myfpga

Release 2.94

Guillermo Díez-Señorans

Dec 13, 2023

CONTENTS:

CHAPTER

ONE

MYFPGA PACKAGE

1.1 Submodules

1.2 myfpga.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).

1.3 myfpga.interfaz_pcps module

myfpga.interfaz_pcps.**bitstr_to_bytestr**(*bitstr_in*, *bitstr_width*)

Toma una lista de bits (bitstr_in) de tamaño 'bitstr_width' y la convierte a una lista de bytes del tamaño mínimo necesario para alojar la entrada.

myfpga.interfaz_pcps.**bitstr_to_int**(*entrada*)

Convierte una lista de bits en su correspondiente numero decimal (entero)

myfpga.interfaz_pcps.**bytestr_to_bitstr**(*bytestr*, *bitstr_width*)

Toma una lista de bytes y los aloja en una lista de bits de tamaño 'bitstr_width'. Si 'bitstr_width' es insuficiente, la lista de bits se truncará.

myfpga.interfaz_pcps.**bytestr_to_int**(*entrada*)

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

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

Esta funcion dispara un ciclo RDY->CALC->PRINT->RDY a una fsm conectada a traves de 'serialport' y devuelve un bitstr de tamaño 'buffer_out_width'

myfpga.interfaz_pcps.**int_to_bitstr**(*entrada*)

Esta funcion convierte un numero 'entrada' en un bitstr (i.e. en su representacion en base 2). Como siempre, el numero mas a la dcha del bitstr (result[0]) es el menos significativo

myfpga.interfaz_pcps.**int_to_bytestr**(*entrada*)

Esta funcion convierte un numero 'entrada' en un bytestr (i.e. en su representacion en base 256). Como siempre, el numero mas a la dcha del bytestr (result[0]) es el menos significativo

myfpga.interfaz_pcps.**print_bitstr**(*bitstr_in*)

myfpga.interfaz_pcps.**print_bytestr**(*bytestr_in*)

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

Recibe 'bytestr_size' bytes a traves del puerto serie 'serialport'

myfpga.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

myfpga.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

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

Esta funcion toma un array_old de tamano arbitrario y transfiere los caracteres a un array de tamano nbits, empezando por el caracter menos significativo (a la dcha.). Si nbits no es suficiente para contener todo el array_old, la salida estara truncada. Si nbits es excesivo para contener array_old, los huecos que sobren se rellenan con ceros.

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

Esta funcion dispara un ciclo RDY->SCAN->RDY a una fsm conectada a traves de 'serialport' y sube un bitstr de tamaño 'buffer_in_width'

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

Envía 'bytestr_out' bytes a traves del puerto serie 'serialport'

myfpga.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

1.4 myfpga.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 myfpga.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.

1.4.1 Variables:

osc_coord : coordenadas de los osciladores.

help()

Ayuda de la clase 'Dominio'.

class myfpga.ring_osc.**GaloisMatrix**(*N_inv=3*, *dominios=<myfpga.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.

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()'.

1.4.2 Parámetros:

out_name
[<string> (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.

1.4.3 Parámetros:

projname
[<string>] Nombre del proyecto de Vivado.

projdir
[<string>] Directorio donde se creará el proyecto de Vivado y las fuentes (por defecto el directorio de trabajo actual).

njobs
[<int>] Número de núcleos que utiilizará Vivado paralelamente para la síntesis/implementación.

debug
[<bool>] 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>] 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
[<string>] Placa de desarrollo utilizada en el proyecyo. Las opciones soportadas son: 'pynqz2', 'zybo', 'cmoda7_15t' o 'cmoda7_35t'.

qspi
[<bool>] 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>] 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, por defecto False>] 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>] Esta opción especifica la anchura del canal de datos PS<–>PL.

buffer_out_width
[<int>] 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'.

1.4.4 Parámetros:

puerto
[<string>] Esta opción especifica el puerto serie al que se conecta la FPGA.

osc
[<int o lista de int>] Lista de osciladores a medir.

pdl
[<int o lista de 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 myfpga.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'.

Un anillo 'GaloisRing' consta de N_inv+2 elementos:
. Inversor inicial. . N_inv-1 elementos inversores/XNOR. . Inversor de salida. . flip-flop de muestreo.

help()

Ayuda de la clase 'GaloisRing'.

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

Bases: object

Objeto que contiene una matriz de osciladores de anillo estándar.

1.4.5 Parámetros:

N_inv
[<int>] Número de inversores de cada oscilador.

dominios
[<objeto 'Dominio' o lista de estos>] 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
[<caracter o lista de caracteres>] 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
[<cadena de caracteres o lista de cadenas>] 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, por defecto False>] 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.

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()'.

1.4.5.1 Parámetros:

out_name
[<string> (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.

1.4.5.2 Parámetros:

projname
[<string>] Nombre del proyecto de Vivado.

projdir
[<string>] Directorio donde se creará el proyecto de Vivado y las fuentes (por defecto el directorio de trabajo actual).

njobs
[<int>] Número de núcleos que utiilizará Vivado paralelamente para la síntesis/implementación.

debug
[<bool>] 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>] 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
[<string>] Placa de desarrollo utilizada en el proyecyo. Las opciones soportadas son: 'pynqz2', 'zybo', 'cmoda7_15t' o 'cmoda7_35t'.

qspi
[<bool>] 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>] 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, por defecto False>] 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(dw)
[<int>] Esta opción especifica la anchura del canal de datos PS<–>PL.

buffer_out_width(bow)
[<int>] Esta opción especifica la anchura de la palabra de respuesta (i.e., de la medida).

f_clock
[<int>] 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'.

1.4.5.3 Parámetros:

puerto
[<string>] Esta opción especifica el puerto serie al que se conecta la FPGA.

osc
[<int o lista de int>] Lista de osciladores a medir.

pdl
[<int o lista de 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 17, i.e., 2^17 = 131072 ciclos).

f_ref = <real>
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>] 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 myfpga.ring_osc.**StdRing**(*name*, *N_inv*, *loc*, *bel=''*, *pin=''*, *pdl=False*)

Bases: object

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'.

Un anillo 'StdRing' consta de N_inv+1 elementos:
. Puerta AND. . N_inv inversores.

help()

Ayuda de la clase 'StdRing'.

myfpga.ring_osc.**clog2**(*N*)

Numero de bits necesarios para especificar 'N' estados.

myfpga.ring_osc.**load**(*file_name*)

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

myfpga.ring_osc.**np_normal**()

normal(loc=0.0, scale=1.0, size=None)

Draw random samples from a normal (Gaussian) distribution.

The probability density function of the normal distribution, first derived by De Moivre and 200 years later by both Gauss and Laplace independently [2]_, is often called the bell curve because of its characteristic shape (see the example below).

The normal distributions occurs often in nature. For example, it describes the commonly occurring distribution of samples influenced by a large number of tiny, random disturbances, each with its own unique distribution [2]_.

Note: New code should use the *~numpy.random.Generator.normal* method of a *~numpy.random.Generator* instance instead; please see the random-quick-start.

1.4.6 Parameters

loc
: [float or array_like of floats] Mean ("centre") of the distribution.

scale
: [float or array_like of floats] Standard deviation (spread or "width") of the distribution. Must be non-negative.

size
: [int or tuple of ints, optional] Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. If size is `None` (default), a single value is returned if `loc` and `scale` are both scalars. Otherwise, `np.broadcast(loc, scale).size` samples are drawn.

1.4.7 Returns

out
: [ndarray or scalar] Drawn samples from the parameterized normal distribution.

1.4.8 See Also

scipy.stats.norm
: [probability density function, distribution or] cumulative density function, etc.

random.Generator.normal: which should be used for new code.

1.4.9 Notes

The probability density for the Gaussian distribution is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean and σ the standard deviation. The square of the standard deviation, σ^2, is called the variance.

The function has its peak at the mean, and its "spread" increases with the standard deviation (the function reaches 0.607 times its maximum at $x + \sigma$ and $x - \sigma$ [2]_). This implies that normal is more likely to return samples lying close to the mean, rather than those far away.

1.4.10 References

1.4.11 Examples

Draw samples from the distribution:

```
>>> mu, sigma = 0, 0.1 # mean and standard deviation
>>> s = np.random.normal(mu, sigma, 1000)
```

Verify the mean and the variance:

```
>>> abs(mu - np.mean(s))
0.0  # may vary
```

```
>>> abs(sigma - np.std(s, ddof=1))
0.1  # may vary
```

Display the histogram of the samples, along with the probability density function:

```
>>> import matplotlib.pyplot as plt
>>> count, bins, ignored = plt.hist(s, 30, density=True)
>>> plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
...                np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
...          linewidth=2, color='r')
>>> plt.show()
```

Two-by-four array of samples from the normal distribution with mean 3 and standard deviation 2.5:

```
>>> np.random.normal(3, 2.5, size=(2, 4))
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],   # random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]])  # random
```

myfpga.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.

1.4.12 Parámetros:

N_rep
[<int, opcional, por defecto 1>] Número de repeticiones simuladas.

N_pdl
[<int, opcional, por defecto 1>] Número de PDL simulados.

N_osc
[<int, opcional, por defecto 1>] Número de celdas simuladas.

std_rep
[<float, opcional, por defecto 1.0>] Desviación estándar de una misma celda, para un mismo PDL entre medidas sucesivas.

std_pdl
[<float, opcional, por defecto 10.0>] Desviación estandar de una misma celda entre distintos PDL

std_osc
[<float, opcional, por defecto 100.0>] Desviación estándar entre distintas celdas.

1.5 myfpga.sim_lut module

class myfpga.sim_lut.**LUT3**(*config_ram*, *func_pot*, *O*, *I0='x'*, *I1='x'*, *I2='x'*, *umbral=0.5*, *mu=0*, *sigma=0*)

Bases: object

class myfpga.sim_lut.**MODULO**(*luts*)

Bases: object

probe(*name*)

run(*t0=0*, *t1=100*, *dt=1*, *waveform_in=False*, *out=False*)

setw(*name*, *value*)

step()

class myfpga.sim_lut.**WAVEFORM**(*x=False*, *names=[]*, *waves=[]*)

Bases: object

myfpga.sim_lut.**delayLineModel**(*expected_value*, *old_value*, *func_pot*, *mu=0*, *sigma=0*)

myfpga.sim_lut.**normal**(*loc=0.0*, *scale=1.0*, *size=None*)

Draw random samples from a normal (Gaussian) distribution.

The probability density function of the normal distribution, first derived by De Moivre and 200 years later by both Gauss and Laplace independently [2]_, is often called the bell curve because of its characteristic shape (see the example below).

The normal distributions occurs often in nature. For example, it describes the commonly occurring distribution of samples influenced by a large number of tiny, random disturbances, each with its own unique distribution [2]_.

Note: New code should use the *~numpy.random.Generator.normal* method of a *~numpy.random.Generator* instance instead; please see the random-quick-start.

1.5.1 Parameters

loc
[float or array_like of floats] Mean (“centre”) of the distribution.

scale
[float or array_like of floats] Standard deviation (spread or “width”) of the distribution. Must be non-negative.

size
[int or tuple of ints, optional] Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. If size is `None` (default), a single value is returned if `loc` and `scale` are both scalars. Otherwise, `np.broadcast(loc, scale).size` samples are drawn.

1.5.2 Returns

out
[ndarray or scalar] Drawn samples from the parameterized normal distribution.

1.5.3 See Also

scipy.stats.norm
[probability density function, distribution or] cumulative density function, etc.

random.Generator.normal: which should be used for new code.

1.5.4 Notes

The probability density for the Gaussian distribution is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean and σ the standard deviation. The square of the standard deviation, σ^2, is called the variance.

The function has its peak at the mean, and its "spread" increases with the standard deviation (the function reaches 0.607 times its maximum at $x + \sigma$ and $x - \sigma$ [2]_). This implies that normal is more likely to return samples lying close to the mean, rather than those far away.

1.5.5 References

1.5.6 Examples

Draw samples from the distribution:

```
>>> mu, sigma = 0, 0.1 # mean and standard deviation
>>> s = np.random.normal(mu, sigma, 1000)
```

Verify the mean and the variance:

```
>>> abs(mu - np.mean(s))
0.0  # may vary
```

```
>>> abs(sigma - np.std(s, ddof=1))
0.1  # may vary
```

Display the histogram of the samples, along with the probability density function:

```
>>> import matplotlib.pyplot as plt
>>> count, bins, ignored = plt.hist(s, 30, density=True)
>>> plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
...                np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
...          linewidth=2, color='r')
>>> plt.show()
```

Two-by-four array of samples from the normal distribution with mean 3 and standard deviation 2.5:

```
>>> np.random.normal(3, 2.5, size=(2, 4))
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],   # random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]])  # random
```

myfpga.sim_lut.**numDerivative**(*func*, *x*, *eps=1e-10*)

myfpga.sim_lut.**random**(*size=None*)

Return random floats in the half-open interval [0.0, 1.0). Alias for *random_sample* to ease forward-porting to the new random API.

1.6 Module contents

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

Bases: object

Modelo de FPGA-FF.

impl()

Esta función devuelve un 'string' que contiene el código en lenguaje Verilog necesario para implementar el FF inicializado.

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

Bases: object

Modelo de FPGA-LUT de 1 entrada.

impl()

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

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

Bases: object

Modelo de FPGA-LUT de 2 entradas.

impl()

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

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

Bases: object

Modelo de FPGA-LUT de 3 entradas.

impl()

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

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

Bases: object

Modelo de FPGA-LUT de 4 entradas.

impl()

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

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

Bases: object

Modelo de FPGA-LUT de 6 entradas.

impl()

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

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m