedios para depuración. De igual forma el mismo Makefile permite la modificaicón de los archivos sintetizados para que incluyan el módulo de medición de potencia, así como la adición de los retardos y los flops de la versión sintetizada con cmos_cells. Se realizan 3 procesos de síntesis de manera simultánea, contemplando cada escenario planteado en el enunciado.

README

```
En este directorio se encuentra el código
    verilog para sintetizar
y probar un registro desplazante de 4 bits.
    Para poder verificar
su funcionamiento, basta con utilizar el
    Makefile proporcionado.

Las opciones disponibles son las siguientes:

make/make all compila, ejecuta, sintetiza y
    visualiza el sistema.

make compile  compila y genera un ejecutable
    usando los .v en
    el archivo de texto files.txt

make run  ejecuta el binario generado por make,
     así como
    el .vcd para su posterior visualización con
    gtkwave. Genera un archivo results.txt con
    los mensajes de ejecución

make wave abre el .vcd para su visualización.

make yosys  sintetiza el .v usando la receta
    provista por
    el archivo yoscript.txt

make yosysA sintetiza el .v usando la receta
    provista por
    el archivo yoscriptA.txt

make clean  limpia el directorio de ejecutables
     y otros
    residuos.
```

Makefile

```
SRC=files.txt
EXE=pcie
YOS=pcie_synth.yos
YOSD=pcie_synth_delay.yos

all: synth synth_delay compile run wave

compile:
  iverilog -o ${EXE}.exe -c ${SRC} -Ttyp

typical: compile

max:
  iverilog -o ${EXE}.exe -c ${SRC} -Tmax

min:
  iverilog -o ${EXE}.exe -c ${SRC} -Tmin

run:
  vvp ${EXE}.exe > results.txt

synth:
  yosys -s ${YOS}
  sed -i '1s;^;`include "cmos_cells.v"\n;'
    pcie_synth.v
```

```
  sed -i 's/_cond/_synth/' pcie_synth.v

synth_delay:
  yosys -s ${YOSD}
  sed -i '1s;^;`include "cmos_cells_delay.v"\n
    ;' pcie_synth_delay.v
  sed -i 's/_cond/_synth_delay/'
    pcie_synth_delay.v
  sed -i 's#BUF#BUF_delay#' pcie_synth_delay.v
  sed -i 's#NOT#NOT_delay#' pcie_synth_delay.v
  sed -i 's#NAND#NAND_delay#' pcie_synth_delay.
    v
  sed -i 's#NOR#NOR_delay#' pcie_synth_delay.v
  sed -i 's#DFF#DFF_delay#' pcie_synth_delay.v
  sed -i 's#DFFSR#DFFSR_delay#'
    pcie_synth_delay.v
  sed -i 's/<=/<= #15 /' pcie_synth_delay.v

wave:
  gtkwave ${EXE}.vcd ${EXE}.sav &

clean:
  rm -rf ${EXE}.exe ${EXE}.vcd *.ps *.eps *.dot
```

Para la visualización en gtkwave, se incluye un archivo.sav para facilitar la revisión de los datos, pues estos se despliegan sin necesidad de manipular la interfaz gráfica.

pcie.sav

```
[*]
[*] GTKWave Analyzer v3.3.89 (w)1999-2018 BSI
[*] Sat Jun  2 03:23:18 2018
[*]
[dumpfile] "/home/wivill/git/ie0523/proyecto
    -01/pcie.vcd"
[dumpfile_mtime] "Sat␣Jun␣␣2␣03:21:16␣2018"
[dumpfile_size] 165309
[savefile] "/home/wivill/git/ie0523/proyecto
    -01/pcie.sav"
[timestart] 0
[size] 1366 697
[pos] -1 -1
*-22.135500 8820000 1250000 1000000 9750000
    5750000 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
    -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[treeopen] testbench.
[treeopen] testbench.cond.
[treeopen] testbench.cond.bj_rx_cond.
[treeopen] testbench.cond.tx_rx_cond.
[treeopen] testbench.cond.tx_rx_cond.
    low_rx_cond.
[treeopen] testbench.cond.tx_rx_cond.
    low_tx_cond.
[sst_width] 213
[signals_width] 238
[sst_expanded] 1
[sst_vpaned_height] 187
@28
>2166
testbench.cond.IN_CLK_2MHz
>2166
testbench.cond.CLK_1MHz
>2166
testbench.cond.CLK_500KHz
>2166
testbench.cond.CLK_250KHz
@22
>2166
testbench.IN_CTRL_tb[3:0]
>2166
testbench.IN_TLP_tb[7:0]
>2166
testbench.IN_COM_tb[7:0]
>2166
testbench.IN_PAD_tb[7:0]
>2166
testbench.IN_SKP_tb[7:0]
>2166
testbench.IN_STP_tb[7:0]
>2166
testbench.IN_SDP_tb[7:0]
>2166
testbench.IN_END_tb[7:0]
>2166
testbench.IN_EDB_tb[7:0]
>2166
testbench.IN_FTS_tb[7:0]
>2166
testbench.IN_TLP_tb[7:0]
>2166
testbench.IN_IDL_tb[7:0]
>2166
testbench.cond.tx_input_cond.muxOUT[7:0]
@c00022
>2166
testbench.cond.tx_rx_cond.IN_LANE0[7:0]
@28
>2166
(0)testbench.cond.tx_rx_cond.IN_LANE0[7:0]
>2166
(1)testbench.cond.tx_rx_cond.IN_LANE0[7:0]
>2166
(2)testbench.cond.tx_rx_cond.IN_LANE0[7:0]
>2166
(3)testbench.cond.tx_rx_cond.IN_LANE0[7:0]
>2166
(4)testbench.cond.tx_rx_cond.IN_LANE0[7:0]
>2166
(5)testbench.cond.tx_rx_cond.IN_LANE0[7:0]
>2166
(6)testbench.cond.tx_rx_cond.IN_LANE0[7:0]
>2166
(7)testbench.cond.tx_rx_cond.IN_LANE0[7:0]
@1401200
>2166
-group_end
@c00022
>2166
testbench.cond.tx_rx_cond.IN_LANE1[7:0]
@28
>2166
(0)testbench.cond.tx_rx_cond.IN_LANE1[7:0]
>2166
(1)testbench.cond.tx_rx_cond.IN_LANE1[7:0]
>2166
(2)testbench.cond.tx_rx_cond.IN_LANE1[7:0]
>2166
(3)testbench.cond.tx_rx_cond.IN_LANE1[7:0]
>2166
(4)testbench.cond.tx_rx_cond.IN_LANE1[7:0]
>2166
(5)testbench.cond.tx_rx_cond.IN_LANE1[7:0]
>2166
(6)testbench.cond.tx_rx_cond.IN_LANE1[7:0]
>2166
(7)testbench.cond.tx_rx_cond.IN_LANE1[7:0]
@1401200
>2166
-group_end
```

```
@22
>2166
testbench.cond.tx_rx_cond.IN_LANE2[7:0]
>2166
testbench.cond.tx_rx_cond.IN_LANE3[7:0]
@c00022
>2166
testbench.cond.tx_rx_cond.LANE[3:0]
@28
>2166
(0)testbench.cond.tx_rx_cond.LANE[3:0]
>2166
(1)testbench.cond.tx_rx_cond.LANE[3:0]
>2166
(2)testbench.cond.tx_rx_cond.LANE[3:0]
>2166
(3)testbench.cond.tx_rx_cond.LANE[3:0]
@1401200
>2166
-group_end
@c00023
>2166
testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
@28
>0
(0)testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
(1)testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
(2)testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
(3)testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
(4)testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
(5)testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
(6)testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
(7)testbench.cond.tx_rx_cond.OUT_LANE0[7:0]
@1401201
-group_end
@22
>2166
testbench.cond.tx_rx_cond.OUT_LANE1[7:0]
>2166
testbench.cond.tx_rx_cond.OUT_LANE2[7:0]
>2166
testbench.cond.tx_rx_cond.OUT_LANE3[7:0]
>2166
testbench.cond.bj_rx_cond.out[7:0]
@28
>0
testbench.cond.tx_rx_cond.low_rx_cond.reg_lane0
    .CTR[2:0]
[pattern_trace] 1
[pattern_trace] 0
```

VI. Ejemplos de resultados

En la figura 3 se muestran las simulaciones de la capa física del PCIe, esto mediante la comunicación entre el Tx y el Rx.

Referencias

[1] “Icarus verilog,” http://iverilog.icarus.com/.

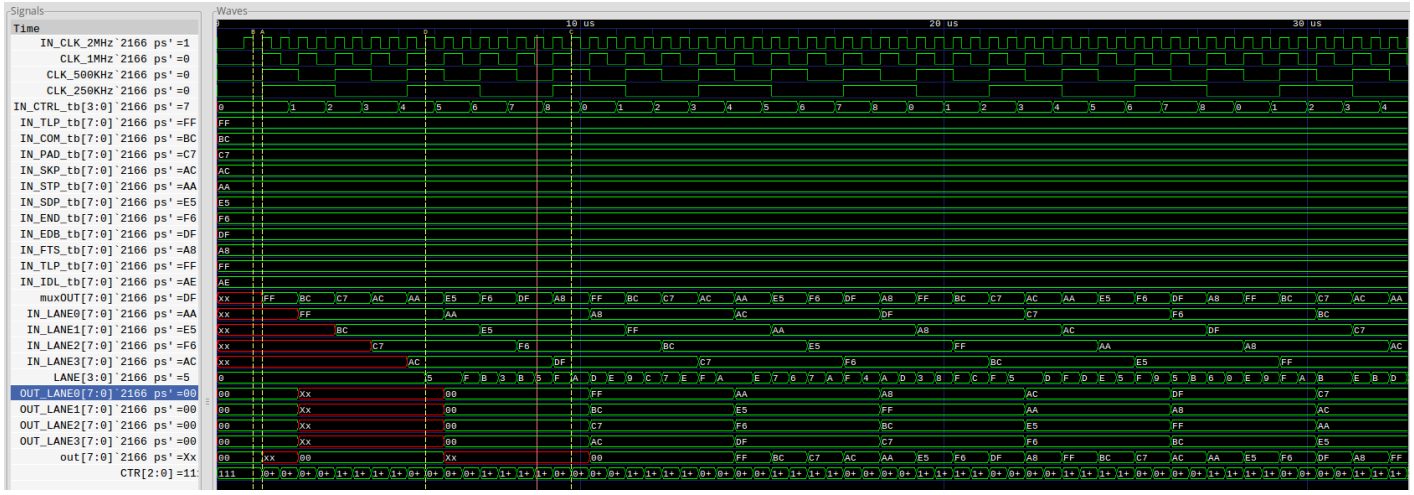

Figura 3. Pruebas del registro de desplazamiento con bandera -Ttyp y frecuencia de 16.67 MHz.