thesis

gds

Jan 17, 2024

CONTENTS:

CHAPTER

ONE

FPGA PACKAGE

class fpga.**FlipFlop**(*name*, *loc*, *w_out*, *w_clock*, *w_in*, *bel=''*)

Bases: object

Modelo de FPGA-FF.

Parameters

- **name** (*str*) – Nombre de la instancia FF.
- **loc** (*str*) – Posición de la celda a la que pertenece el FF en la FPGA. El formato de esta opción es un string formado por dos números enteros (coordenadas X, Y) separados por una coma.
- **w_out** (*str*) – Nombre del cable que contiene la señal de salida del FF.
- **w_clock** (*str*) – Nombre del cable que lleva la señal de reloj al FF.
- **w_in** (*str*) – Nombre del cable que contiene la señal de entrada al FF.
- **bel** (*{'A', 'B', 'C', 'D'}*) – Restrición BEL que indica qué FF concreto es ocupado dentro de la celda.
- **pin** (*str*) – Restricción de los pines lógicos (“I”) a los pines físicos (“A”) de la LUT. El formato es el mismo que el de la restricción LOCK_PINS de XDC, i.e., un string de elementos ‘I<puerto lógico>:A<puerto físico>’ separados por comas, donde el puerto lógico es ‘I0’, y los puertos físicos varían de ‘A1’ a ‘A6’.

bel

Restricciones BEL.

impl()

Esta función devuelve un ‘string’ que contiene el código en lenguaje Verilog necesario para implementar el FF inicializada.

Returns
Cadena que representa el código Verilog para la implementación del FF inicializado, con los parámetros actuales de la instancia.

Return type
str

loc

Posición del FF en la FPGA.

name

Nombre de la instancia FF.

w_clock

Señal de reloj.

w_in

Señal de entrada.

w_out

Señal de salida.

class fpga.**Lut1**(*name*, *init*, *loc*, *w_out*, *w_in*, *bel=''*, *pin=''*)

Bases: object

Modelo de FPGA-LUT de 1 entrada.

Parameters

- **name** (*str*) – Nombre de la instancia LUT.
- **init** (*str*) – Valor numérico que representa la función realizada por la LUT. La relación entrada/salida se asigna de la forma:

I0	Out
0	init[0]
1	init[1]

  donde init[i] es el i-ésimo bit del número 'init' en representación binaria LSB (i.e., init[0] es el bit menos significativo -"más a la izquierda"- de init).
- **loc** (*str*) – Posición de la celda a la que pertenece la LUT en la FPGA. El formato de esta opción es un string formado por dos números enteros (coordenadas X, Y) separados por una coma.
- **w_out** (*str*) – Nombre del cable que contiene la señal de salida de la LUT.
- **w_in** (*str*) – Nombre del cable que contiene la señal de entrada a la LUT. Se trata de una string con el nombre de la entrada correspondiente al pin I0.
- **bel** (*{'A', 'B', 'C', 'D'}*) – Restricción BEL que indica qué LUT concreta es ocupada dentro de la celda.
- **pin** (*str*) – Restricción de los pines lógicos ("I") a los pines físicos ("A") de la LUT. El formato es el mismo que el de la restricción LOCK_PINS de XDC, i.e., un string de elementos 'I<puerto lógico>:A<puerto físico>' separados por comas, donde el puerto lógico es 'I0', y los puertos físicos varían de 'A1' a 'A6'.

bel

Restricciones BEL.

impl()

Esta función devuelve un 'string' que contiene el código en lenguaje Verilog necesario para implementar la LUT inicializada.

Returns

Cadena que representa el código Verilog para la implementación de la LUT inicializada, con los parámetros actuales de la instancia.

Return type

str

init

función realizada por la LUT.

loc

Posición de la LUT en la FPGA.

name

Nombre de la instancia LUT.

pin

Restricciones de pines.

w_in

Señal de entrada.

w_out

Señal de salida.

class fpga.**Lut2**(*name*, *init*, *loc*, *w_out*, *w_in*, *bel=''*, *pin=''*)

Bases: `object`

Modelo de FPGA-LUT de 2 entradas.

Parameters

- **name** (*str*) – Nombre de la instancia LUT.
- **init** (*str*) – Valor numérico que representa la función realizada por la LUT. La relación entrada/salida se asigna de la forma:

I1	I0	Out
0	0	init[0]
0	1	init[1]
1	0	init[2]
1	1	init[3]

  donde init[i] es el i-ésimo bit del número 'init' en representación binaria LSB (i.e., init[0] es el bit menos significativo -"más a la izquierda"- de init).
- **loc** (*str*) – Posición de la celda a la que pertenece la LUT en la FPGA. El formato de esta opción es un string formado por dos números enteros (coordenadas X, Y) separados por una coma.
- **w_out** (*str*) – Nombre del cable que contiene la señal de salida de la LUT.
- **w_in** (*str*) – Nombre del cable que contiene la señal de entrada a la LUT. Se trata de una string con el nombre de la entrada correspondiente al pin I0.
- **bel** (*{'A', 'B', 'C', 'D'}*) – Restricción BEL que indica qué LUT concreta es ocupada dentro de la celda.
- **pin** (*str*) – Restricción de los pines lógicos ("I") a los pines físicos ("A") de la LUT. El formato es el mismo que el de la restricción LOCK_PINS de XDC, i.e., un string de elementos 'I<puerto lógico>:A<puerto físico>' separados por comas, donde el puerto lógico es 'I0', y los puertos físicos varían de 'A1' a 'A6'.

bel

Restricciones BEL.

impl()

Esta función devuelve un 'string' que contiene el código en lenguaje Verilog necesario para implementar la LUT inicializada.

Returns
Cadena que representa el código Verilog para la implementación de la LUT inicializada, con los parámetros actuales de la instancia.

Return type
str

init

función realizada por la LUT.

loc

Posición de la LUT en la FPGA.

name

Nombre de la instancia LUT.

pin

Restricciones de pines.

w_in

Lista de señales de entrada.

w_out

Señal de salida.

class fpga.**Lut3**(*name*, *init*, *loc*, *w_out*, *w_in*, *bel=''*, *pin=''*)

Bases: object

Modelo de FPGA-LUT de 3 entradas.

Parameters

- **name** (*str*) – Nombre de la instancia LUT.
- **init** (*str*) – Valor numérico que representa la función realizada por la LUT. La relación entrada/salida se asigna de la forma:

I2	I1	I0	Out
0	0	0	init[0]
0	0	1	init[1]
0	1	0	init[2]
.	.	.	.
1	1	1	init[7]

  donde init[i] es el i-ésimo bit del número 'init' en representación binaria LSB (i.e., init[0] es el bit menos significativo -"más a la izquierda"- de init).

- **loc** (*str*) – Posición de la celda a la que pertenece la LUT en la FPGA. El formato de esta opción es un string formado por dos números enteros (coordenadas X, Y) separados por una coma.
- **w_out** (*str*) – Nombre del cable que contiene la señal de salida de la LUT.
- **w_in** (*str*) – Nombre del cable que contiene la señal de entrada a la LUT. Se trata de una string con el nombre de la entrada correspondiente al pin I0.

- **bel** (*{'A', 'B', 'C', 'D'}*) – Restricción BEL que indica qué LUT concreta es ocupada dentro de la celda.
- **pin** (*str*) – Restricción de los pines lógicos (“I”) a los pines físicos (“A”) de la LUT. El formato es el mismo que el de la restricción LOCK_PINS de XDC, i.e., un string de elementos ‘I<puerto lógico>:A<puerto físico>’ separados por comas, donde el puerto lógico es ‘I0’, y los puertos físicos varían de ‘A1’ a ‘A6’.

bel

Restricciones BEL.

impl()

Esta función devuelve un ‘string’ que contiene el código en lenguaje Verilog necesario para implementar la LUT inicializada.

Returns
Cadena que representa el código Verilog para la implementación de la LUT inicializada, con los parámetros actuales de la instancia.

Return type
str

init

función realizada por la LUT.

loc

Posición de la LUT en la FPGA.

name

Nombre de la instancia LUT.

pin

Restricciones de pines.

w_in

Lista de señales de entrada.

w_out

Señal de salida.

class fpga.**Lut4**(*name*, *init*, *loc*, *w_out*, *w_in*, *bel=''*, *pin=''*)

Bases: object

Modelo de FPGA-LUT de 4 entradas.

Parameters

- **name** (*str*) – Nombre de la instancia LUT.
- **init** (*str*) – Valor numérico que representa la función realizada por la LUT. La relación entrada/salida se asigna de la forma:

I3	I2	I1	I0	Out
0	0	0	0	init[0]
0	0	0	1	init[1]
0	0	1	0	init[2]
.	.	.	.	.
1	1	1	1	init[15]

donde init[i] es el i-ésimo bit del número 'init' en representación binaria LSB (i.e., init[0] es el bit menos significativo -"más a la izquierda"- de init).

- **loc** (*str*) – Posición de la celda a la que pertenece la LUT en la FPGA. El formato de esta opción es un string formado por dos números enteros (coordenadas X, Y) separados por una coma.
- **w_out** (*str*) – Nombre del cable que contiene la señal de salida de la LUT.
- **w_in** (*str*) – Nombre del cable que contiene la señal de entrada a la LUT. Se trata de una string con el nombre de la entrada correspondiente al pin I0.
- **bel** (*{'A', 'B', 'C', 'D'}*) – Restricción BEL que indica qué LUT concreta es ocupada dentro de la celda.
- **pin** (*str*) – Restricción de los pines lógicos ("I") a los pines físicos ("A") de la LUT. El formato es el mismo que el de la restricción LOCK_PINS de XDC, i.e., un string de elementos 'I<puerto lógico>:A<puerto físico>' separados por comas, donde el puerto lógico es 'I0', y los puertos físicos varían de 'A1' a 'A6'.

bel

Restricciones BEL.

impl()

Esta función devuelve un 'string' que contiene el código en lenguaje Verilog necesario para implementar la LUT inicializada.

Returns
Cadena que representa el código Verilog para la implementación de la LUT inicializada, con los parámetros actuales de la instancia.

Return type
str

init

función realizada por la LUT.

loc

Posición de la LUT en la FPGA.

name

Nombre de la instancia LUT.

pin

Restricciones de pines.

w_in

Lista de señales de entrada.

w_out

Señal de salida.

class fpga.**Lut6**(*name*, *init*, *loc*, *w_out*, *w_in*, *bel=''*, *pin=''*)

Bases: object

Modelo de FPGA-LUT de 6 entradas.

Parameters

- **name** (*str*) – Nombre de la instancia LUT.

- **init** (*str*) – Valor numérico que representa la función realizada por la LUT. La relación entrada/salida se asigna de la forma:

I5	I4	I3	I2	I1	I0	Out
0	0	0	0	0	0	init[0]
0	0	0	0	0	1	init[1]
0	0	0	0	1	0	init[2]
.	.	.	.	.	.	.
1	1	1	1	1	1	init[63]

  donde init[i] es el i-ésimo bit del número 'init' en representación binaria LSB (i.e., init[0] es el bit menos significativo -"más a la izquierda"- de init).

- **loc** (*str*) – Posición de la celda a la que pertenece la LUT en la FPGA. El formato de esta opción es un string formado por dos números enteros (coordenadas X, Y) separados por una coma.
- **w_out** (*str*) – Nombre del cable que contiene la señal de salida de la LUT.
- **w_in** (*str*) – Nombre del cable que contiene la señal de entrada a la LUT. Se trata de una string con el nombre de la entrada correspondiente al pin I0.
- **bel** (*{'A', 'B', 'C', 'D'}*) – Restricción BEL que indica qué LUT concreta es ocupada dentro de la celda.
- **pin** (*str*) – Restricción de los pines lógicos ("I") a los pines físicos ("A") de la LUT. El formato es el mismo que el de la restricción LOCK_PINS de XDC, i.e., un string de elementos 'I<puerto lógico>:A<puerto físico>' separados por comas, donde el puerto lógico es 'I0', y los puertos físicos varían de 'A1' a 'A6'.

bel

Restricciones BEL.

impl()

Esta función devuelve un 'string' que contiene el código en lenguaje Verilog necesario para implementar la LUT inicializada.

Returns

Cadena que representa el código Verilog para la implementación de la LUT inicializada, con los parámetros actuales de la instancia.

Return type

str

init

función realizada por la LUT.

loc

Posición de la LUT en la FPGA.

name

Nombre de la instancia LUT.

pin

Restricciones de pines.

w_in

Lista de señales de entrada.

w_out
Señal de salida.

1.1 Submodules

1.1.1 fpga.fabric module

Este módulo contiene una lista de algunos recursos relacionados con la estructura interna de la FPGA; estas cantidades son invariantes (no se modifican durante la ejecución de un programa).

fpga.fabric.**pynqz2**
Matriz de tamaño 114x150 cuya entrada i,j contiene los parámetros físicos de la celda SLICE_X{i}Y{j} para la FPGA implementada sobre una placa de desarrollo PYNQ-Z2. Si la celda no existe en esta FPGA se devuelve *None*.

Type
lista de lista de str

1.1.2 fpga.interfaz_pcps module

fpga.interfaz_pcps.**bitstr_to_bytestr**(*bitstr_in*, *bitstr_width*)
Toma una lista de bits *bitstr_in* y la convierte a una lista de bytes del tamaño mínimo necesario para alojar *bitstr_width* bits.

Parameters
- **bitstr_in** (*lista de {0,1}*) – Lista de bits.
- **bitstr_width** (*int*) – Número de bits mínimos acomodados a la salida.

Returns
Lista de bytes.

Return type
lista de u8

fpga.interfaz_pcps.**bitstr_to_int**(*entrada*)
Convierte una lista de bits en su correspondiente número decimal (entero).

Parameters
entrada (*lista de {0,1}*) – Representación binaria de un valor.

Returns
Representación decimal de *entrada*.

Return type
int

fpga.interfaz_pcps.**bytestr_to_bitstr**(*bytestr*, *bitstr_width*)
Esta función toma una lista de bytes *bytestr* y los aloja en una lista de bits de tamaño *bitstr_width*. Si *bitstr_width* es insuficiente, la lista de bits se truncará.

Parameters
- **bytestr** (*lista de u8*) – Lista de enteros entre 0 y 255 (bytes).
- **bitstr_width** (*int*) – Tamaño de la lista de bits resultante.

Returns
Lista de bits.

Return type
lista de {0,1}

fpga.interfaz_pcps.**bytestr_to_int**(*entrada*)

Convierte una lista de bytes en su correspondiente número decimal (entero).

Parameters
entrada (*lista de u8*) – Lista de *int* entre 0 y 255 que representa un valor en base 256.

Returns
Valor entero (base 10) dado por la lista *entrada*.

Return type
int

fpga.interfaz_pcps.**calc**(*serialport*, *buffer_out_width*)

Esta funcion dispara un ciclo RDY->CALC->PRINT->RDY en una máquina de estados finitos conectada a traves de *serialport* y devuelve un bitstr de tamaño *buffer_out_width*.

Parameters

- **serialport** (*pyserial.Serial*) – Objeto *Serial* del paquete 'pyserial'.
- **buffer_out_width** (*int*) – Tamaño del 'bitstr' devuelto.

Returns
Respuesta binaria devuelta por la máquina de estados finitos después de realizar el procesamiento en FPGA.

Return type
lista de {0,1}

fpga.interfaz_pcps.**int_to_bitstr**(*entrada*)

Esta funcion convierte un numero *entrada* en su representacion en base 2 ('bitstr', lista de 0/1). El número más a la dcha del bitstr (result[0]) es el menos significativo.

Parameters
entrada (*int*) – Valor entero en base 10.

Returns
Representación binaria de *entrada*.

Return type
lista de {0,1}

fpga.interfaz_pcps.**int_to_bytestr**(*entrada*)

Esta funcion convierte un numero *entrada* en su representacion en base 256 (lista de u8 'bytestr'). El número más a la dcha del bytestr (result[0]) es el menos significativo.

Parameters
entrada (*int*) – Número a convertir en 'bytestr'.

Returns
Lista que representa *entrada* en base 256.

Return type
lista de u8

fpga.interfaz_pcps.**print_bitstr**(*bitstr_in*)

Esta función imprime una lista de bits como una 'string'.

Parameters
bitstr_in (*lista de {0,1}*) – Lista de bits a imprimir.

fpga.interfaz_pcps.**print_bytestr**(*bytestr_in*)

Esta función imprime una lista de bytes como una 'string'.

Parameters
bytestr_in (*lista de u8*) – Lista de bytes a imprimir.

fpga.interfaz_pcps.**receive_bytestr**(*serialport*, *bytestr_size*)

Recibe *bytestr_size* bytes a traves del puerto serie *serialport*.

Parameters

- **serialport** (*pyserial.Serial*) – Objeto *Serial* del paquete 'pyserial'.
- **bytestr_size** (*int*) – Número de bytes a leer a través del puerto *serialport*.

Returns
Lista de bytes leídos del puerto serie.

Return type
lista de u8

fpga.interfaz_pcps.**receive_u32**(*serialport*)

Esta funcion implementa un bucle para leer 4 bytes del puerto serie *serialport*, y los devuelve en formato u32 (entero). Si no habia datos en el buffer de entrada a la hora de llamar a la funcion devuelve -1.

Parameters
serialport (*pyserial.Serial*) – Objeto *Serial* del paquete 'pyserial'.

Returns
En caso de éxito, entero leído de *serialport*. En caso contrario, -1.

Return type
int

fpga.interfaz_pcps.**receive_u8**(*serialport*)

Esta funcion implementa un bucle para leer 1 byte del puerto serie *serialport*, y devuelve dicho byte en formato u8 (un numero entre 0 y 255). Si no habia datos en el buffer de entrada a la hora de llamar a la funcion, devuelve -1.

Parameters
serialport (*pyserial.Serial*) – Objeto *Serial* del paquete 'pyserial'.

Returns
En caso de éxito, 'byte' leído del puerto serie en formato decimal. En otro caso, -1.

Return type
int

fpga.interfaz_pcps.**resize_array**(*array_old*, *array_size*)

Esta funcion toma un *array_old* de tamaño arbitrario y transfiere los elementos a un array de tamaño *array_size*, empezando por el dígito menos significativo (a la dcha.). Si *array_size* no es suficiente para contener todo el *array_old*, la salida estará truncada. Si *array_size* es excesivo para contener *array_old*, los huecos que sobren se rellenan con 0.

Parameters

- **array_old** (*lista*) – Array a redimensionar (tipo de datos arbitrario).

- **array_size** (*int*) – Tamaño del nuevo array.

Returns
Lista del mismo tipo que *array_old* redimensionada.

Return type
lista

fpga.interfaz_pcps.**scan**(*serialport*, *bitstr_in*, *buffer_in_width*)

Esta funcion dispara un ciclo RDY->SCAN->RDY en una máquina de estados finitos conectada a traves de *serialport* y carga a la máquina un bitstr de tamaño *buffer_in_width*.

Parameters

- **serialport** (*pyserial.Serial*) – Objeto *Serial* del paquete 'pyserial'.
- **bitstr_in** (*lista de {0,1}*) – Lista de bits a cargar en la máquina de estados finitos, conteniendo datos de entrada necesarios para la operación de la FPGA.
- **buffer_in_width** (*int*) – Número de bits escritos en la máquina de estados finitos.

fpga.interfaz_pcps.**send_bytestr**(*serialport*, *bytestr_out*)

Envía la lista de bytes *bytestr_out* a traves del puerto serie *serialport*.

Parameters

- **serialport** (*pyserial.Serial*) – Objeto *Serial* del paquete 'pyserial'.
- **bytestr_out** (*lista de u8*) – Lista de bytes (entero entre 0 y 255) a enviar a través del puerto serie.

Return type
0

fpga.interfaz_pcps.**send_u8**(*serialport*, *number*)

Esta funcion envia un numero en formato u8 a traves de *serialport*. Si *number* es mayor que 256, se enviará el resto: number%256.

Parameters

- **serialport** (*pyserial.Serial*) – Objeto *Serial* del paquete 'pyserial'.
- **number** (*u8*) – Byte (entero entre 0 y 255) a enviar a través del puerto serie.

Return type
0

1.1.3 fpga.ring_osc module

Este módulo contiene una serie de clases y funciones para implementar y medir una matriz de osciladores de anillo en FPGA, tanto estándar como de Galois.

class fpga.ring_osc.**Dominio**(*N_osc=10*, *x0=0*, *x1=inf*, *dx=1*, *y0=0*, *y1=inf*, *dy=1*, *directriz='y'*)

Bases: object

Este objeto contiene las localizaciones de un conjunto de osciladores dispuestos atendiendo a diversos parámetros geométricos. Si directriz=y, estos se colocan formando una matriz rectangular, la cual crece en dirección y en incrementos de dy. Cuando se alcanza el límite y1, la matriz se incrementa una cantidad dx en la dirección x, y vuelve a la coordenada y0. Si directriz=x, el comentario anterior se aplica substituyendo x <-> y.

Parameters

- **N_osc** (*int, opcional*) – Número de osciladores del dominio.

- **x0** (*int, opcional*) – Coordenada X del primer oscilador de la matriz.
- **x1** (*int, opcional*) – Coordenada X máxima de la matriz (no se sobrepasará).
- **dx** (*int, opcional*) – Incremento de la coordenada X. Notar que habitualmente en las FPGA de Xilinx se reservan las coordenadas X par/impar para distintos tipos de celda ('0' y '1').
- **y0** (*int, opcional*) – Coordenada Y del primer oscilador de la matriz.
- **y1** (*int, opcional*) – Coordenada Y máxima de la matriz (no se sobrepasará).
- **dy** (*int, opcional*) – Incremento de la coordenada Y.
- **directriz** (*char, opcional*) – Dirección de crecimiento de la matriz (hasta llegar a la coordenada máxima). Puede ser 'y' o 'x'.

N_osc

Número de osciladores del anillo.

directriz

Dirección de crecimiento de la matriz de anillos.

dx

Incremento de la coordenada 'X'.

dy

Incremento de la coordenada 'Y'.

help()

Ayuda de la clase 'Dominio'.

osc_coord

Lista de las coordenadas de los anillos, dados en forma de par x,y.

x0

Coordenada 'X' inicial.

x1

Coordenada 'X' final.

y0

Coordenada 'Y' inicial.

y1

Coordenada 'Y' final.

class fpga.ring_osc.**GaloisMatrix**(*N_inv=3*, *dominios=<fpga.ring_osc.Dominio object>*, *bel=''*, *pin=''*, *pdl=False*, *trng=0*, *poly=-1*, *inverted_end=True*)

Bases: object

Objeto que contiene una matriz de osciladores de anillo de Galois.

Parameters

- **N_inv** (*int, opcional*) – Número de inversores de cada oscilador.
- **dominios** (*Dominio* | list(*Dominio*), opcional) – Osciladores que forman la matriz. Se construye como una lista de objetos 'Dominio'. Si solo pasamos un dominio de osciladores podemos pasar un objeto 'Dominio', en lugar de una lista.

- **bel** (*char | list(char), opcional*) – Dado que todos los anillos de la matriz son idénticos por diseño, esta opción es la misma que la aplicada para un solo oscilador (ver 'bel' en 'GaloisRing').
- **pin** (*str | list(str), opcional*) – Dado que todos los anillos de la matriz son idénticos por diseño, esta opción es la misma que la aplicada para un solo oscilador (ver 'pin' en 'GaloisRing').
- **pdl** (*bool, opcional*) – Si 'True' se utilizan modelos LUT6 para los inversores, permitiendo utilizar 5 puertos para configurar el anillo mediante PDL. Si 'False' se utilizan modelos LUT1 para los inversores y LUT2 para el enable AND.
- **trng** (*int, opcional*) – Esta variable modifica de manera profunda el diseño de la matriz GARO implementada. Si se incluye este parámetro como un entero mayor que cero, el diseño genera un arreglo de bits generados por cada GARO como TRNG. En este caso, el buffer de salida del diseño tiene un tamaño 'trng' dado por esta variable. Alternativamente, si no se incluye esta opción (o vale '0'), el diseño incluye un medidor de sesgo que devuelve el valor del sesgo de cada GARO, en lugar de producir un arreglo de bits.
- **poly** (*int, opcional*) – Esta variable determina el polinomio a implementar en hardware. Si su valor es negativo (por defecto), entonces se implementa un anillo de Galois genérico configurable en tiempo de ejecución. En caso contrario, se producirá un anillo que implementa un polinomio fijo, utilizando la misma codificación que la función ',edir()'. Esto puede ahorrar una cierta cantidad de recursos hardware, a costa de perder flexibilidad.
- **inverted_end** (*bool, opcional*) – Si esta opción es 'True', se añadirá un inversor al final del anillo, lo cual según parece evita acoplos entre anillos cercanos. Notar que la presencia o no de este inversor cambia el número de elementos a efectos de las opciones 'bel' y 'pin'.

gen_garomatrix(*out_name='garomatrix.v'*)

Genera un diseño 'out_name' en formato Verilog con la implementación de los dominios introducidos durante la inicialización del objeto. El principal uso de esta función es dentro de la función 'implement()'.

Parameters

out_name (*str, opcional*) – Nombre del fichero de salida.

help()

Ayuda de la clase 'GaloisMatrix'.

implement(*projname='project_garomatrix'*, *projdir='.'*, *njobs=4*, *files=True*, *board='pynqz2'*, *qspi=False*, *routing=False*, *pblock=False*, *data_width=32*, *buffer_out_width=32*)

Copia en el directorio 'self.projdir' todos los archivos necesarios para implementar una matriz de osciladores de anillo de Galois con medición del sesgo ("bias") y comunicación pc <-> microprocesador <-> FPGA.

Parameters

- **projname** (*str, opcional*) – Nombre del proyecto de Vivado.
- **projdir** (*str, opcional*) – Directorio donde se creará el proyecto de Vivado y las fuentes (por defecto el directorio de trabajo actual).
- **njobs** (*int, opcional*) – Número de núcleos que utiilizará Vivado paralelamente para la síntesis/implementación.
- **debug** (*bool, opcional*) – Si "True", se implementará una matriz de divisores de reloj de frecuencia conocida, lo que permite depurar el diseño al conocer qué resultados deben salir.
- **files** (*bool, opcional*) – Si "True", pinta los archivos necesarios para implementar la matriz en FPGA. Esta opción se puede desactivar (False) cuando queremos configurar un

objeto tipo 'Romatrix' pero no vamos a implementarla físicamente (por ejemplo porque ya lo hemos hecho y solo queremos medir, o vamos a simularla sin realizarla).

- **board** (*str, opcional*) – Placa de desarrollo utilizada en el proyecyo. Las opciones soportadas son: 'pynqz2', 'zybo', 'cmoda7_15t' o 'cmoda7_35t'.
- **qspi** (*bool, opcional*) – Si "True", el flujo de diseño incluirá el guardado del bitstream en la memoria flash de la placa para que se auto-programe al encenderse.
- **routing** (*bool, opcional*) – Si "True", el flujo de diseño incluirá el cableado de los inversores después de la síntesis. Esto aumenta las probabilidades de que la herramienta haga un cableado idéntico, pero es recomendable comprobarlo. (NOTA: no tengo garantías de que esta opción sea del todo compatible con -qspi).
- **pblock** (*bool, opcional*) – Si esta opción es 'True' se inserta la matriz en un pblock tal que el espacio dentro del bloque se excluye para toda lógica que no sea la propia matriz.
- **data_width** (*int, opcional*) – Esta opción especifica la anchura del canal de datos PS<–>PL.
- **buffer_out_width** (*int, opcional*) – Esta opción especifica la anchura de la palabra de respuesta (i.e., de la medida).

medir(*puerto='/dev/ttyS1'*, *osc=[0]*, *pdl=[0]*, *N_rep=1*, *resol=14*, *poly=0*, *fdiv=9*, *bias=False*, *log=False*, *verbose=True*, *baudrate=9600*)

Esta función mide la frecuencia de una matriz de osciladores de Galois 'GaloisMatrix', una vez esta ha sido implementado en FPGA. El resultado se devuelve como un objeto 'Tensor'.

Parameters

- **puerto** (*str*) – Esta opción especifica el puerto serie al que se conecta la FPGA.
- **osc** (*int | list(int)*) – Lista de osciladores a medir.
- **pdl** (*int | list(int)*) – Lista de PDL a medir.
- **N_rep** (*int*) – Número de repeticiones a medir.
- **resol** (*int*) – log_2 del número de ciclos de referencia a completar para dar por terminada la medida (por defecto 14, i.e., 2^14 = 16384 ciclos).
- **poly** (*int*) – Esta variable indica el índice del polinomio a medir.
- **fdiv** (*int*) – log_2 del factor de división menos uno, para el reloj de muestreo (por defecto 9, i.e., 2^(9+1)=1024; cin f_ref=100 MHz esto supone una frecuencia de muestre f_s=97.65 kHz).
- **bias** (*bool*) – Si se pasa esta opción como "True" el resultado se dará en tanto por 1.
- **log** (*bool*) – Si se pasa "True" se escriben algunos datos a modo de log.
- **verbose** (*bool*) – Si se pasa "True" se pinta una barra de progreso de la medida. Desactivar esta opción ("False") hace más cómodo utilizar esta función en un bucle.
- **baudrate** (*int*) – Tasa de transferencia del protocolo serie UART PC<–>PS. Debe concordar con el programa compilador en PS.

save(*file_name*)

'Wrapper' para guardar objetos serializados con el módulo 'pickle'.

class fpga.ring_osc.**GaloisRing**(*name*, *N_inv*, *loc*, *bel=''*, *pin=''*, *pdl=False*, *poly=-1*, *inverted_end=True*)

Bases: object

Lista de elementos (LUT) que constituyen un oscilador de anillo de Galois junto con la información necesaria para su implementación en FPGA utilzando el software 'Vivado'.

Parameters

- **name** (*str*) – Nombre utilizado para identificar el anillo.
- **N_inv** (*int*) – Número de inversores que forman el anillo.
- **loc** (*str*) – Parámetro 'loc' de la primera LUT del anillo (ver clase 'Lut'). Solo se da la posición de la primera LUT porque los anillos siempre se construyen ocupando todos los 'bel' de una celda antes de pasar a la celda inmediatamente superior (de modo que, dadas las coordenadas del primer elemento, las demás están predeterminadas). La "celda inmediatamente superior" es X+1 si X es par, Y+1 si X es impar.
- **bel** (*char o lista de char*) – Lista de parámetros 'bel' para cada elemento del anillo (ver clase 'Lut'). Cada elemento de la lista se aplica en orden correlativo al elemento del anillo (la primera restricción se aplica al primer elemento, la segunda al segundo, etc). Pueden darse menos restricciones que inversores forman el anillo, pero entonces quedarán LUT sin fijar. Notar además que el valor de esta restricción es excluyente entre LUT que forman parte de la misma celda: cada celda tiene cuatro posibles posiciones A, B, C y D, y dos LUT no pueden ocupar el mismo espacio. Esta función no avisa de esta violación: el usuario es responsable de que los valores 'bel' introducidos sean todos distintos entre sí en grupos de cuatro. En otro caso, el diseño fallará. Además de una lista de caracteres, esta opción admite un único caracter, pero entonces solo se restringirá la primera LUT. Notar que el último elemento en un anillo 'GaloisRing' siempre será un flip-flop, y el penúltimo elemento depende de la opción 'inverted_end': si 'True', el número de elementos total del anillo será igual a N_inv+2, siendo el penúltimo elemento siempre una LUT1 (inversor final). Si por el contrario esta opción es 'False' entonces el número de elementos del anillo será N_inv+1.
- **pin** (*str o lista de str*) – Lista de parámetros 'pin' de cada LUT que forma parte del anillo (ver clase 'Lut'). Cada elemento de la lista se aplica en orden correlativo al elemento del anillo (la primera restricción se aplica al primer elemento, la segunda al segundo, etc). Pueden darse menos restricciones que inversores forman el anillo, pero entonces quedarán LUT sin mapear. Notar que el mapeo debe ser coherente con el tipo de LUT que se está fijando, y en particular los inversores inicial y final siempre son de tipo LUT1 (i.e., solo se puede fijar el pin 'I0'). Notar que, dado que el último elemento (N_inv+2 o N_inv+1, dependiendo de la opción 'inverted_end') de un anillo 'GaloisRing' es un flip-flop, a efectos de la opción "pin" este se ignora (esta opción restringe solo los pines de las LUT, no del flip-flop). Por definición, el puerto lógico 'I0' siempre es el que recoge la señal del elemento precedente en el bucle.
- **pdl** (*bool, opcional*) – Si 'True' se utilizan modelos LUT6 para los inversores, permitiendo utilizar 5 puertos para configurar el anillo mediante PDL. Si 'False' se utilizan modelos LUT1 para los inversores y LUT2 para el enable AND.
- **poly** (*int, opcional*) – Esta variable determina el polinomio a implementar en hardware. Si su valor es negativo (por defecto), entonces se implementa un anillo de Galois genérico configurable en tiempo de ejecución. En caso contrario, se producirá un anillo que implementa un polinomio fijo, utilizando la misma codificación que la función ',edir()'. Esto puede ahorrar una cierta cantidad de recursos hardware, a costa de perder flexibilidad.
- **inverted_end** (*bool, opcional*) – Si esta opción es 'True', se añadirá un inversor al final del anillo, lo cual según parece evita acoplos entre anillos cercanos. Notar que la presencia o no de este inversor cambia el número de elementos a efectos de las opciones 'bel' y 'pin'.

N_inv

Número de inversores del anillo.

bel

Lista de restricciones BEL para cada elemento del anillo.Lista de restricciones BEL para cada elemento del anillo.

elements

Lista de objetos que conforman el anillo. Esta consisitirá en N_inv objetos 'Lut', más un inversor en caso de que la opción `inverted_end`=True, más un flip-flop 'FF'.

help()

Ayuda de la clase 'GaloisRing'.

loc

Parámetro LOC de la primera puerta del anillo.

name

Nombre del anillo instanciado.

pin

Lista de restricciones PIN para cada elemento del anillo.Lista de restricciones PIN para cada elemento del anillo.

class fpga.ring_osc.**StdMatrix**(*N_inv=3*, *dominios=<fpga.ring_osc.Dominio object>*, *bel=''*, *pin=''*, *pdl=False*)

Bases: object

Objeto que contiene una matriz de osciladores de anillo estándar.

Parameters

- **N_inv** (*int*) – Número de inversores de cada oscilador.
- **dominios** (*Dominio* o lista de *Dominio*) – Osciladores que forman la matriz. Se construye como una lista de objetos *Dominio*. Si solo pasamos un dominio de osciladores podemos pasar un objeto *Dominio*, en lugar de una lista.
- **bel** (*char | list(char)*) – Dado que todos los anillos de la matriz son idénticos por diseño, esta opción es la misma que la aplicada para un solo oscilador (ver *bel* en *StdRing*).
- **pin** (*str | list(str)*) – Dado que todos los anillos de la matriz son idénticos por diseño, esta opción es la misma que la aplicada para un solo oscilador (ver 'pin' en 'StdRing').
- **pdl** (*bool, opcional*) – Si *True* se utilizan modelos LUT6 para los inversores, permitiendo utilizar 5 puertos para configurar el anillo mediante PDL. Si *False* se utilizan modelos :obj:Lut1 para los inversores y :obj:Lut2 para el enable AND.

N_bits_osc

Número de bits necesarios para especificar cada oscilador.

N_bits_pdl

Número de bits necesarios para especificar cada PDL.

N_bits_resol

Número de bits necesarios para especificar la resolución del proceso de medida.

N_inv

Número de inversores de cada anillo de la matriz.

N_osc

Número total de osciladores de la matriz.

bel

Restricciones BEL de cada anillo.

dominios

Dominio de que consta la matriz.

Type
Lista de

Type
obj

gen_romatrix(*out_name='romatrix.v'*, *debug=False*)

Genera un diseño 'out_name' en formato Verilog con la implementación de los dominios introducidos durante la inicialización del objeto. El principal uso de esta función es dentro de la función 'implement()'.

Parameters

- **out_name** (*str, opcional*) – Nombre del fichero de salida.
- **debug** (*bool, opcional*) – Flag que indica si se debe geenrar un diseño de depuración en lugar de una verdadera matriz de osciladores de anillo. En el diseño de depuración se substituye cada anillo por un divisor de reloj de frecuencia conocida, lo que permite depurar la interfaz de medida.

help()

Ayuda de la clase 'StdMatrix'.

implement(*projname='project_romatrix'*, *projdir='.'*, *njobs=4*, *debug=False*, *files=True*, *board='pynqz2'*, *qspi=False*, *routing=False*, *pblock=False*, *data_width=32*, *buffer_out_width=32*, *f_clock=100*)

Copia en el directorio 'self.projdir' todos los archivos necesarios para implementar una matriz de osciladores de anillo con medición de la frecuencia y comunicación pc <-> microprocesador <-> FPGA.

Parameters

- **projname** (*str, opcional*) – Nombre del proyecto de Vivado.
- **projdir** (*str, opcional*) – Directorio donde se creará el proyecto de Vivado y las fuentes (por defecto el directorio de trabajo actual).
- **njobs** (*int, opcional*) – Número de núcleos que utiilizará Vivado paralelamente para la síntesis/implementación.
- **debug** (*bool, opcional*) – Si 'True', se implementará una matriz de divisores de reloj de frecuencia conocida, lo que permite depurar el diseño al conocer qué resultados deben salir.
- **files** (*bool, opcional*) – Si 'True', pinta los archivos necesarios para implementar la matriz en FPGA. Esta opción se puede desactivar (False) cuando queremos configurar un objeto tipo 'Romatrix' pero no vamos a implementarla físicamente (por ejemplo porque ya lo hemos hecho y solo queremos medir, o vamos a simularla sin realizarla).
- **board** (*str, opcional*) – Placa de desarrollo utilizada en el proyecyo. Las opciones soportadas son: 'pynqz2', 'zybo', 'cmoda7_15t' o 'cmoda7_35t'.
- **qspi** (*bool, opcional*) – Si "True", el flujo de diseño incluirá el guardado del bitstream en la memoria flash de la placa para que se auto-programe al encenderse.
- **routing** (*bool, opcional*) – Si "True", el flujo de diseño incluirá el cableado de los inversores después de la síntesis. Esto aumenta las probabilidades de que la herramienta haga un cableado idéntico, pero es recomendable comprobarlo. (NOTA: no tengo garantías de que esta opción sea del todo compatible con -qspi).

- **pblock** (*bool, opcional*) – Si esta opción es 'True' se inserta la matriz en un pblock tal que el espacio dentro del bloque se excluye para toda lógica que no sea la propia matriz.
- **data_width** (*int, opcional*) – Esta opción especifica la anchura del canal de datos PS<–>PL.
- **buffer_out_width** (*int, opcional*) – Esta opción especifica la anchura de la palabra de respuesta (i.e., de la medida).
- **f_clock** (*int, opcional*) – Frecuencia del reloj del diseño (en MHz).

medir(*puerto='/dev/ttyS1'*, *osc=[0]*, *pdl=[0]*, *N_rep=1*, *resol=17*, *f_ref=False*, *log=False*, *verbose=True*, *baudrate=9600*)

Esta función mide la frecuencia de una matriz de osciladores estándar 'StdMatrix', una vez esta ha sido implementado en FPGA. El resultado se devuelve como un objeto 'Tensor'.

Parameters

- **puerto** (*str, opcional*) – Esta opción especifica el puerto serie al que se conecta la FPGA.
- **osc** (*int | list(int), opcional*) – Lista de osciladores a medir.
- **pdl** (*int | list(int), opcional*) – Lista de PDL a medir.
- **N_rep** (*int, opcional*) – Número de repeticiones a medir.
- **resol** (*int, opcional*) – log_2 del número de ciclos de referencia a completar para dar por terminada la medida (por defecto 17, i.e., 2^17 = 131072 ciclos).
- **float** (*f_ref =*) – Frecuencia del reloj de referencia. Si se proporciona este valor, el resultado obtenido se devuelve en las mismas unidades en que se haya pasado este valor "f_ref".
- **opcional** – Frecuencia del reloj de referencia. Si se proporciona este valor, el resultado obtenido se devuelve en las mismas unidades en que se haya pasado este valor "f_ref".
- **log** (*bool, opcional*) – Si se pasa "True" se escriben algunos datos a modo de log.
- **verbose** (*bool, opcional*) – Si se pasa "True" se pinta una barra de progreso de la medida. Desactivar esta opción ("False") hace más cómodo utilizar esta función en un bucle.
- **baudrate** (*int, opcional*) – Tasa de transferencia del protocolo serie UART PC<–>PS. Debe concordar con el programa compilador en PS.

osc_list

Lista completa de osciladores que componen la matriz.

pdl

Valor booleano que indica si los anillos de la matriz son configurables mediante PDL.

pin

Restricciones PIN de cada anillo.

save(*file_name*)

'Wrapper' para guardar objetos serializados con el módulo 'pickle'.

Parameters

file_name (*str*) – Nombre del archivo de salida.

class fpga.ring_osc.**StdRing**(*name*, *N_inv*, *loc*, *bel=''*, *pin=''*, *pdl=False*)

Bases: object

Este objeto contiene una lista de elementos (LUT) que constituyen un oscilador de anillo estándar junto con la información necesaria para su implementación en FPGA utilzando el software 'Vivado'.

Parameters

- **name** (*str*) – Nombre utilizado para identificar el anillo.
- **N_inv** (*int*) – Número de inversores que forman el anillo.
- **loc** (*str*) – Parámetro 'loc' de la primera LUT del anillo (ver clase 'Lut'). En un anillo estándar esta LUT siempre corresponde a la puerta AND inicial. Solo se da la posición de la primera LUT porque los anillos siempre se construyen ocupando todos los 'bel' de una celda antes de pasar a la celda inmediatamente superior (de modo que, dadas las coordenadas del primer elemento, las demás están predeterminadas). La "celda inmediatamente superior" es X+1 si X es par, Y+1 si X es impar.
- **bel** (*char o lista de char*) – Lista de parámetros 'bel' para cada LUT del anillo (ver clase 'Lut'). Cada elemento de la lista se aplica en orden correlativo al elemento del anillo (la primera restricción se aplica al AND inicial, la segunda al primer inversor, etc). Pueden darse menos restricciones que inversores forman el anillo, pero entonces quedarán LUT sin fijar. Notar además que el valor de esta restricción es excluyente entre LUT que forman parte de la misma celda: cada celda tiene cuatro posibles posiciones A, B, C y D, y dos LUT no pueden ocupar el mismo espacio. Esta función no avisa de esta violación: el usuario es responsable de que los valores 'bel' introducidos sean todos distintos entre sí en grupos de cuatro. En otro caso, el diseño fallará. Además de una lista de caracteres, esta opción admite un único caracter, pero entonces solo se restringirá la AND inicial. Ver opción 'bel' de la clase 'Lut'. Notar que el número de elementos total de un anillo "StdRing" es igual a N_inv+1, siendo el primero siempre una LUT2 (AND).
- **pin** (*str o lista de str*) – Lista de parámetros 'pin' de cada LUT que forma parte del anillo (ver clase 'Lut'). Cada elemento de la lista se aplica en orden correlativo al elemento del anillo (la primera restricción se aplica al AND inicial, la segunda al primer inversor, etc). Pueden darse menos restricciones que inversores forman el anillo, pero entonces quedarán LUT sin mapear. Notar que el mapeo debe ser coherente con el tipo de LUT que se está fijando, y en particular el AND inicial siempre es de tipo LUT2 (i.e., solo se pueden fijar los pines 'I0' y 'I1'). Notar que el número de elementos total de un anillo "StdRing" es igual a N_inv+1, siendo el primero siempre una LUT2 (AND). Por definición, el puerto lógico 'I0' siempre es el que recoge la señal del elemento precedente en el bucle.
- **pdl** (*bool, opcional*) – Si 'True' se utilizan modelos LUT6 para los inversores, permitiendo utilizar 5 puertos para configurar el anillo mediante PDL. Si 'False' se utilizan modelos LUT1 para los inversores y LUT2 para el enable AND. Por defecto False.

N_inv

Número de inversores del anillo.

bel

Lista de restricciones BEL para cada elemento del anillo.

elements

Lista de objetos Lut que conforman el anillo. El primer elemento será Lut2, y los restantes *N_inv* elementos serán Lut1 o Lut6 en función de si la opción *pdl* es 'True'.

help()

Ayuda de la clase 'StdRing'.

loc

Parámetro LOC de la primera puerta del anillo.

name

Nombre del anillo instanciado.

pin

Lista de restricciones PIN para cada elemento del anillo.

fpga.ring_osc.**clog2**(*N*)

Numero de bits necesarios para especificar 'N' estados.

Parameters
N (*int*) – Número del cual calcular la función 'ceiling log'

Returns
Número entero más pequeño que es mayor o igual al resultado del logaritmo base 2 de *N*.

Return type
float

fpga.ring_osc.**load**(*file_name*)

'Wrapper' para la función 'load' del módulo *pickle*, que permite cargar un objeto guardado serializado de cualquier clase.

Parameters
file_name (*str*) – Nombre del archivo a cargar.

fpga.ring_osc.**sim_romatrix**(*N_rep=1*, *N_pdl=1*, *N_osc=1*, *std_rep=1*, *std_pdl=10*, *std_osc=100*)

Esta función proporciona un simulador naíf de una matriz de celdas; reproduce las medidas de una instancia para un número de repeticiones (N_rep), pdl (N_pdl) y celdas (N_osc), así como ajustar las desviaciones estándar de cada proceso. El comportamiento estándar es: std_rep<std_pdl<std_osc La función genera los valores aleatorios como una distribución normal, pero luego los escala para devolver siempre valores enteros positivos.

Parameters

- **N_rep** (*int, opcional*) – Número de repeticiones simuladas.
- **N_pdl** (*int, opcional*) – Número de PDL simulados.
- **N_osc** (*int, opcional*) – Número de celdas simuladas.
- **std_rep** (*float, opcional*) – Desviación estándar de una misma celda, para un mismo PDL entre medidas sucesivas.
- **std_pdl** (*float, opcional*) – Desviación estandar de una misma celda entre distintos PDL
- **std_osc** (*float, opcional*) – Desviación estándar entre distintas celdas.

CHAPTER

TWO

SETUP MODULE

CHAPTER

THREE

UTILS PACKAGE

class utils.**BarraProgreso**(*total*, *long_barra=40*, *caracter='#'*)

Bases: object

Esta clase instancia un objeto BarraProgreso.

Parameters

- **total** (*int*) – Número de iteraciones que hacen el 100% de la barra.
- **long_barra** (*int, opcional*) – Número de caracteres que componen la barra, por defecto 40.
- **caracter** (*char, opcional*) – Caracter que se utiliza para decorar la barra, por defecto '#'.

Return type
None

utils.**export_legend_plt**(*ax*, *name='legend.pdf'*, *pad_inches=0.1*, ***kwargs*)

Exporta la leyenda de la figura actualmente activa.

Esta función es útil para exportar la leyenda de un gráfico cuando se tienen múltiples gráficos que comparten la misma leyenda. Es importante tener en cuenta que se requiere haber creado una figura con 'plt.plot', la cual debe incluir una leyenda, y NO se debe haber ejecutado 'plt.show()'.

Parameters

- **ax** (*matplotlib.axes.Axes*) – Los ejes de la figura de los cuales se quiere extraer la leyenda. Si se utilizan comandos como 'plot', etc., se puede obtener los ejes activos del gráfico con ax=plt.gca().
- **name** (*str, opcional*) – Nombre de la imagen. Por defecto es "legend.pdf".
- **pad_inches** (*float, opcional*) – Esta opción se pasa directamente a la función "savefig". Por defecto es 0.1.
- **kwargs** (*dict*) – Las opciones introducidas aquí se pasan directamente a la función *ax.legend*.

Return type
None

utils.**run_in_parallel**(*func*, *args*)

Ejecuta la función en paralelo utilizando argumentos proporcionados.

Esta función toma un método 'func' y una lista de argumentos 'args', y repite la ejecución de la función en paralelo. El método 'func' puede recibir un número arbitrario y tipos de argumentos, pero serán pasados en orden según aparezcan en 'args'.

Parameters

- **func** (*function*) – Procedimiento que puede tomar un número arbitrario y tipos de argumentos, pero son específicos en su posición.
- **args** (*list*) – Lista que contiene una lista con los argumentos a pasar a 'func' en cada proceso.

Returns
Lista con los resultados de cada proceso.

Return type
list

> **Warning:** Esta función siempre debe ser ejecutada dentro de *if __name__=='__main__':*.

utils.**set_size_plt**(*ax*, *x=5.333333333333333*, *y=4*)

Esta función permite dibujar una figura fijando el tamaño del plot, y no de la figura completa. Toma los valores *x*, *y* que representan las dimensiones en dichos ejes de un plot, y construye la figura del tamaño que sea necesario para acomodar los ejes.

Parameters

- **ax** (*matplotlib.axes.Axes*) – Objeto que contiene la figura a redimensionar.
- **x** (*float, opcional*) – Nueva anchura, por defecto 16/3.
- **y** (*float, opcional*) – Nueva altura, por defecto 4.

Return type
None

3.1 Submodules

3.1.1 utils.bool module

class utils.bool.**LinearSystECC**(*dim=4*, *length=7*, *G=False*, *H=False*)

Bases: object

Clase para manipular sistemas de códigos de corrección de errores lineales.

Esta clase proporciona funcionalidades para la generación de matrices G y H, codificación y decodificación de vectores para códigos de corrección de errores lineales en un cuerpo finito.

Parameters

- **dim** (*int, opcional*) – Dimensión de la matriz. Por defecto 4.
- **length** (*int, opcional*) – Longitud de la matriz. Por defecto 7.
- **G** (*list of list of bool, opcional*) – Matriz booleana G para operaciones de generación de código. Por defecto False.
- **H** (*list of list of bool, opcional*) – Matriz booleana H para operaciones de generación de código. Por defecto False.

Raises
ValueError – Si la matriz generadora y la matriz de paridad no cumplen GH^=0.

Notes

Esta clase permite realizar operaciones en un cuerpo finito para códigos de corrección de errores lineales. Los métodos 'encode' y 'decode' permiten la codificación y decodificación de vectores respectivamente, usando las matrices generadora y de paridad.

decode(*vector*)

Decodifica un código de entrada de 'length' bits en un mensaje de 'dim' bits utilizando un método de mínima distancia de Hamming.

Este método es absolutamente ineficiente y no debería usarse con dimensiones mayores que 9.

Parameters
vector (*list of bool*) – Código booleano de entrada a decodificar.

Returns
Mensaje decodificado.

Return type
list of bool

Raises
ValueError – Si la longitud del vector de entrada no coincide con 'length'. Si hay demasiados errores para decodificar el vector de manera fiable.

encode(*vector*)

Codifica un mensaje de entrada de 'dim' bits en un código de 'length' bits.

Parameters
vector (*list of bool*) – Mensaje booleano de entrada a codificar.

Returns
Mensaje booleano codificado.

Return type
numpy.ndarray

Raises
ValueError – Si la longitud del vector de entrada no coincide con 'dim'.

utils.bool.**gauss_elimination_gf2**(*coeff_matrix_in*, *const_vector_in*)

Resuelve un sistema de ecuaciones booleanas en un cuerpo finito con las operaciones XOR ("+") y AND ("*").

Parameters

- **coeff_matrix_in** (*list of list of bool*) – Matriz booleana que representa los coeficientes del sistema de ecuaciones.
- **const_vector_in** (*list of bool*) – Vector booleano que representa el término independiente de cada ecuación.

Returns
Matriz booleana triangular.

Return type
numpy.ndarray

utils.bool.**hamming**(*in1*, *in2*)

Calcula la distancia de Hamming (medida en bits) entre dos entradas.

Parameters

- **in1** (*list of bool*) – Primer vector booleano de entrada.

- **in2** (*list of bool*) – Segundo vector booleano de entrada.

Returns
Distancia de Hamming entre las dos entradas (medida en bits).

Return type
int

utils.bool.**invert_matrix_gf2**(*matrix_in*)

Devuelve la inversa de una matriz booleana en un cuerpo finito con las operaciones XOR ("+") y AND ("*").

Parameters
matrix_in (*list of list of bool*) – Matriz booleana a invertir.

Returns
Matriz booleana inversa.

Return type
numpy.ndarray

utils.bool.**matrix_multiply_gf2**(*matrix1_in*, *matrix2_in*)

Multiplica un par de matrices booleanas en un cuerpo finito con las operaciones XOR ("+") y AND ("*").

Esta función puede utilizarse también para multiplicar vectores por matrices, gestionando automáticamente la disposición del vector: fila si va delante de la matriz, columna si va detrás. Así, si 'matrix1_in' y 'matrix2_in' son dos vectores, el resultado será el producto vectorial.

Parameters

- **matrix1_in** (*list of list of bool*) – Primera matriz booleana.
- **matrix2_in** (*list of list of bool*) – Segunda matriz booleana.

Returns
Producto de las matrices o el producto vectorial en gf(2).

Return type
numpy.ndarray

3.1.2 utils.stats module

utils.stats.**Dks_montecarlo_discrete**(*model*, *fit*, *N*, *verbose=True*, ***kwargs*)

Calcula la distribución del estadístico KS de un modelo discreto en comparación con una distribución teórica.

Parameters

- **model** (*callable*) – Función aleatoria a contrastar. El primer parámetro que debe aceptar es 'N', seguido de un número arbitrario de parámetros que son pasados mediante 'kwargs'. Devuelve un array de valores aleatorios.
- **fit** (*list of float*) – Lista de valores que representa la distribución de probabilidad contra la cual se calcula la distribución de KS.
- **N** (*int*) – Número de veces que se repite el cálculo de Dks.
- **verbose** (*bool, optional*) – Si es True, imprime el progreso. Por defecto es True.
- **kwargs** (*dict*) – Parámetros a pasar a 'model'.

Returns
Lista de valores Dks.

Return type
list of float

Notes

Esta función calcula la distribución del estadístico KS de un modelo 'model' que produce 'N' valores aleatorios frente a una curva teórica 'fit', representada como una lista de valores. Devuelve una lista de estos índices KS.

El método 'model' admite una cantidad arbitraria de parámetros pasados mediante 'kwargs'.

utils.stats.**bin_rv_cont**(*rv_continuous*, *bins_in*, **args*)

Calcula la probabilidad acumulada de un objeto 'rv_continuous' de scipy en los extremos dados por 'bins'.

Parameters

- **rv_continuous** (*scipy.stats.rv_continuous*) – Objeto 'rv_continuous' de scipy.
- **bins_in** (*array_like*) – Bineado del eje de abscisas. Debe ser una lista de 'N_bins+1' elementos, conteniendo los extremos de cada bin.
- **args** (*list, optional*) – Parámetros extra para pasar a 'rv_continuous.cdf', distintos del primer argumento ('x').

Returns
Arreglo de tamaño 'N_bin' elementos que contiene la probabilidad acumulada en los extremos dados por 'bins'. El arreglo de salida está normalizado respecto a la suma de sus elementos.

Return type
numpy.ndarray

Notes

Esta función toma un objeto 'rv_continuous' de scipy y una lista 'bins_in' que contiene el bineado del eje de abscisas. Calcula la probabilidad acumulada en los extremos dados por 'bins' utilizando 'rv_continuous.cdf' y devuelve un numpy array normalizado con las probabilidades acumuladas.

El argumento 'args' se utiliza para pasar parámetros extra distintos al primero ('x') a 'rv_continuous.cdf'.

utils.stats.**bin_rv_discrete**(*rv_discrete*, *bins_in*, **args*)

Calcula la probabilidad acumulada de un objeto 'rv_discrete' de scipy en los extremos dados por 'bins'.

Parameters

- **rv_discrete** (*scipy.stats.rv_discrete*) – Objeto 'rv_discrete' de scipy.
- **bins_in** (*array_like*) – Bineado del eje de abscisas. Debe ser una lista de 'N_bins+1' elementos, conteniendo los extremos de cada bin.
- **args** (*list, optional*) – Parámetros extra para pasar a 'rv_discrete', distintos del primer argumento ('k').

Returns
Arreglo de tamaño 'N_bin' elementos que contiene la probabilidad acumulada en los extremos dados por 'bins'. El arreglo de salida está normalizado respecto a la suma de sus elementos.

Return type
numpy.ndarray

Notes

Esta función toma un objeto 'rv_discrete' de scipy y una lista 'bins_in' que contiene el bineado del eje de abscisas. Calcula la probabilidad acumulada en los extremos dados por 'bins' y devuelve un numpy array normalizado con las probabilidades acumuladas.

El argumento 'args' se utiliza para pasar parámetros extra distintos del primero ('k') a 'rv_discrete'.

utils.stats.**chisq_gof_discrete**(*obs_data*, *model*, *alpha=0.05*, *plothist=False*, ***kwargs*)

Realiza un test chi^2 de bondad del ajuste entre los datos observados y un modelo estocástico discreto (una función pmf -probability mass function-) que hipotéticamente genera dichos datos.

Parameters

- **obs_data** (*array_like*) – Vector de datos observados.
- **model** (*callable*) – Función pmf (probability mass function) que representa el modelo estocástico discreto. El primer argumento siempre debe ser la variable independiente (aleatoria).
- **alpha** (*float, optional*) – Nivel de significancia para el test chi^2. Por defecto es 0.05.
- **plothist** (*bool, optional*) – Indica si se debe graficar el histograma de los datos. Por defecto es False.
- **kwargs** (*dict*) – Opciones adicionales para pasar parámetros a la función pmf introducida como modelo.

Returns

Un diccionario con los campos: - 'chisq': Valor del estadístico chi^2. - 'pvalue': Valor tal que la probabilidad de obtener 'chisq' es al menos pvalue. - 'dof': Número de grados de libertad de la distribución chi^2. - 'testpass': Booleano que indica si el test ha sido superado en función de 'alpha'.

Return type

dict

Notes

Esta función calcula un test chi^2 de bondad del ajuste entre los datos observados y un modelo estocástico discreto proporcionado como una función pmf. Se genera un histograma de los datos y se calculan los valores esperados de probabilidad para cada bin. Luego se aplica la función 'chisquare' de scipy para realizar el test chi^2.

El parámetro 'kwargs' se puede utilizar para pasar parámetros adicionales a la función pmf del modelo.

utils.stats.**fit_rv_cont**(*data*, *rv_cont*, *bins=10*, *alpha=0.05*, *plot=False*)

Encuentra la distribución 'rv_continuous' que mejor ajusta un conjunto de datos 'data' y calcula el 'valor p' correspondiente a una significancia 'alpha'.

Parameters

- **data** (*list of float*) – Vector con los datos a ajustar.
- **rv_cont** (*scipy.stats.rv_continuous*) – Objeto 'scipy.stats.rv_continuous' que define una distribución de probabilidad continua que se ajustará a los datos 'data'.
- **bins** (*int, optional*) – Número de cajas para el histograma de 'data'. Por defecto es 10.
- **alpha** (*float, optional*) – Valor de significancia 'alpha' para el cual se calcula el valor p. Por defecto es 0.05.

- **plot** (*bool, optional*) – Si es True, pinta el histograma y superpone la curva encontrada. La función no ejecuta 'show()', de forma que el usuario puede recabar la figura externamente con 'gca()' y 'gcf()', y editarla antes de representarla. Por defecto es False.

Returns

Una lista que contiene cuatro elementos: - El primero es una lista de los parámetros que mejor ajustan la distribución (en el mismo orden en que se definen en la función 'rv_cont.pdf'). - El segundo es el valor 'valo p' correspondiente a una significancia 'alpha'. - El tercero es un nparray con el histograma. - El cuarto es un nparray con el bineado de los datos de entrada.

Return type

list

Notes

Esta función es un wrapper para encontrar la distribución 'rv_continuous' que mejor ajusta un conjunto de valores 'data' utilizando la máxima verosimilitud. Calcula el 'valor p' correspondiente a una significancia 'alpha'. Resulta útil para distribuciones obtenidas por simulación, que carecen de una función densidad teórica.

utils.stats.**get_area_fraction**(*hist*, *p=0.9*)

Encuentra los extremos 'xmin' y 'xmax' de una lista 'hist' de números (no necesariamente normalizada) de tal manera que el intervalo 'hist[xmin:xmax]' contenga al menos una fracción 'p' de la suma acumulada de los elementos de la lista.

Parameters

- **hist** (*list or array_like*) – Lista de números.
- **p** (*float, optional*) – Fracción mínima deseada. Por defecto es 0.9.

Returns

- **xmin** (*int*) – Índice del extremo izquierdo del intervalo.
- **xmax** (*int*) – Índice del extremo derecho del intervalo.

Notes

Esta función busca los extremos 'xmin' y 'xmax' en la lista 'hist' de números, de tal manera que el intervalo 'hist[xmin:xmax]' contenga al menos una fracción 'p' de la suma acumulada de los elementos de la lista. La búsqueda comienza desde el índice correspondiente al valor máximo de la lista y se desplaza simétricamente hacia ambos extremos.

utils.stats.**get_hist_smooth**(*data*, *p=0.9*, *x0=nan*, *x1=nan*)

Genera un histograma suavizado a partir de un conjunto de datos 'data'.

Parameters

- **data** (*array_like*) – Conjunto de datos para representar en forma de histograma.
- **p** (*float, optional*) – Fracción (tanto por uno) de área bajo la cual se busca que el histograma no presente máximos relativos. Por defecto es 0.9.
- **x0** (*float, optional*) – Extremo inferior del rango del histograma. Por defecto es NaN (se calcula automáticamente a partir de 'data').
- **x1** (*float, optional*) – Extremo superior del rango del histograma. Por defecto es NaN (se calcula automáticamente a partir de 'data').

Returns
Histograma suavizado generado.

Return type
numpy.ndarray

Notes

Esta función devuelve un histograma con el número de bins adecuado para que no presente máximos locales, lo cual resulta en un aspecto más suave. El parámetro 'p' representa la fracción de área bajo la cual se busca que el histograma no presente máximos relativos, medido simétricamente desde el punto máximo del histograma.

Los parámetros 'x0' y 'x1' son opcionales y se utilizan para definir el rango del histograma; si no se proporcionan, se calculan automáticamente a partir de los valores mínimo y máximo de 'data'.

`utils.stats.`**`intervalo_int`**(*inf*, *sup*)

Esta función auxiliar devuelve los extremos de números enteros dado un intervalo real.

Parameters

- **inf** (*float*) – Extremo inferior del intervalo.
- **sup** (*float*) – Extremo superior del intervalo.

Returns
Rango de enteros entre los extremos *inf* y *sup*.

Return type
list

3.1.3 utils.tensor module

Este módulo permite definir nombres para los ejes de un numpyarray.

class `utils.tensor.`**`Tensor`**(*array*, *axis=False*)

Bases: `object`

Un objeto que es esencialmente un nparray junto con una variable 'self.axis' que permite indexar los ejes del array mediante un nombre natural, en lugar de un índice entero.

Parameters

- **array** (*numpy.ndarray*) – Arreglo que contiene los datos y la forma del tensor.
- **axis** (*list of strings, optional*) – Nombres de los ejes. El orden en que se proporcionan los nombres se correlaciona con el correspondiente eje del parámetro 'array'. Si no se introduce una lista de nombres en esta variable, se asignarán por defecto como nombres el índice de cada eje (empezando en '0').

array

Arreglo que contiene los datos y la forma del tensor.

Type
numpy.ndarray

shape

Forma del tensor (igual que en un nparray).

Type
tuple

`axis`

Nombres de los ejes. El i-ésimo elemento contiene el nombre del i-ésimo eje. Si no se proporciona, se asignan nombres predeterminados en forma de índices.

Type
list of strings

`item`(***axes*)

Esta función toma por argumento el nombre de cada eje igualado a un índice, y devuelve el valor guardado en esa coordenada.

Parameters
axes (*dict*) – Argumentos clave-valor donde las claves son los nombres de los ejes y los valores son los índices para esos ejes.

Returns
Valor almacenado en las coordenadas especificadas.

Return type
object

`mean`(*axe*)

Devuelve un Tensor comprimido en el eje 'axe', suprimiéndolo y calculando el promedio a lo largo de dicho eje.

Parameters
axe (*int*) – Índice del eje a comprimir y calcular el promedio.

Returns
Tensor con el eje 'axe' suprimido y cada punto contiene el promedio a lo largo de ese eje.

Return type
Tensor

`save`(*file_name*, *fmt='%.18e'*)

Guarda el Tensor en un archivo de texto.

Parameters

- **file_name** (*str*) – Nombre del archivo donde se guardará el Tensor.
- **fmt** (*str, optional*) – Formato numérico de los datos. Sigue el mismo formato que el parámetro 'fmt' de numpy.savetxt.

Return type
None

`size`(*axis*)

Esta función toma el nombre de un eje y devuelve su dimensión.

Parameters
axis (*str*) – Nombre del eje del que se desea obtener la dimensión.

Returns
Dimensión del eje especificado.

Return type
int

slice(***axes*)

Toma una serie de ejes (por nombre), igualados a una porción 'slice'. Esta porción puede ser un entero, una lista de dos enteros, o una lista de tres enteros, y devuelve un Tensor reducido al 'slice' indicado (usando la misma notación que un slice estándar en Python: [x] es la x-ésima entrada, [x,y] son todas entradas en [x,y), y [x,y,z] son todas las entradas en [x,y) tomadas en saltos de z). Esta función es en realidad un wrapper a 'take', usando la nomenclatura de Python para hacer slicing.

Parameters
axes (*dict*) – Argumentos clave-valor donde las claves son los nombres de los ejes y los valores son objetos 'slice' o listas de enteros para realizar la porción en esos ejes.

Returns
Tensor reducido al 'slice' indicado a lo largo de los ejes especificados.

Return type
Tensor

squeeze()

Elimina todos los ejes del tensor cuya dimensión sea igual a '1'.

Returns
Tensor resultante después de eliminar los ejes con dimensión '1'.

Return type
Tensor

std(*axe*)

Devuelve un Tensor comprimido en el eje 'axe', suprimiéndolo y calculando la desviación estándar a lo largo de dicho eje.

Parameters
axe (*int*) – Índice del eje a comprimir y calcular la desviación estándar.

Returns
Tensor con el eje 'axe' suprimido y cada punto contiene la desviación estándar a lo largo de ese eje.

Return type
Tensor

swap(*axe1*, *axe2*)

Devuelve un Tensor con el mismo "array" pero los ejes "axe1" y "axe2" intercambiados.

Parameters

- **axe1** (*int*) – Índice del primer eje a intercambiar.
- **axe2** (*int*) – Índice del segundo eje a intercambiar.

Returns
Tensor con los ejes "axe1" y "axe2" intercambiados.

Return type
Tensor

take(***axes*)

Toma una serie de ejes (por nombre), igualados a una lista de índices, y devuelve un Tensor reducido a los índices seleccionados.

Parameters
axes (*dict*) – Argumentos clave-valor donde las claves son los nombres de los ejes y los valores son listas de índices a lo largo de esos ejes.

Returns
Tensor reducido a los índices seleccionados a lo largo de los ejes especificados.

Return type
Tensor

utils.tensor.**append**(*axe*, **args*)

Concatena dos objetos 'Tensor' a lo largo de un eje especificado por su nombre.

Parameters

- **axe** (*str*) – Nombre del eje a lo largo del cual se realizará la concatenación.
- **args** (Tensor) – Tensores que se concatenarán. Se deben proporcionar al menos dos tensores de entrada. Todos los tensores deben tener un parámetro 'Tensor.axis' idéntico y las mismas dimensiones en todos los ejes excepto posiblemente en 'axe'.

Returns
Nuevo Tensor que consiste en la concatenación de los argumentos en el orden proporcionado.

Return type
Tensor

Raises
ValueError – Si no se cumplen los requisitos de entrada (al menos dos tensores y 'axe').

utils.tensor.**load_tensor**(*file_name*)

Devuelve un Tensor almacenado en un archivo de texto.

Parameters
file_name (*str*) – Nombre del archivo que contiene el Tensor.

Returns
Tensor cargado desde el archivo de texto.

Return type
Tensor

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

u