Filtros FIR en FPGA

Designing from scratch

Ing. David Caruso

Presentación

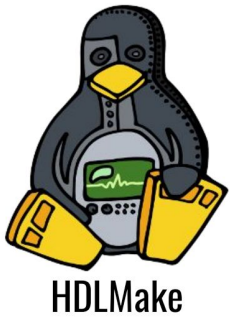
Ing. David Caruso

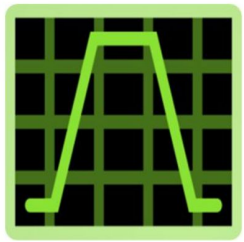
- Ingeniero en Electrónica egresado de la UTN-FRA en 2013
- 2006-2007: Desarrollo de hardware y software (independiente)
- 2007: Becario Laboratorio de Ingeniería, Investigación e Innovación Tecnológica (UTN-FRA)
- 2008-2012: R&D en Lab. de Desarrollo Electrónico con Software Libre (INTI)
  - *Linux, FPGA, Microcontroladores, Scripts*
- 2012-2015: Jefe de Laboratorio de Circuitos Electrónicos Avanzados (INTI)
  - *Coordinación, Diseño de PCB de alta velocidad, Linux, FPGA, Microcontroladores, Scripts*
- 2012-2015: Ayudante Análisis Matemático II (UTN-FRA)
- 2012-2015: Coordinador del Grupo de Robótica (UTN-FRA)
- 2015-hoy: R&D Electronic Engineer and Electronic Production Engineer (Satellogic)
  - *FPGA, Cortex-M, DSP, Image Processing, Complex PCB design, Linux*

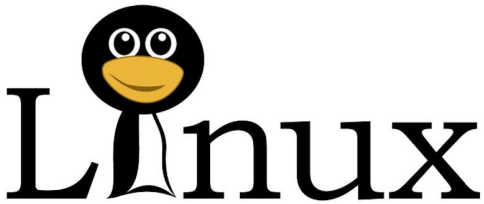
mail: david@satellogic.com / carusodvd@gmail.com

Introducción

The big picture



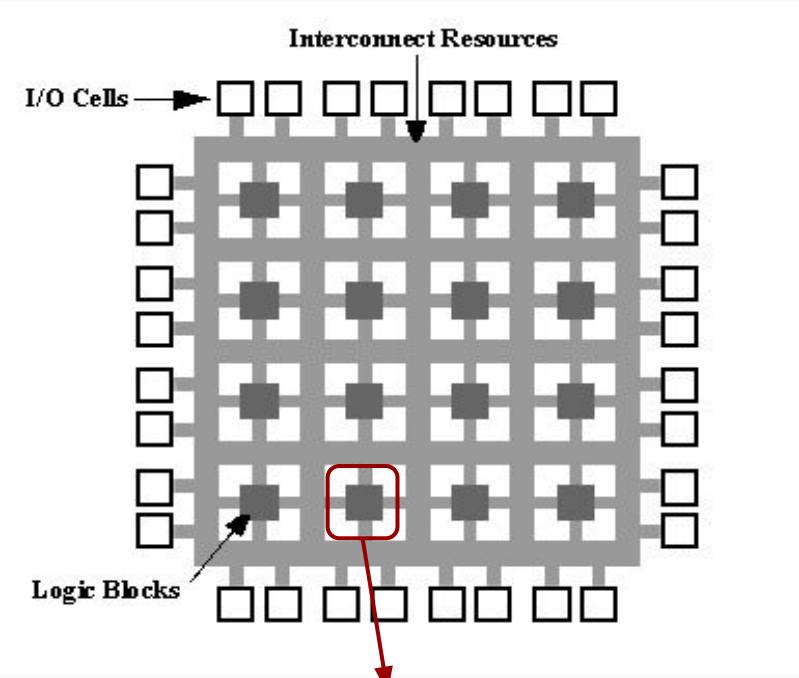


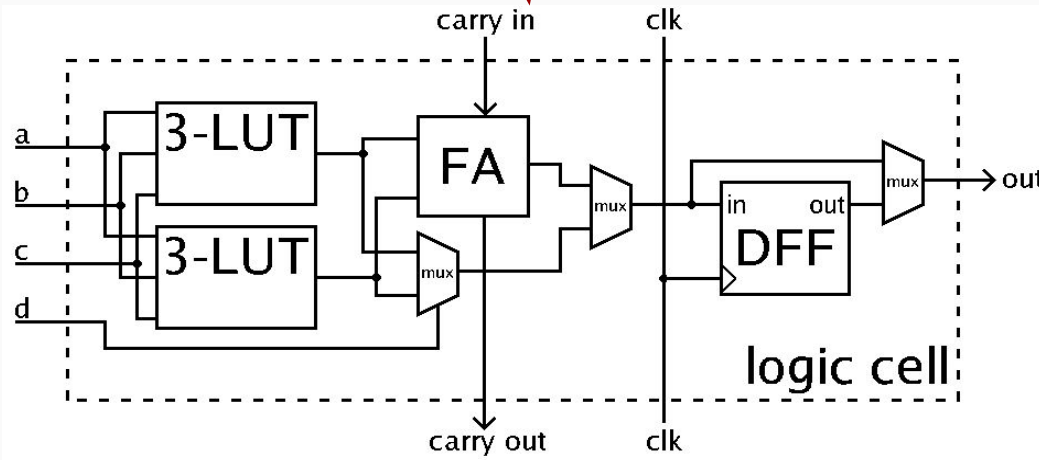


FPGA: Qué es?

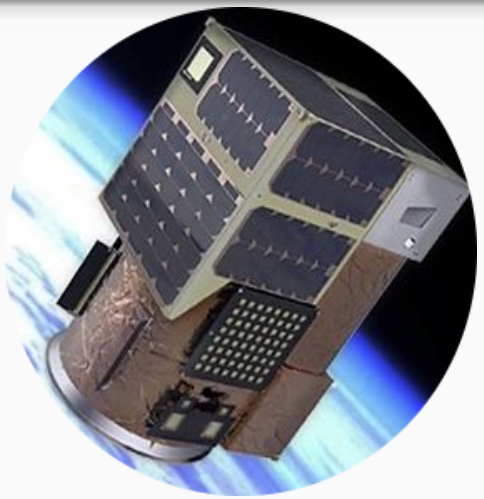
Es un dispositivo digital, que tiene una gran cantidad de circuitos dispuestos en una matriz de interconexión.
Se compone de:

- Circuitos combinacionales (LUTs)
- Circuitos secuenciales (FF)
- Primitivas de hardware: MAC, PLL, Dual Port RAM, etc
- Multiples I/O configurables

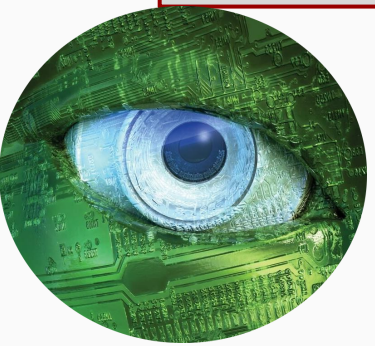




FPGA: Para qué se usa?



Se utilizan en distintos ámbitos, siempre relacionados a problemas de alto paralelismo, bajo consumo o prototipos de ASICs.



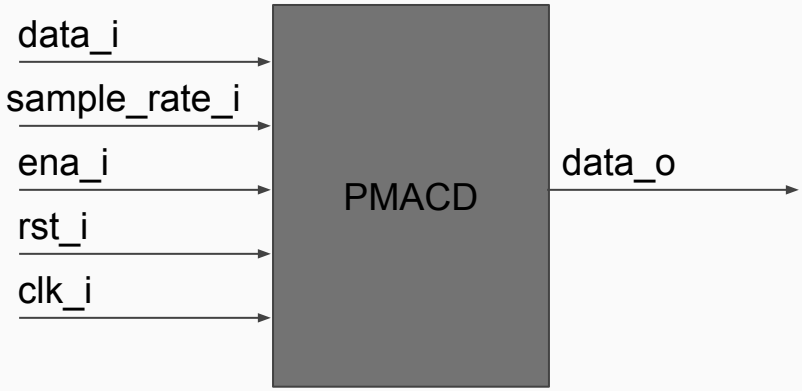
FPGA: como se usa?

Mediante un lenguaje de descripción de hardware se especifica cómo debería ser el comportamiento lógico del mismo.

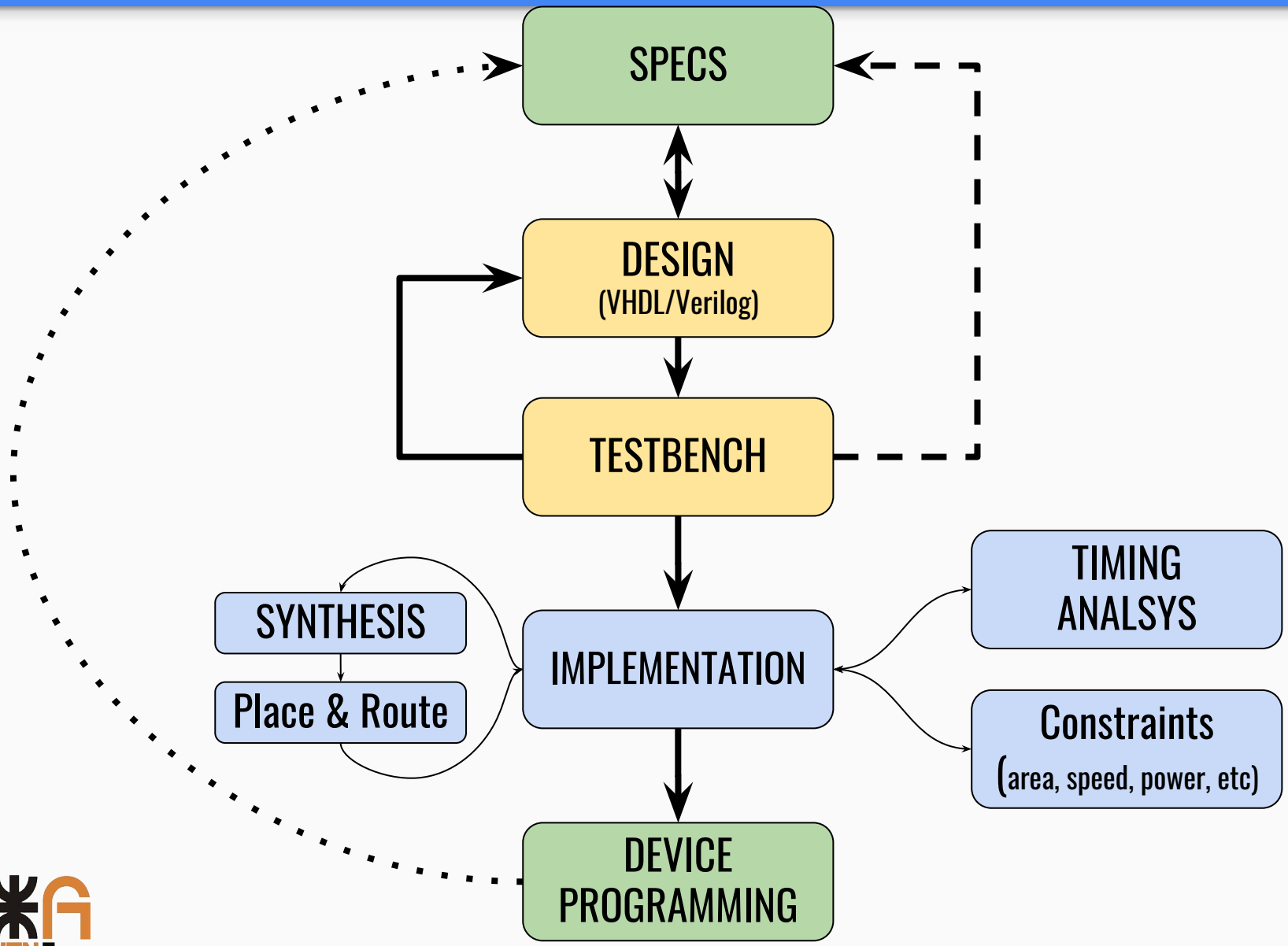
El lenguaje contempla definiciones combinacionales y secuenciales de un dispositivo digital.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;
library work;
use work.types.all;
use work.coefficients.all;

entity pmacd is
    generic(
        FW_REGS       : natural:=0;
        BW_REGS       : natural:=0;
        WBITS_IN      : natural:=10;
        WBITS_OUT     : natural:=28);
    port(
        clk_i         : in    std_logic;
        rst_i         : in    std_logic;
        ena_i         : in    std_logic;
        sample_rate_i: in    std_logic;
        data_i        : in    signed((WBITS_IN-1) downto 0);
        data_o        : out   signed((WBITS_OUT-1) downto 0)
        );
end entity pmacd;
```



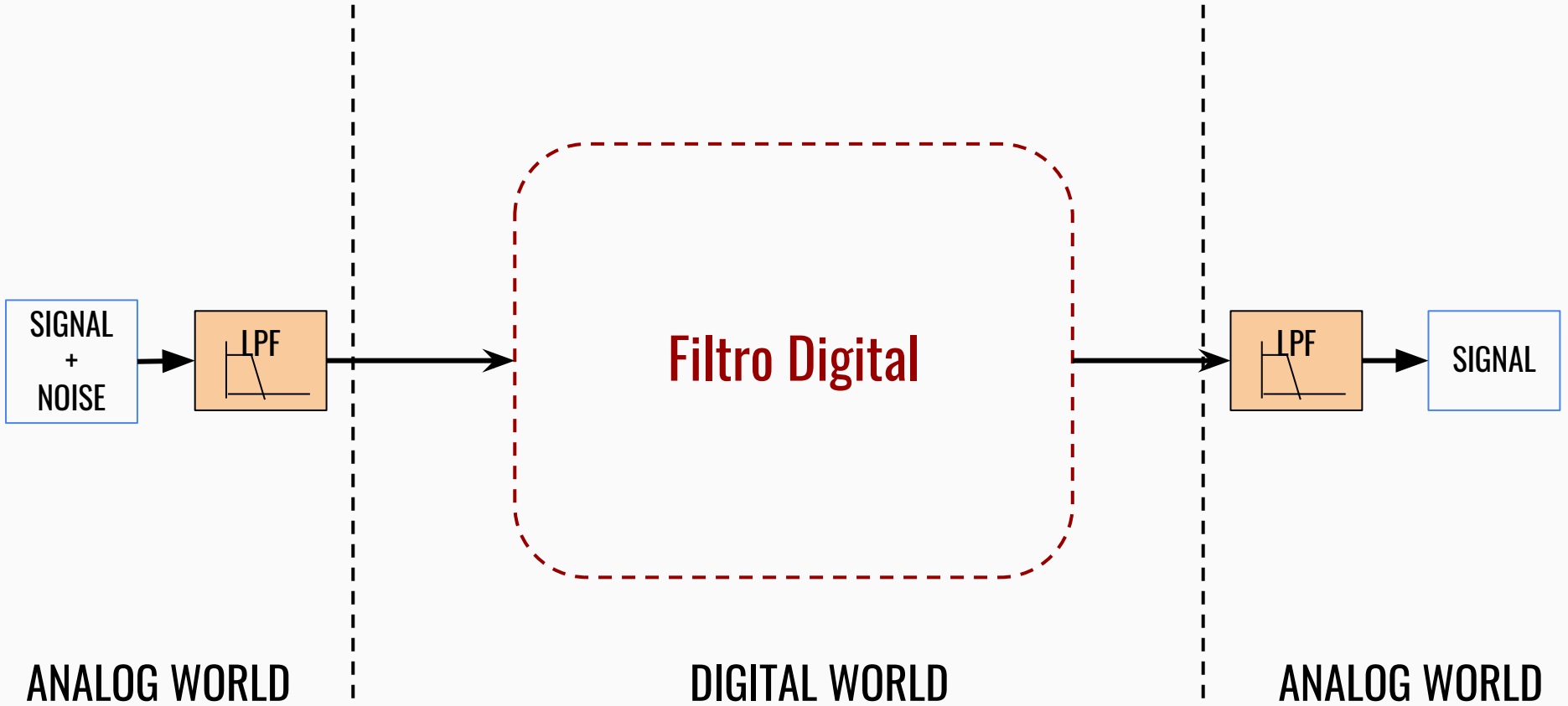
FPGA: workflow



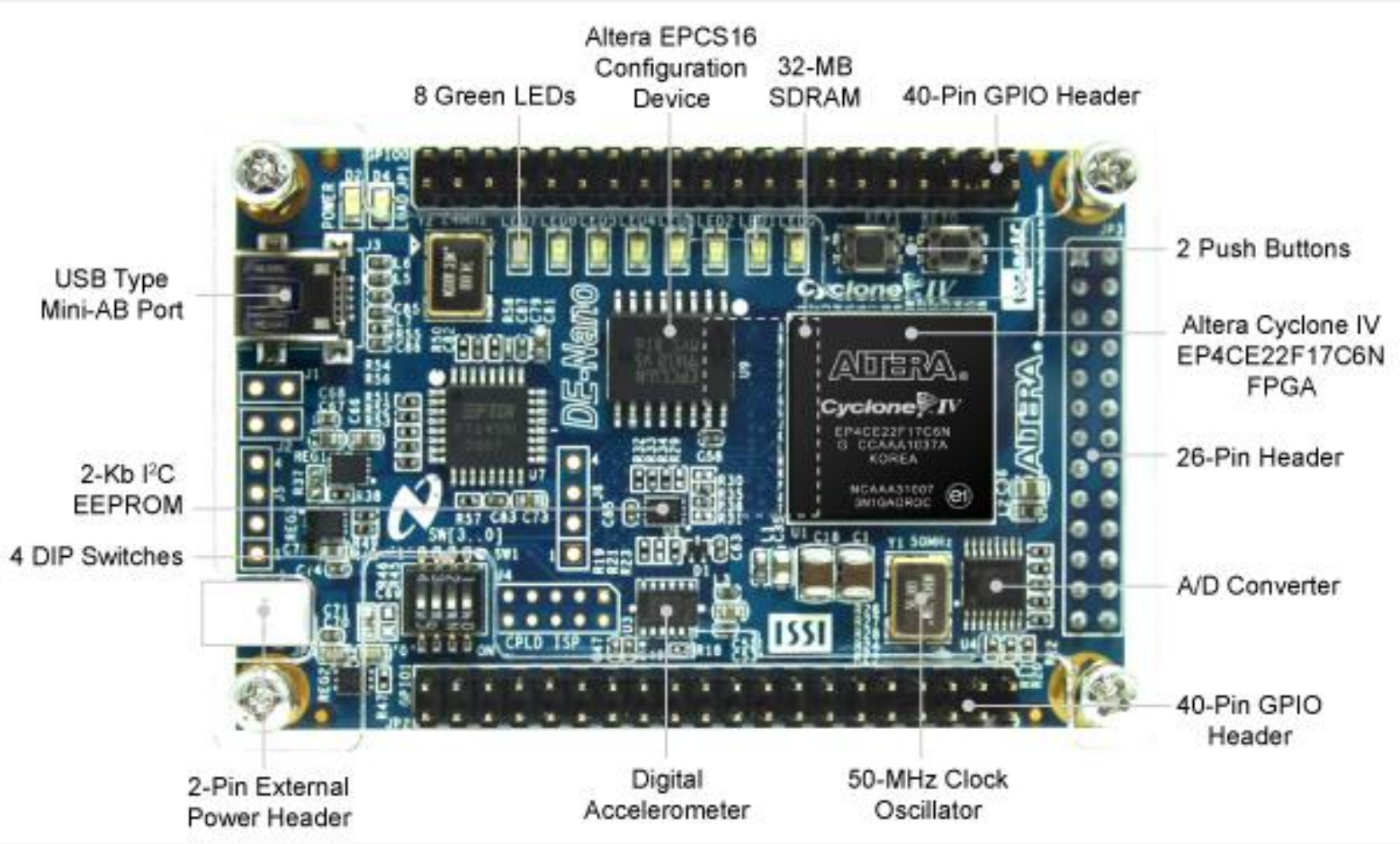
blabla!

Mejor vamos con un ejemplo de implementación

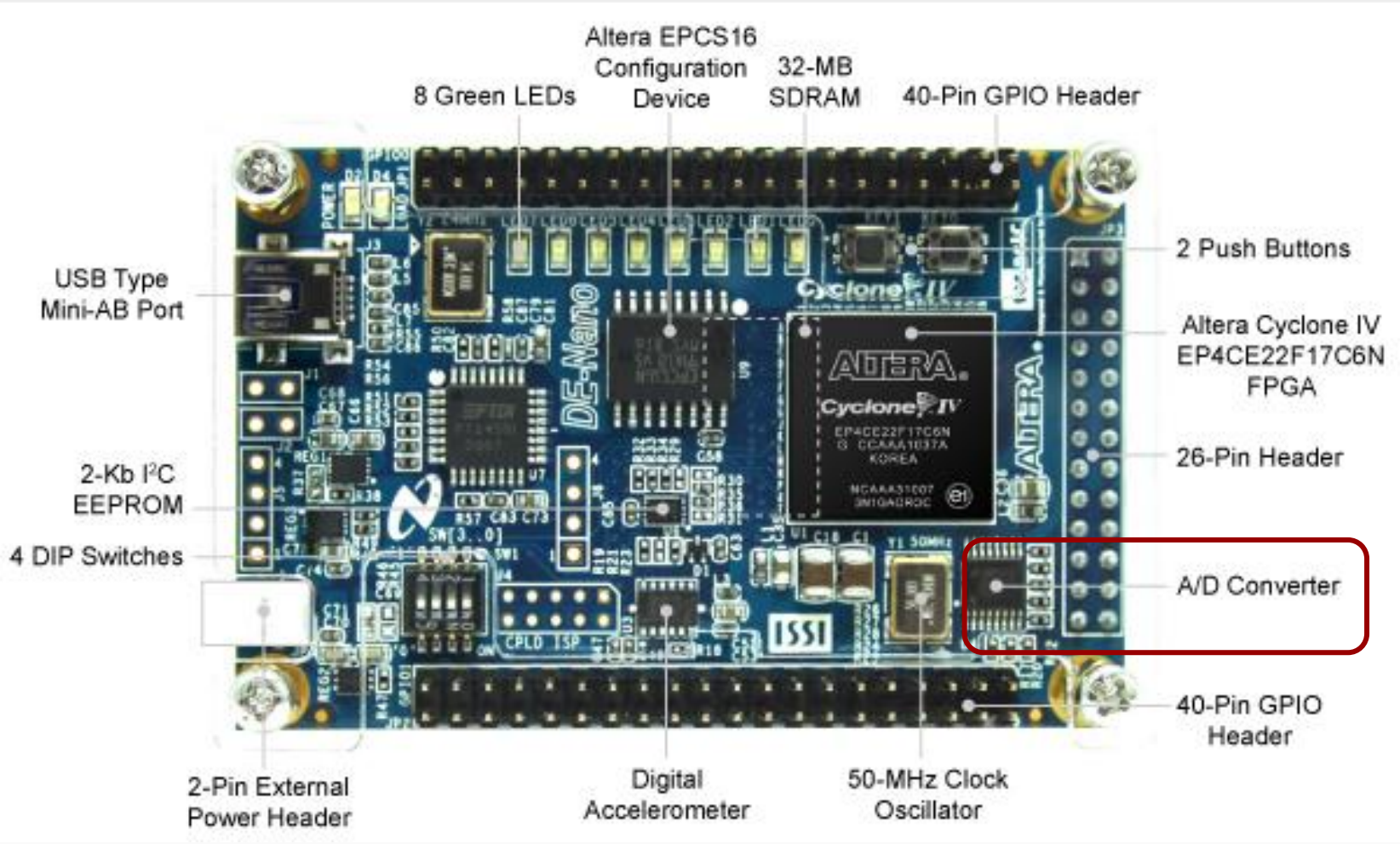
Sistema a Implementar



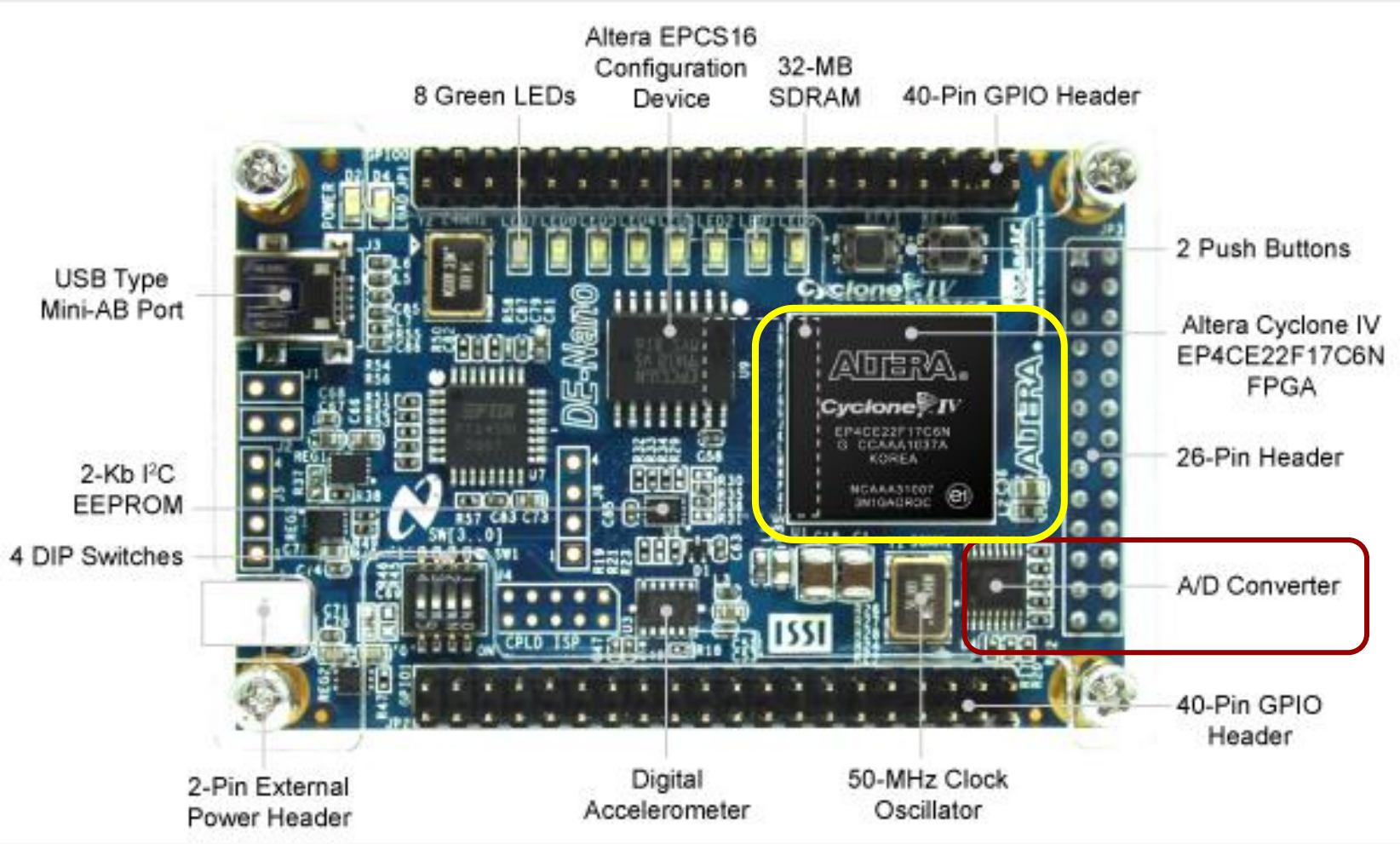
Kit: DEO-nano board (ALTERA)



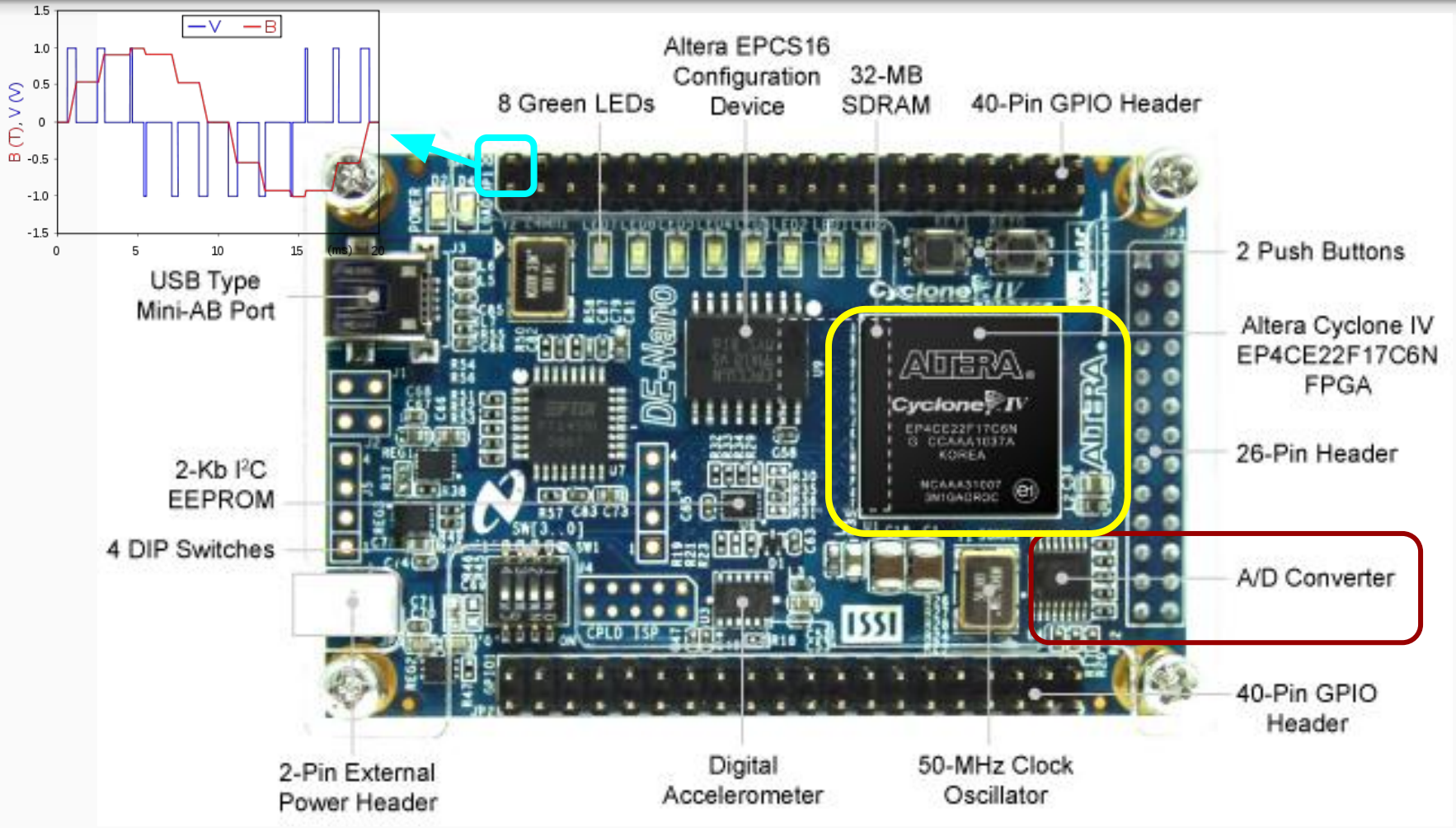
Kit: DEO-nano board (ALTERA)



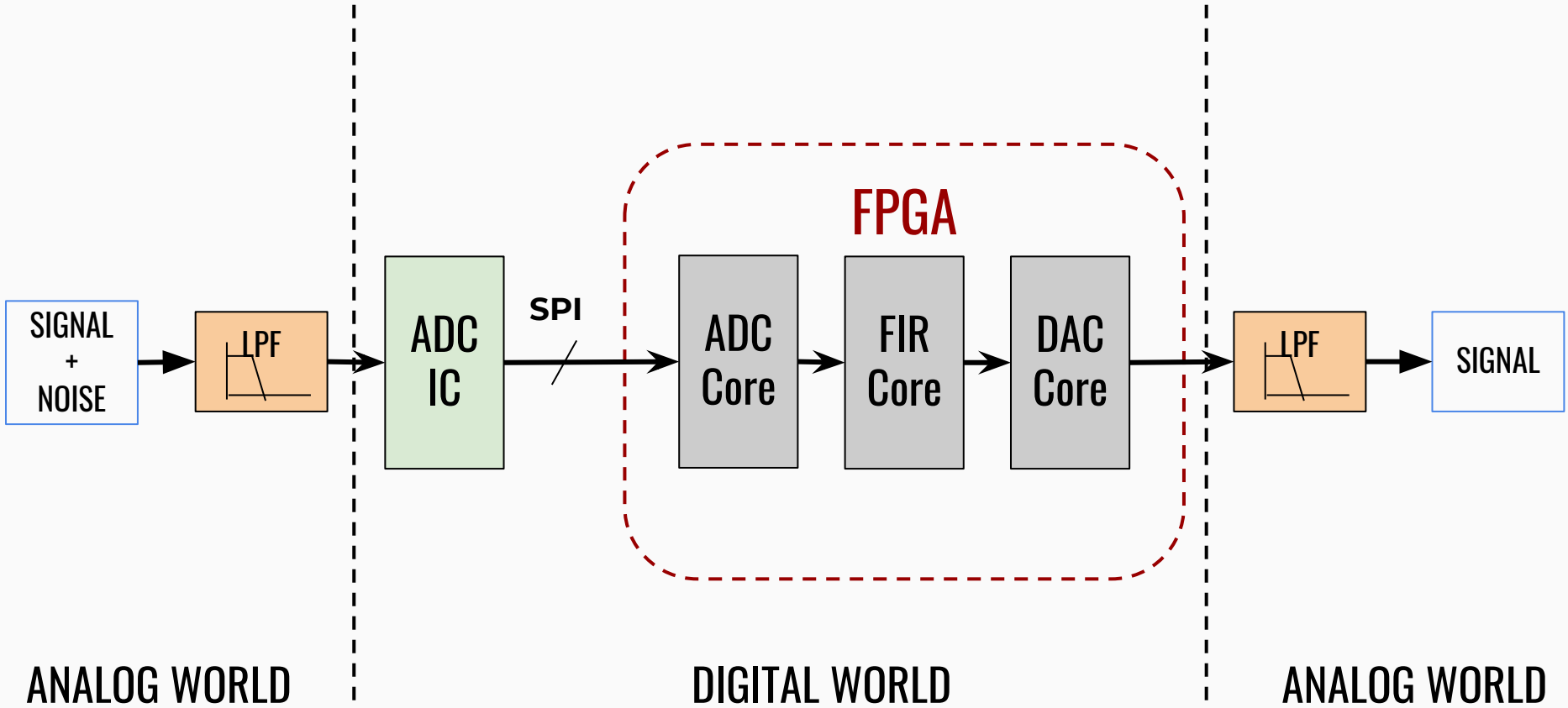
Kit: DEO-nano board (ALTERA)



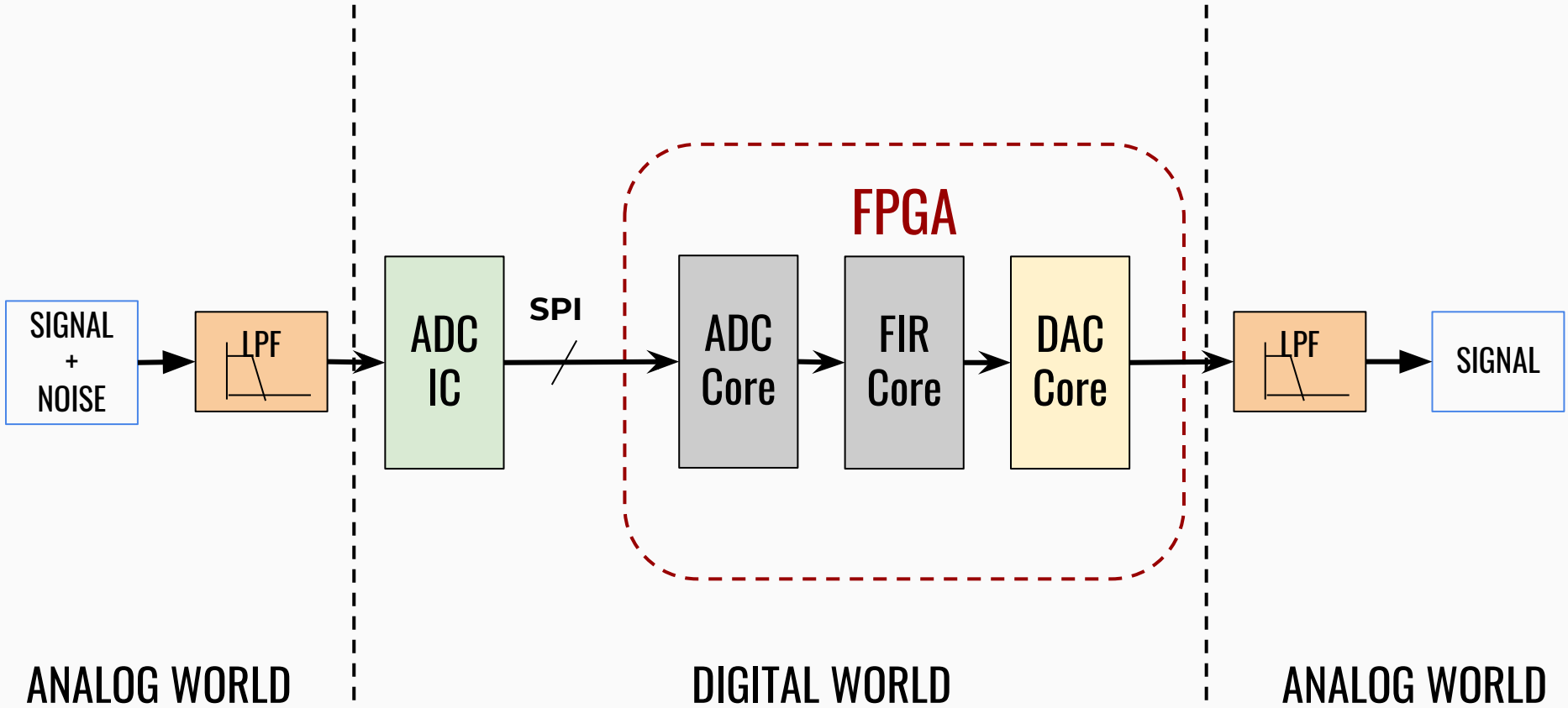
Kit: DEO-nano board (ALTERA)



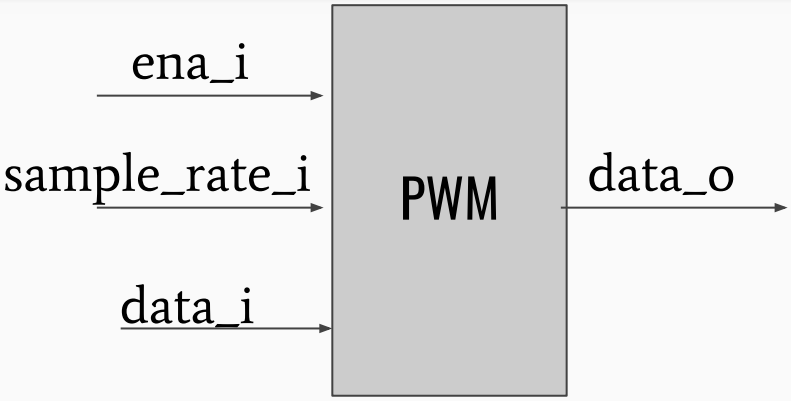
Sistema a Implementar



DAC

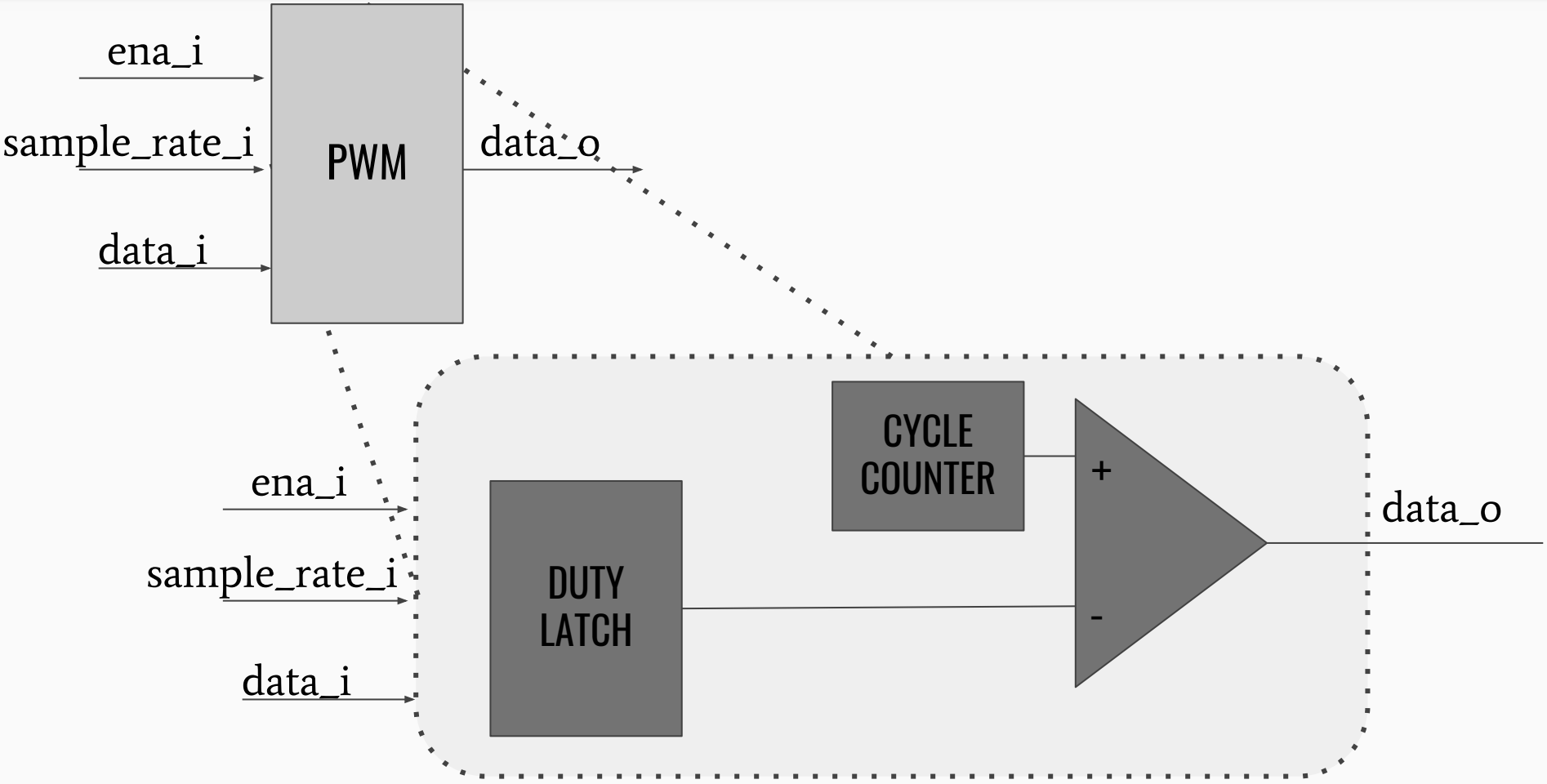

SIGNAL
+
NOISE
LPF
ADC
IC
SPI
FPGA
ADC
Core
FIR
Core
DAC
Core
LPF
SIGNAL
ANALOG WORLD
DIGITAL WORLD
ANALOG WORLD

DAC/PWM Especificación



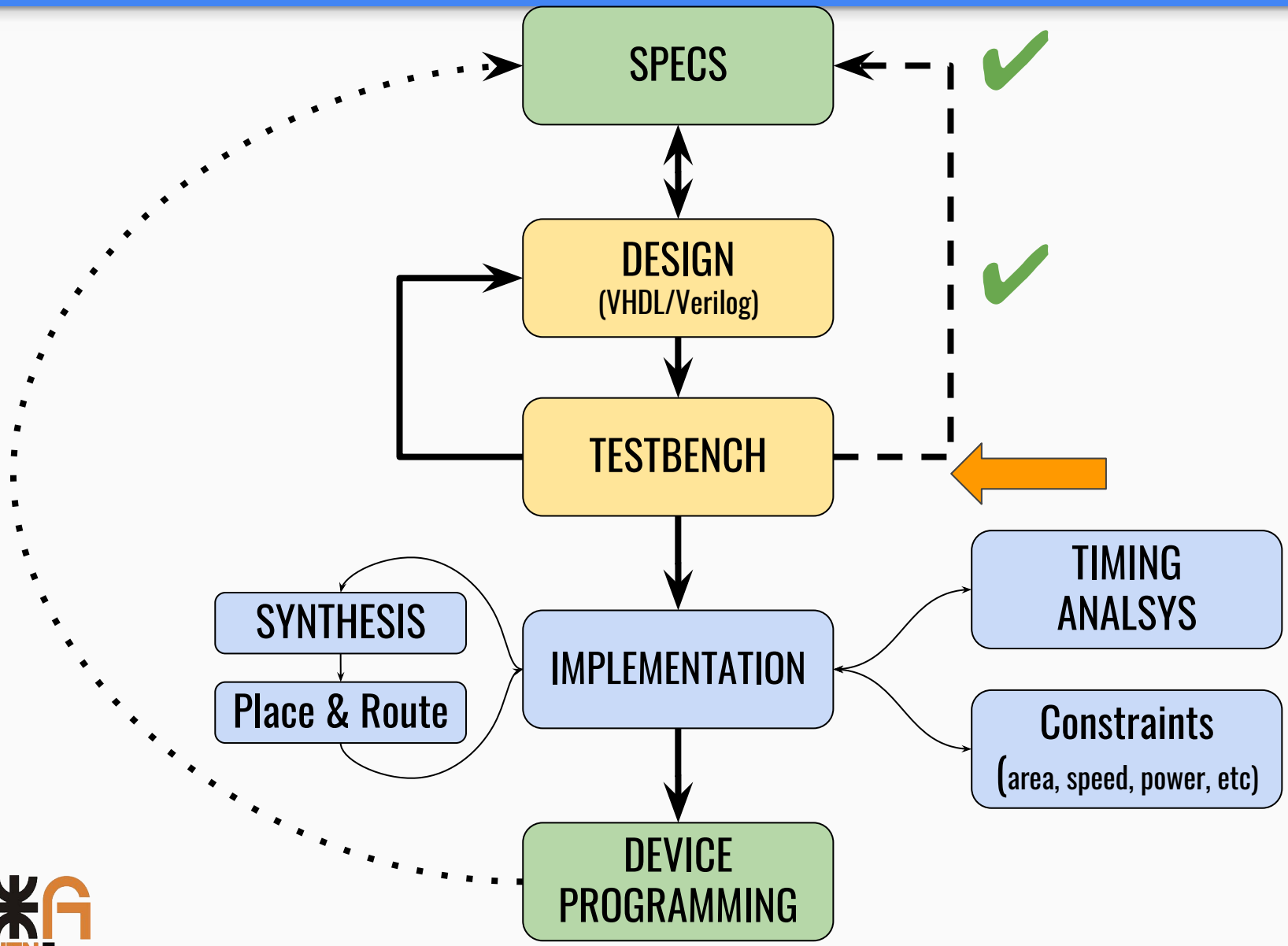
Señal	Función
ena_i	Define cada cuanto opera (Fpwm)
sample_rate_i	Define cada cuanto se actualiza el duty cycle
data_i	Nuevo valor con el duty cycle
data_o	Señal modulada en ancho de pulso

DAC/PWM Diseño

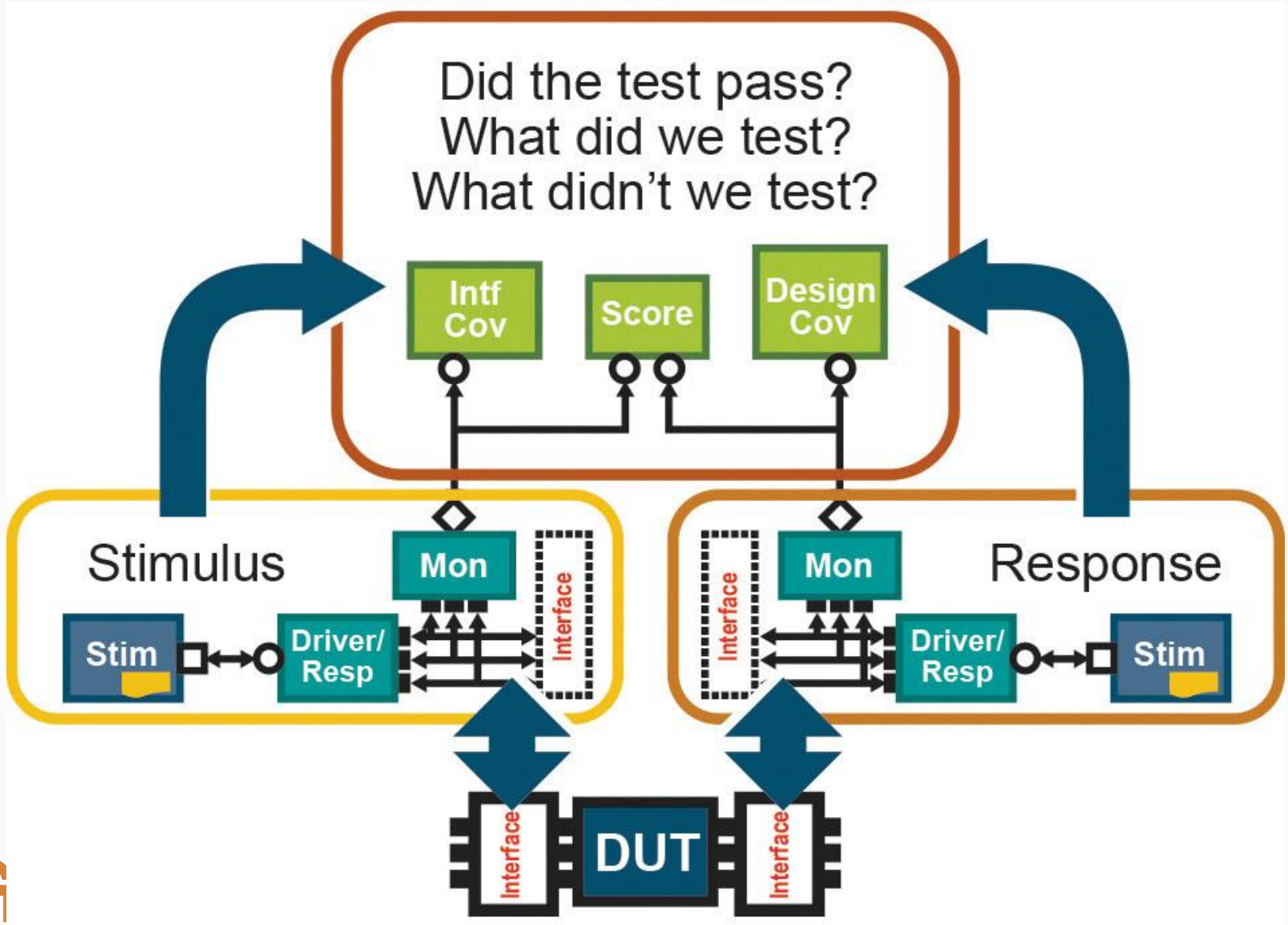


Al código del PWM!

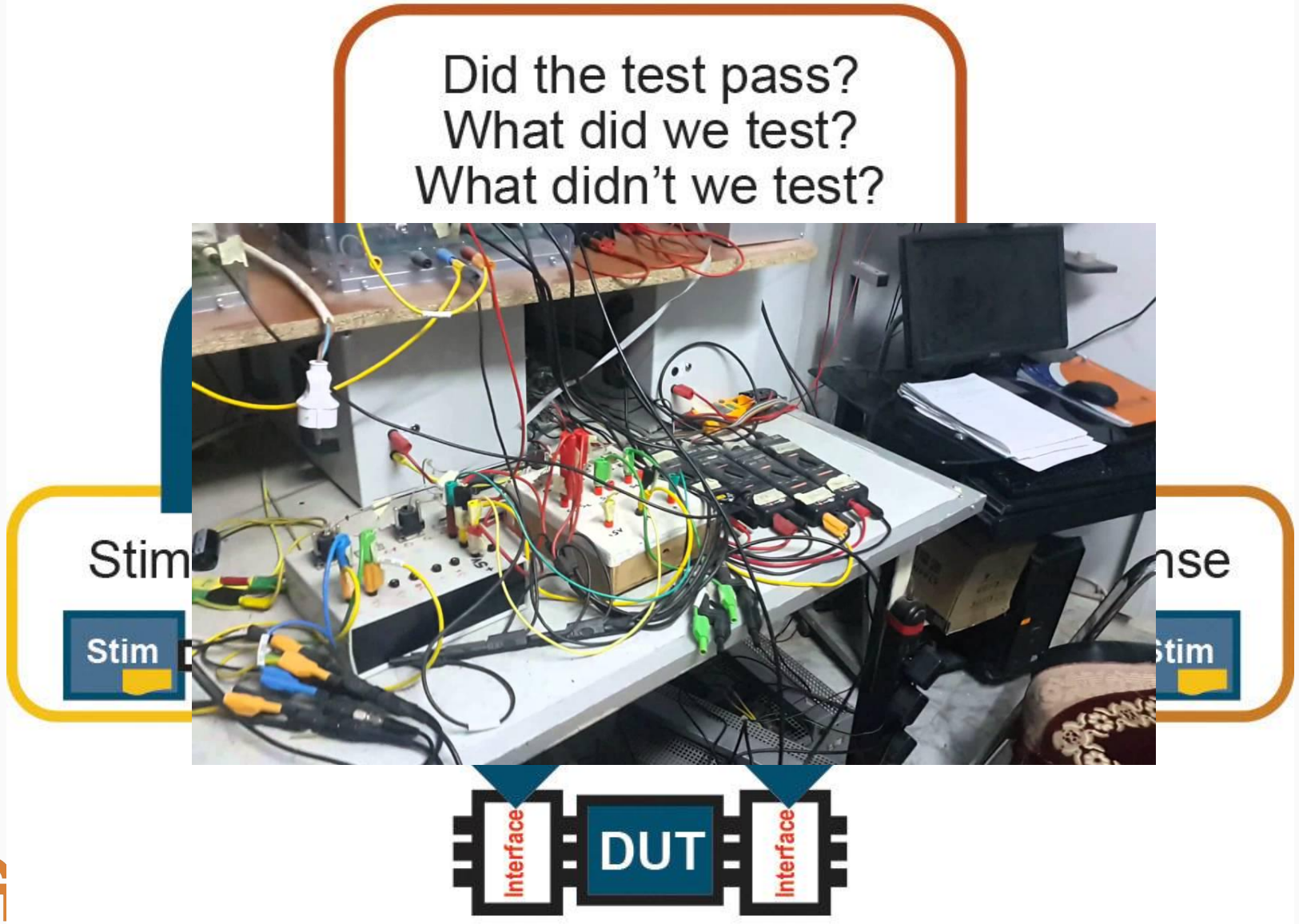
FPGA: workflow



Testbench: qué es?

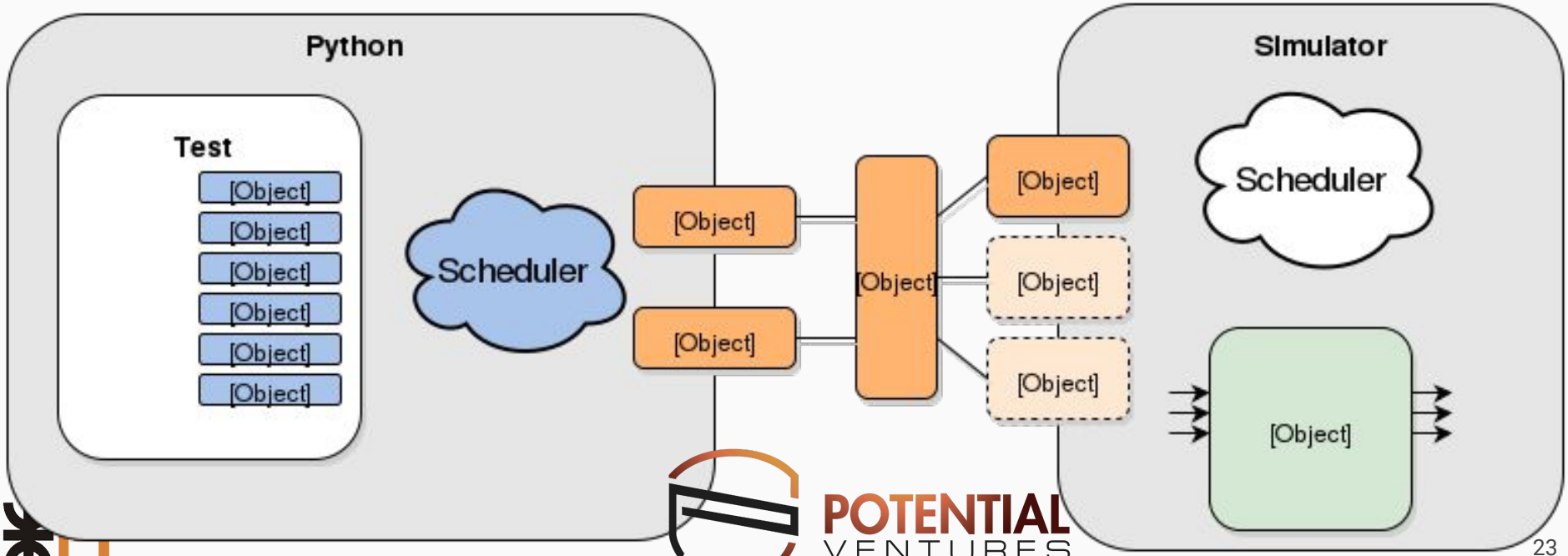


Testbench: qué es?



Testbench: COCOTB!

- Cocotb is a COroutine based COsimulation TestBench environment for verifying VHDL/Verilog RTL using Python.
- Cocotb is completely free, open source (under the BSD License) and hosted on GitHub.
- Cocotb requires a simulator to simulate the RTL. Simulators that have been tested and known to work with Cocotb



Testbench: COCOTB! + PYTHON

Python

- Lenguaje de programación interpretado de fácil lectura (au
- Código Abierto
- Multiparadigma
  - Funcional
  - Objetos
  - Imperativo
- Comunidad de millones de personas (https://www.python.org/) y comunidad local (http://www.python.org.ar/)
- En cuanto a signal processing:
  - matplotlib: http://matplotlib.org/ (Para graficar)
  - scipy: http://www.scipy.org/ (Orientado a matemáticas, ciencias e ingeniería)
  - numpy: http://www.numpy.org/ (Fundamental para computación científica)
- etc, etc, etc, etc, etc

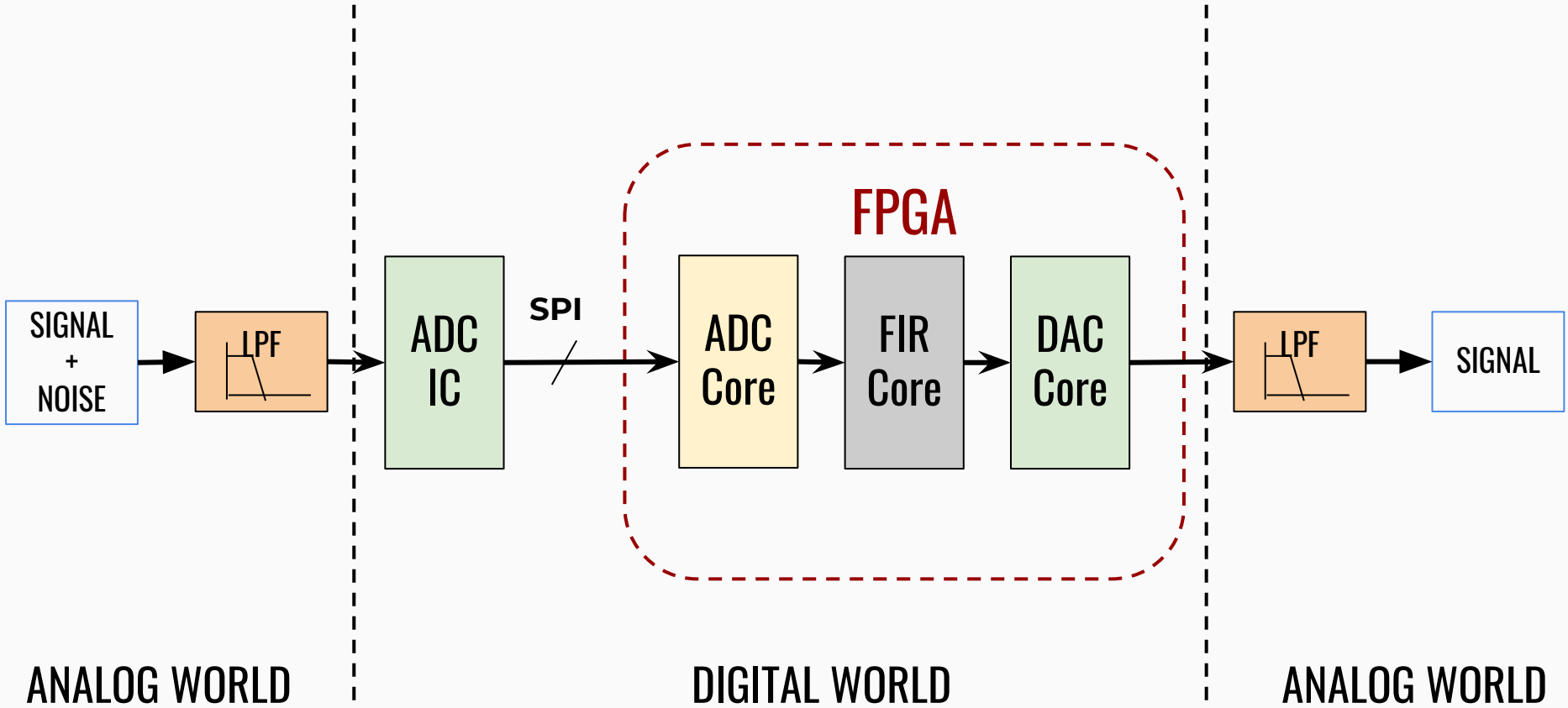
Testbench: COCOTB! + PYTHON

Python

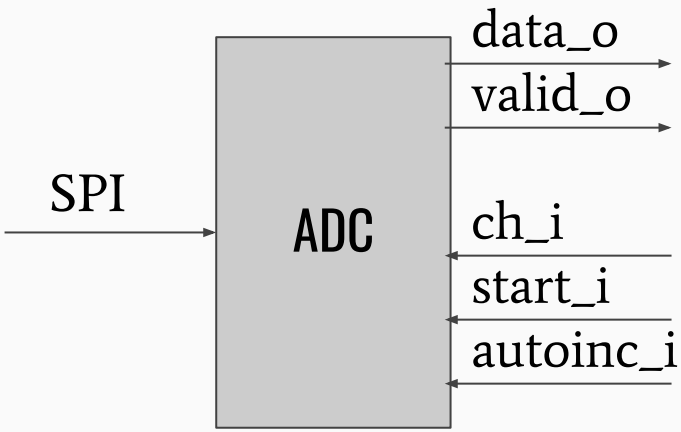
VHDL/Verilog

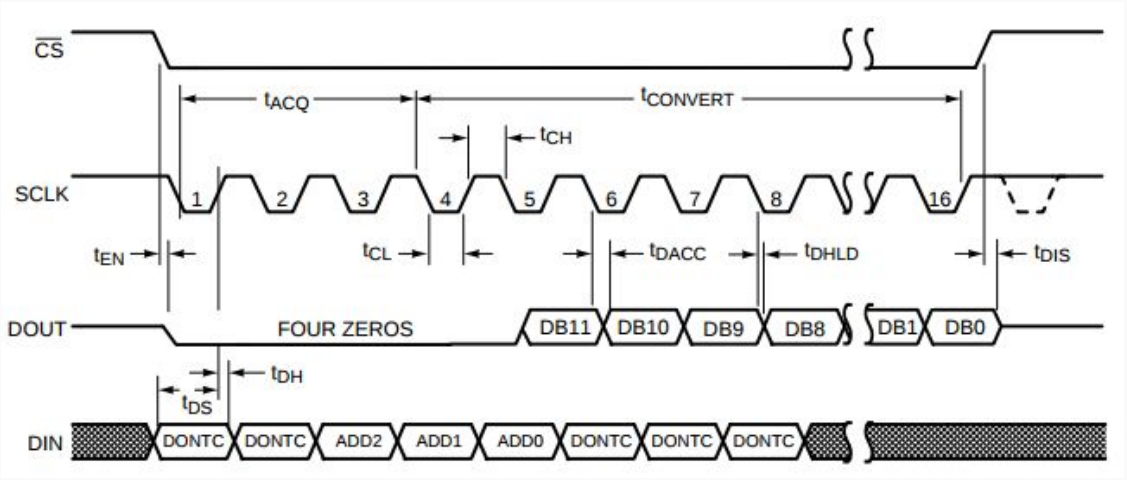
AI testbench!

ADC


SIGNAL + NOISE
LPF
ADC IC
SPI
FPGA
ADC Core
FIR Core
DAC Core
LPF
SIGNAL
ANALOG WORLD
DIGITAL WORLD
ANALOG WORLD

ADC: Especificación

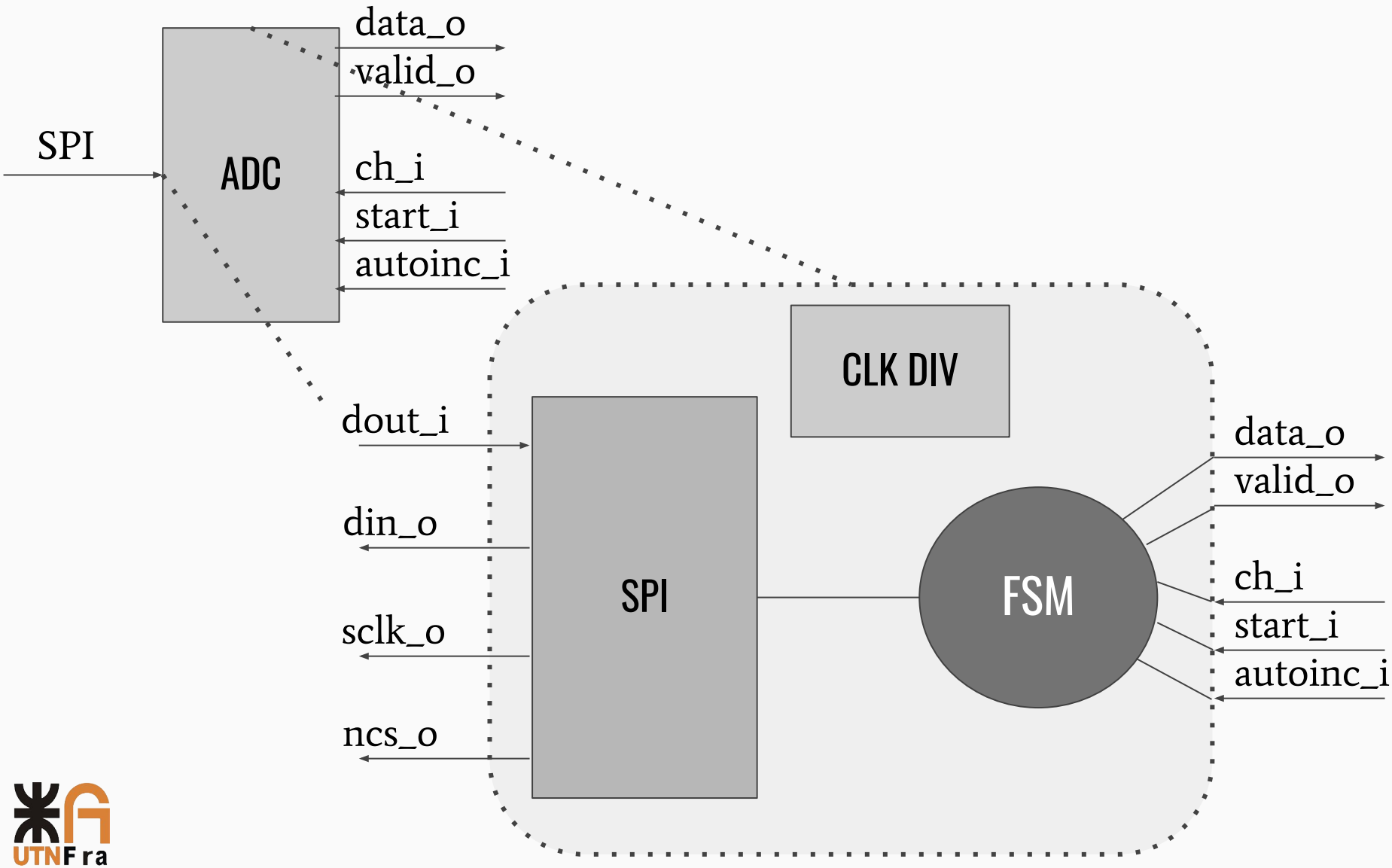




ADC128S022

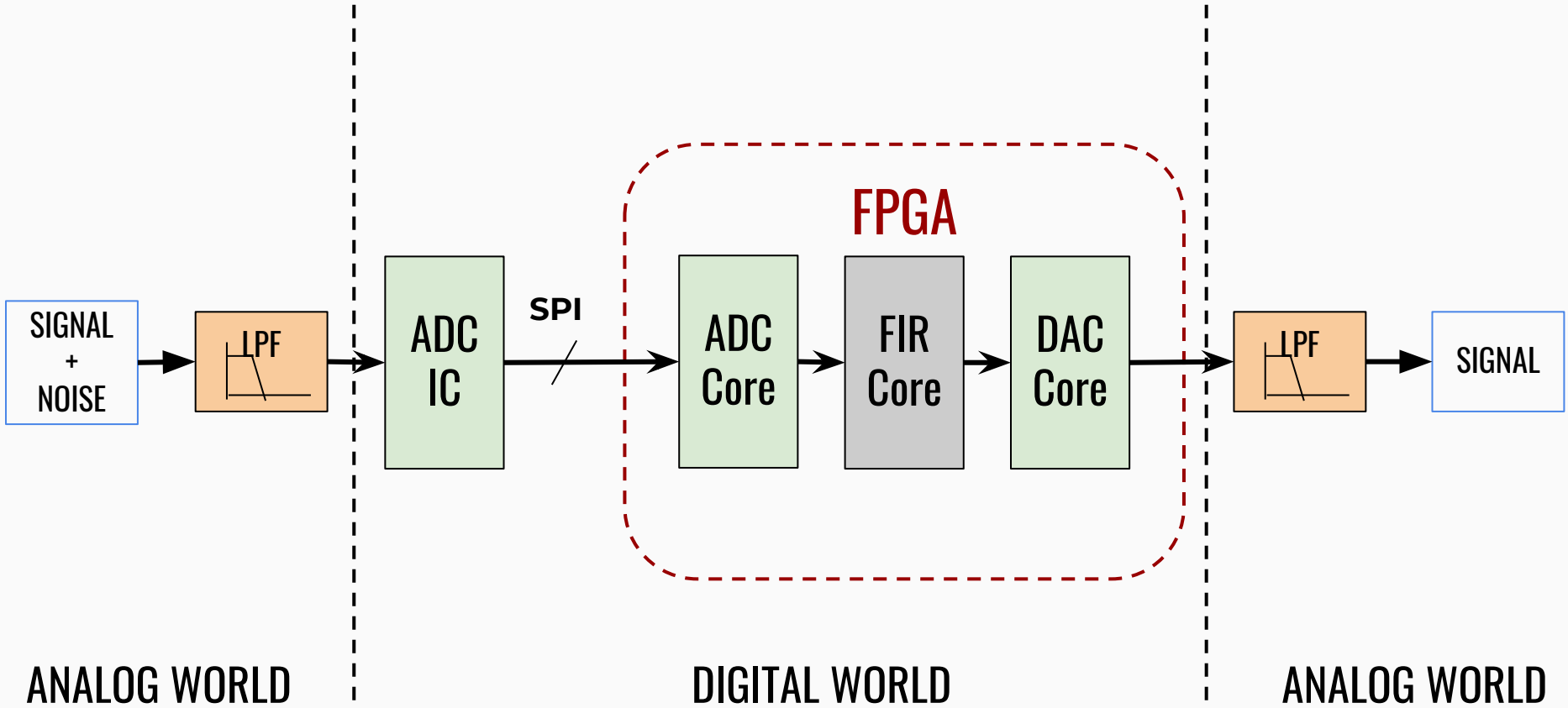
FMAX	3.2MHz
Sample Rate MAX	200kS/s
Sample Rate	F_{SCLK} / 16

ADC: Diseño (Divide & Conquer)

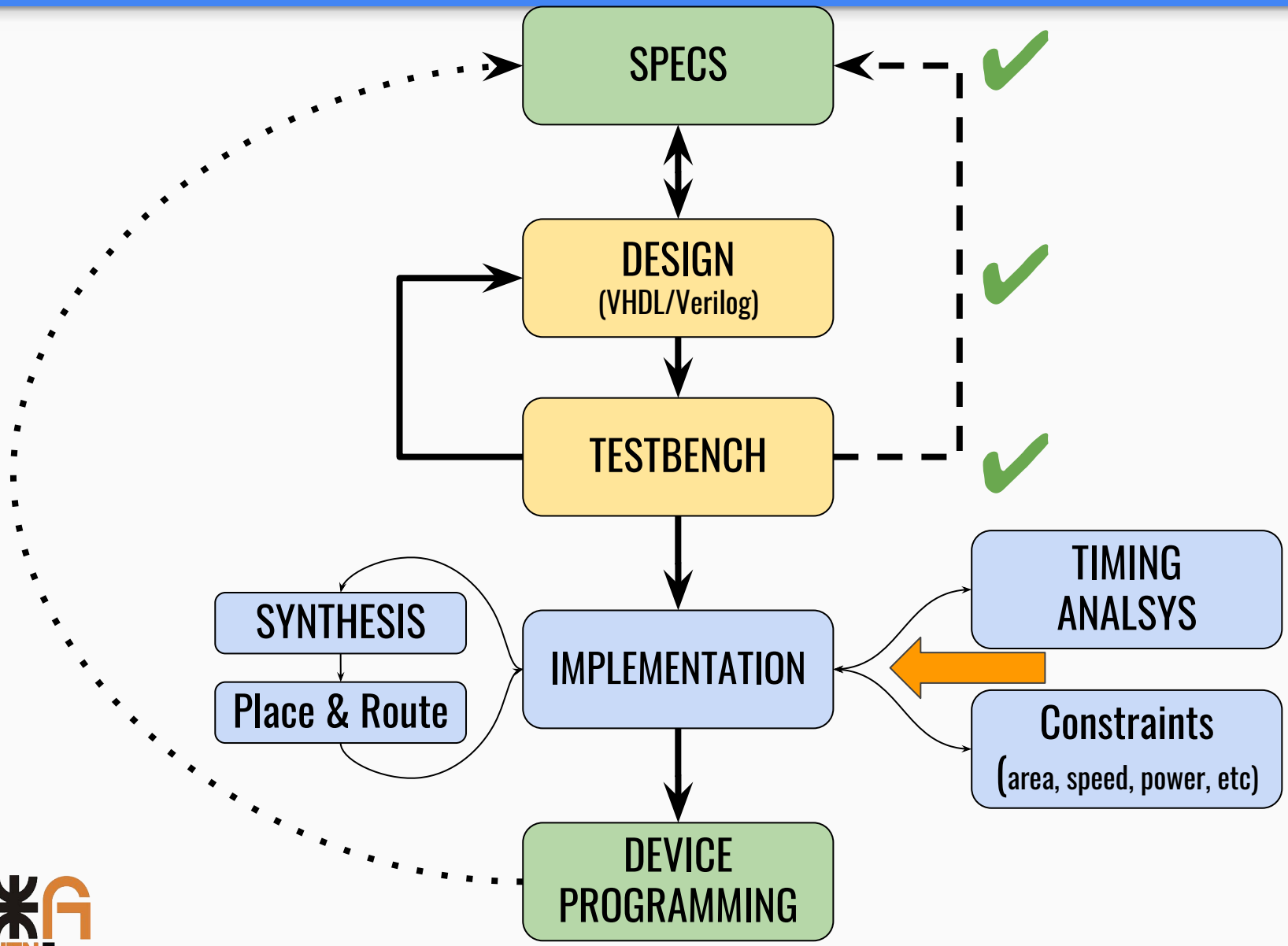


Al código del ADC!

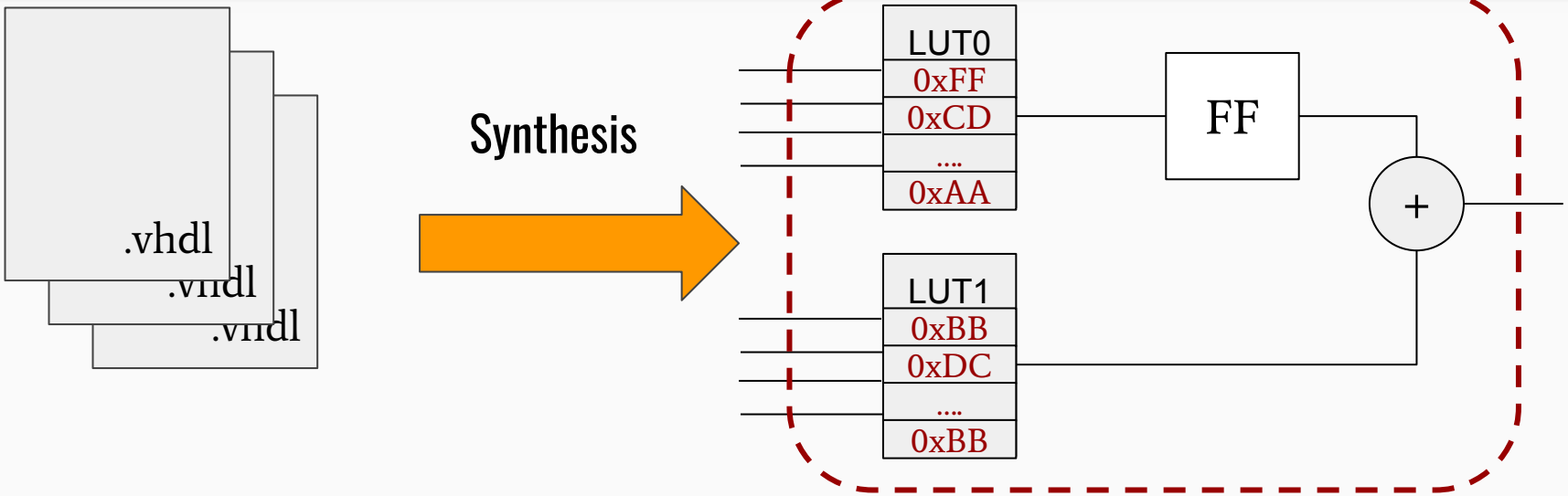
Listo: ADC y DAC



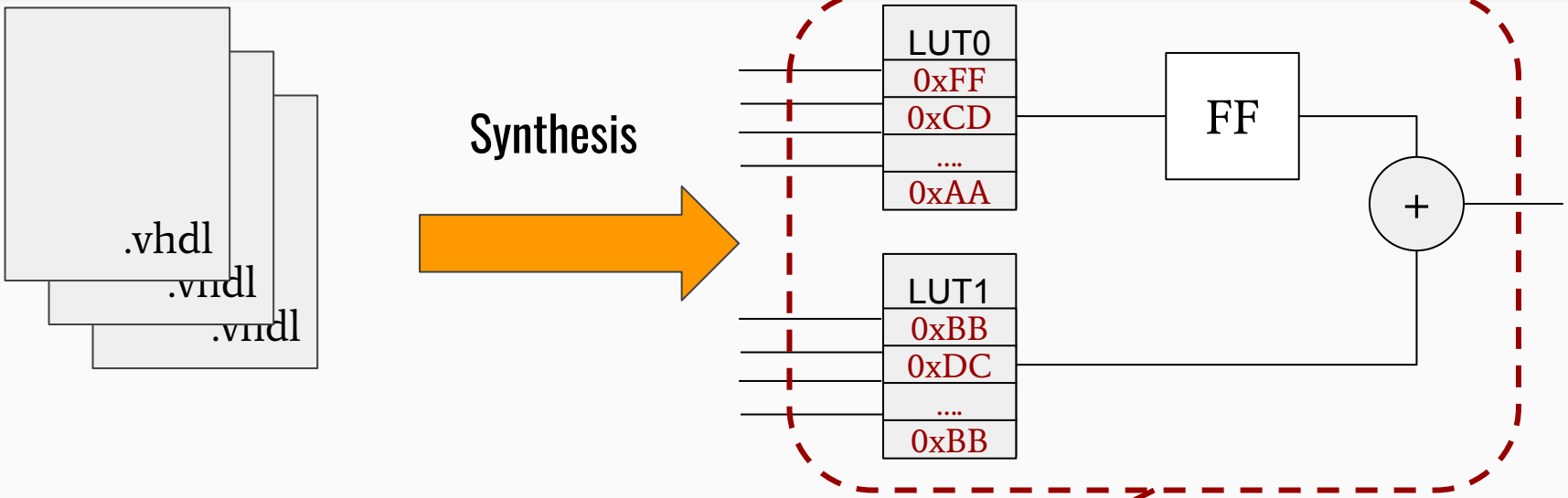
FPGA: workflow

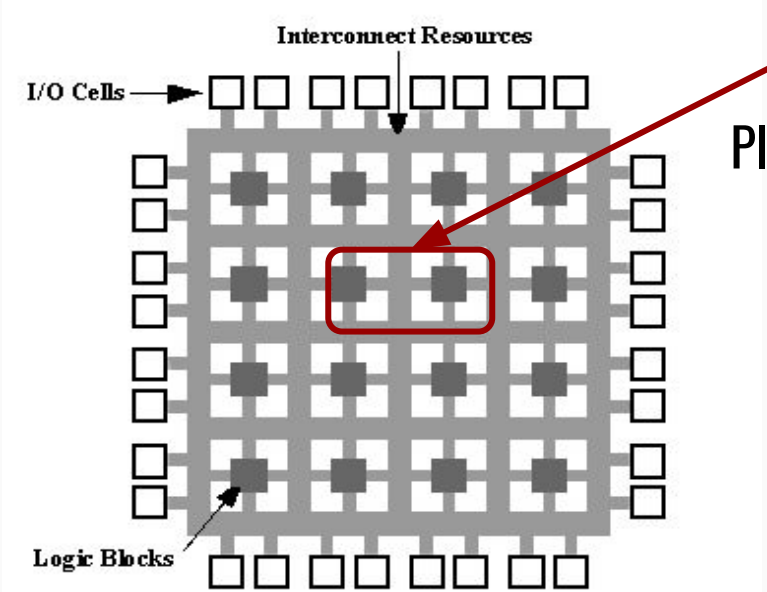


Implementación: Síntesis, Place and Route



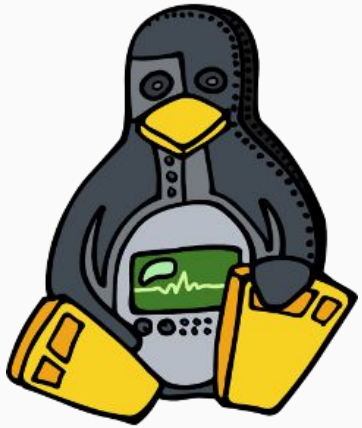
Implementación: Síntesis, Place and Route





Place & Route

Implementación: HDLMake



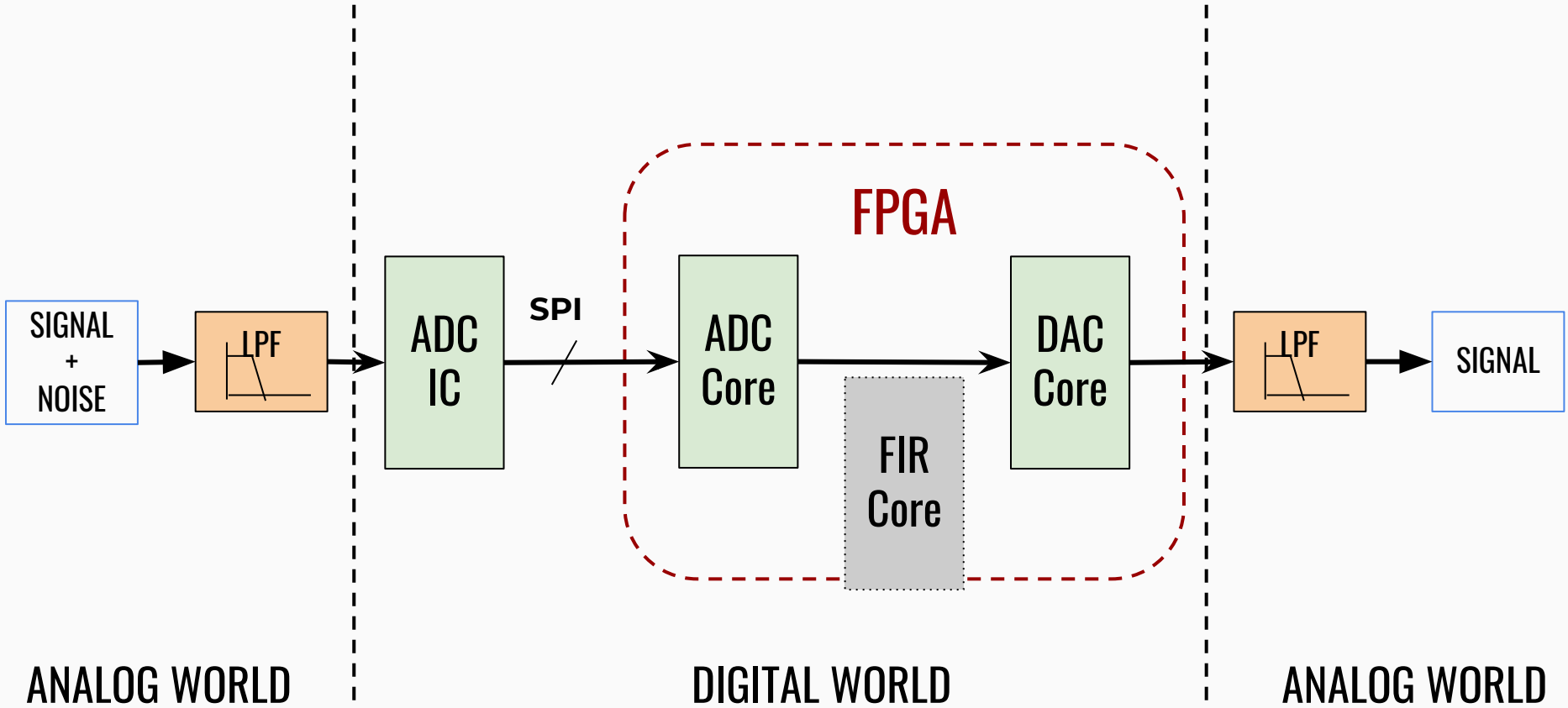
Tool for generating multi-purpose makefiles for FPGA projects.

Main features:

- makefile generation for:
  - fetching modules from repositories
  - simulating HDL projects
  - synthesizing HDL projects
  - synthesizing project remotely (keeping your local resources free)
- generating multi-vendor project files (no clicking in the IDE!)
- many other things without involving make and makefiles

Hdlmake supports **modularity**, **scalability**, **revision control systems**. Hdlmake can be run on any Linux or Windows machine with any HDL project.

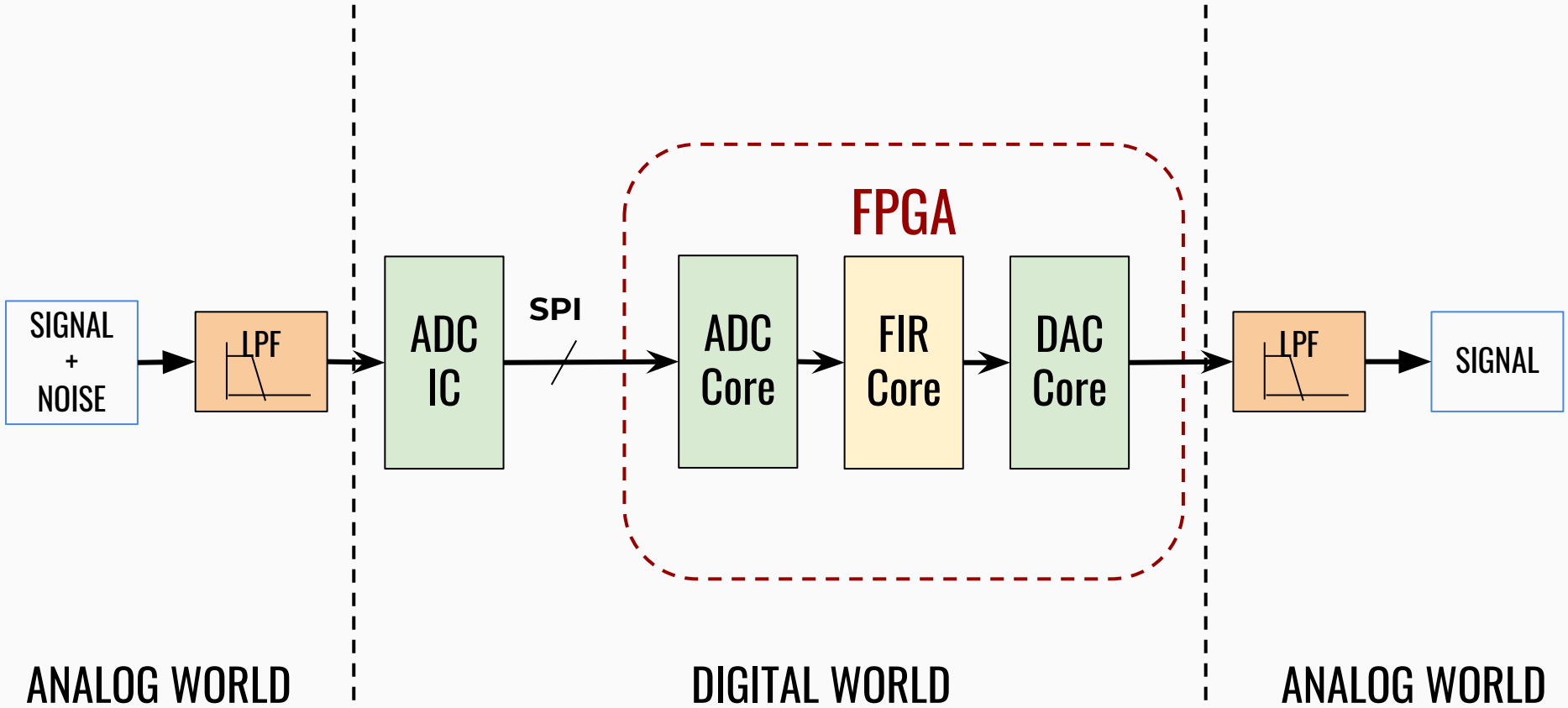
Implementación: ADC+PWM



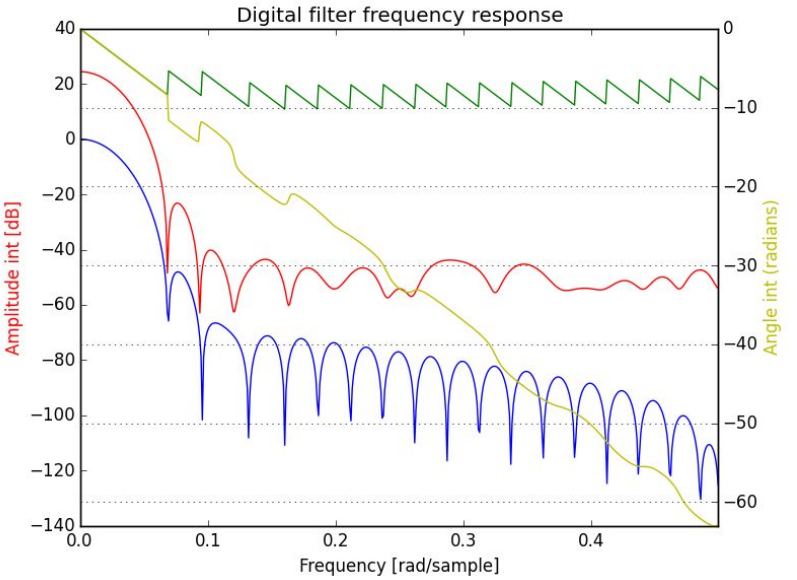
Implementación en hardware

Filtro FIR

Ahora arranca la charla...



Filtro FIR

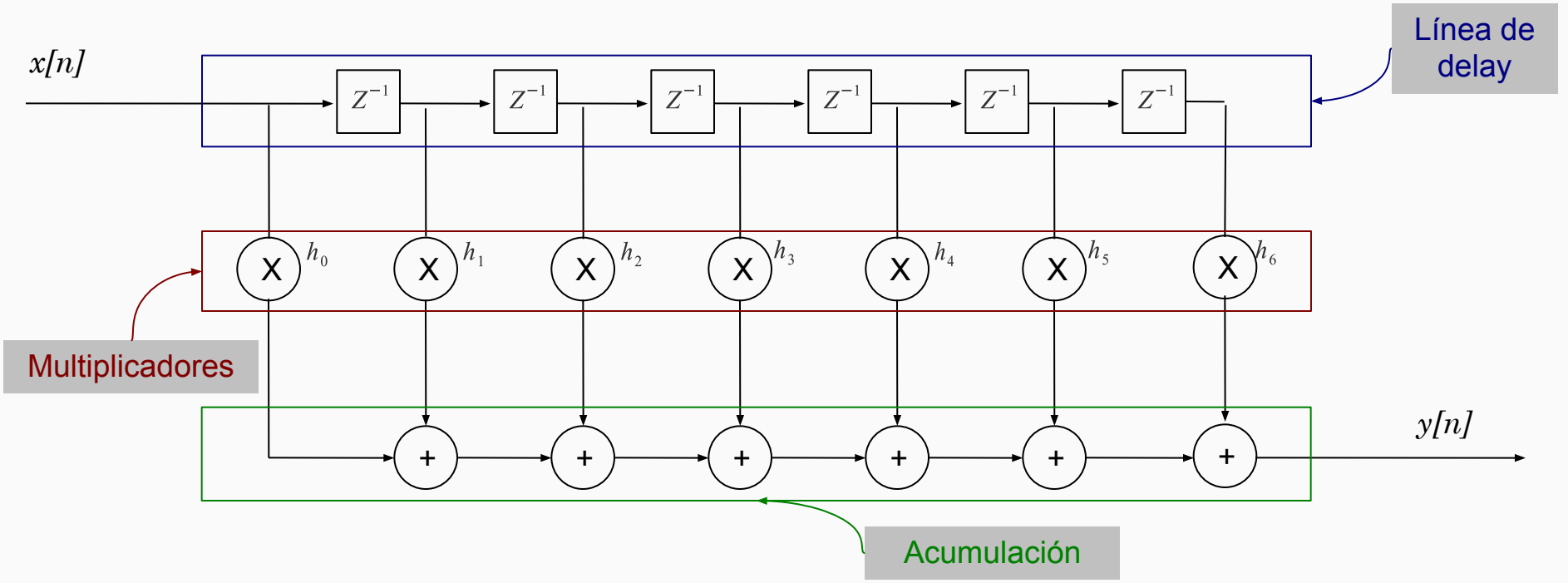


In signal processing, a **finite impulse response (FIR)** filter is a filter whose impulse response (or response to any finite length input) is of *finite* duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).
https://en.wikipedia.org/wiki/Finite_impulse_response

$$y[n] = h_0 x[n] + \ldots + h_M x[n - M]$$

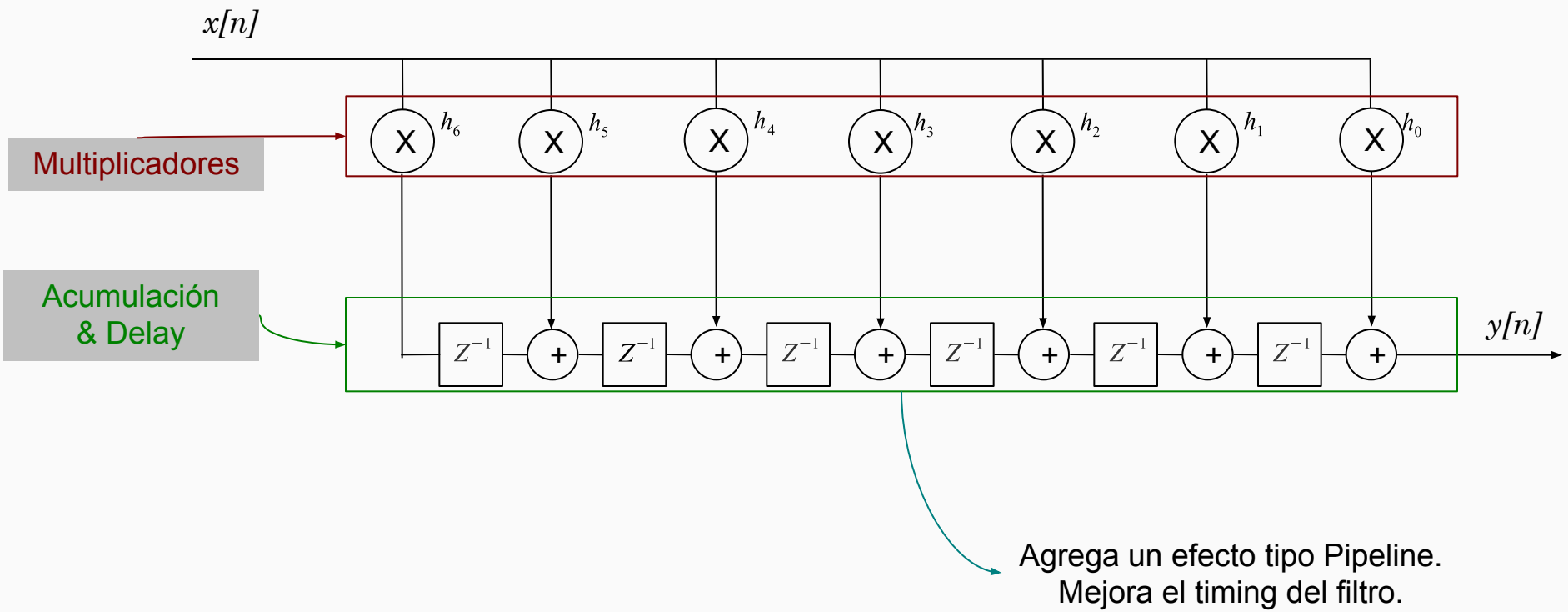
$$y[n] = \sum_{i=0}^{M} h_i x[n - i]$$

Filtro FIR: forma directa



$$y[n] = \sum_{i=0}^{M} h_i x[n-i]$$

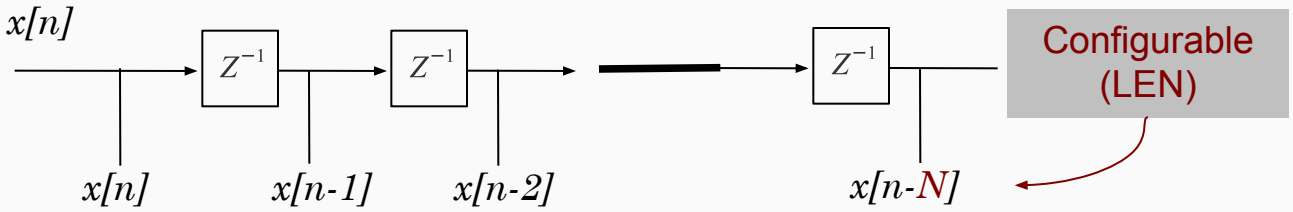
Filtro FIR: forma transpuesta o inversa



$$y[n] = \sum_{i=0}^{M} h_i x[n-i]$$

Criterio de Diseño: Divide & Conquer

Filtro FIR: Delay-IN

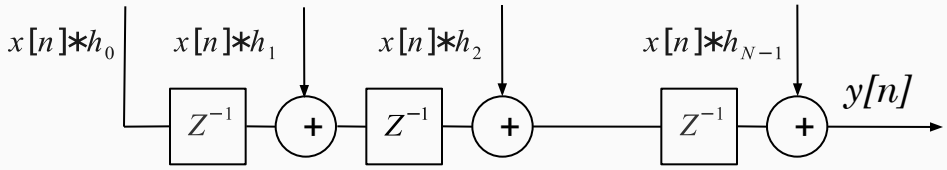



```
entity DelayLine_in is
    generic(
        LEN         : natural:= 4;
        WBITS       : integer:= 10);
    port(
        clk_i       : in    std_logic;
        rst_i       : in    std_logic;
        ena_i       : in    std_logic:='1';
        data_i      : in    signed((WBITS-1) downto 0);
        data_o      : out   signed_array(0 to (LEN-1))
    );
end entity DelayLine_in;
```

- La entrada rst_i garantiza un estado en el tiempo t=0
- La señal ena_i permite realizar el shifteo a un múltiplo de la frecuencia de clk_i.
- WBITS: define la resolución de la entrada

Filtro FIR: Delay-OUT

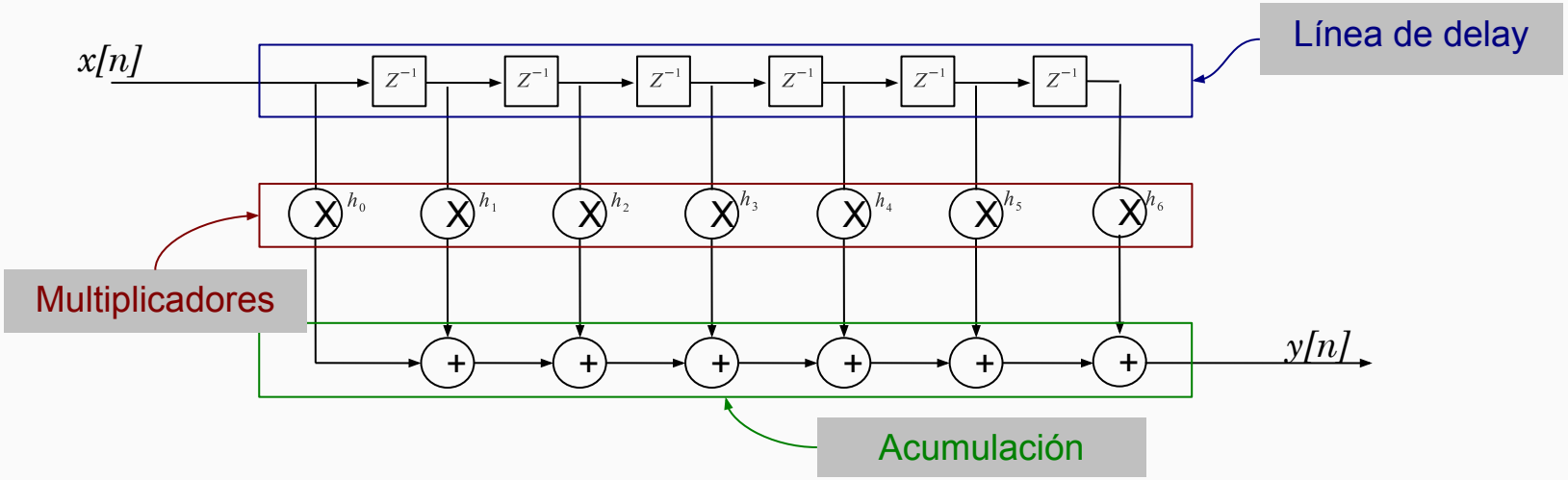
Elementos básicos: Delay-Out



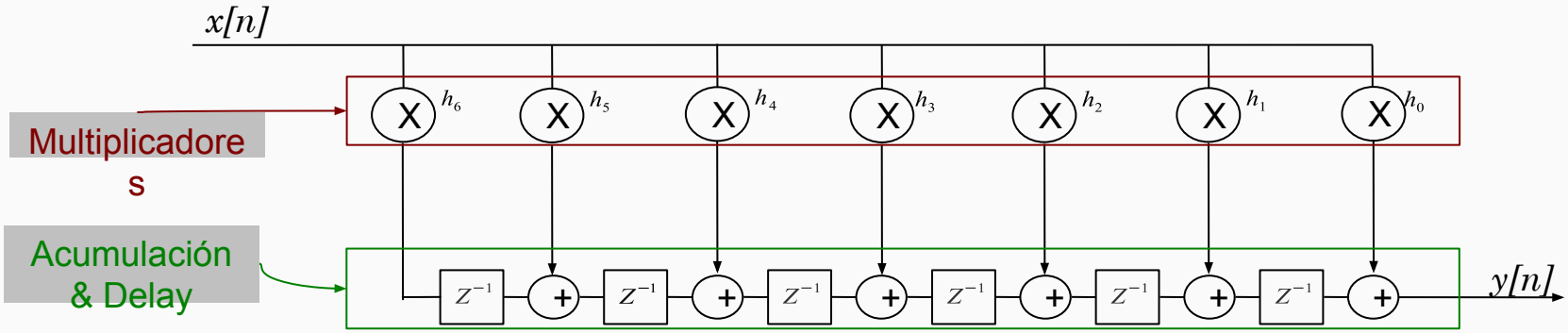

```
entity DelayLine_OUT is
    generic(
        LEN         : natural:=4;
        WBITS       : integer:= 10);
    port(
        clk_i       : in    std_logic;
        rst_i       : in    std_logic;
        ena_i       : in    std_logic;
        data_i      : in    signed_array(0 to (LEN-1));
        data_o      : out   signed((WBITS-1) downto 0)
    );
end entity DelayLine_OUT;
```

- La entrada rst_i garantiza un estado en el tiempo t=0
- La señal ena_i permite realizar el shifteo a un múltiplo de la frecuencia de clk_i.
- WBITS: define la resolución de la entrada

Filtros FIR: Arquitecturas en paralelo



DIRECTA

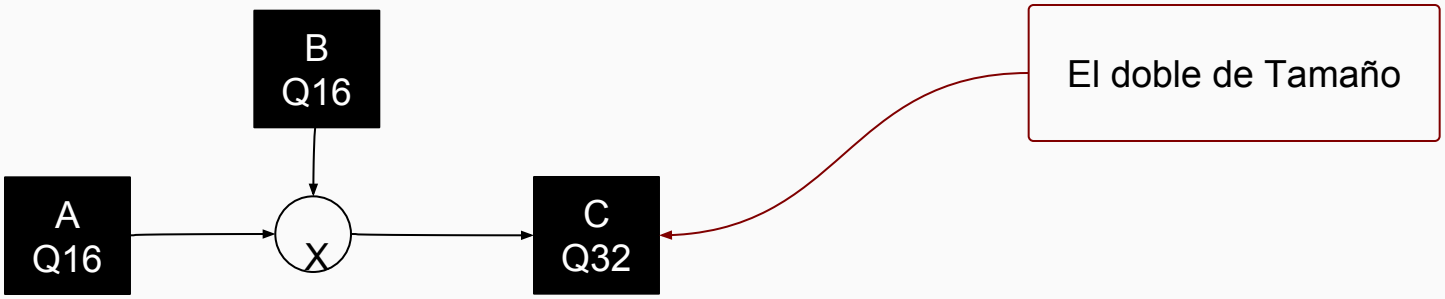


INVERSA

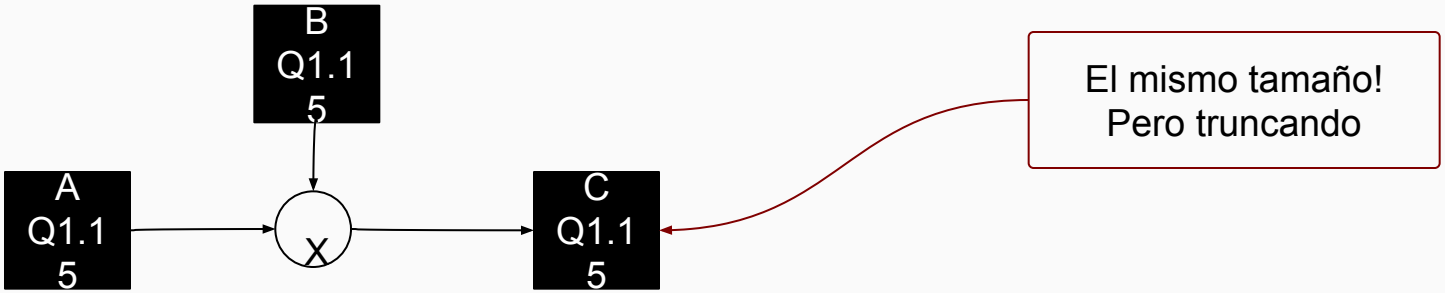
Detalles para la Implementación

Sistema de numeración y tamaños de señales

Es necesario que las señales intermedias tengan un tamaño para contener los resultados parciales de los productos



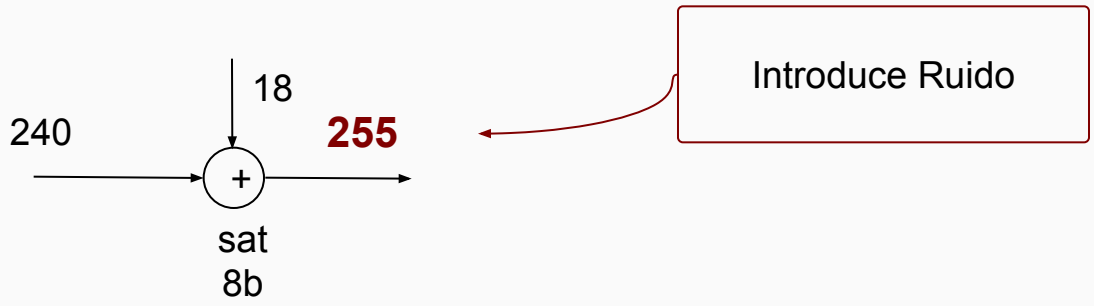
Si se usa un sistema de numeración en punto fijo como: 1.X. El resultado queda auto-contenido en la misma representación



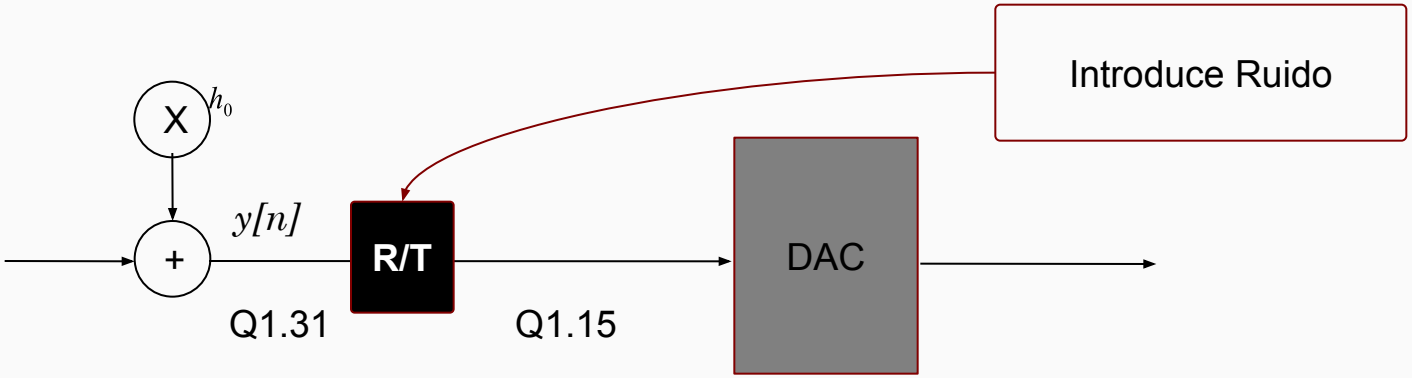
Detalles para la Implementación

Overflow y Ajuste de Salida

Se puede usar lógica saturada en los sumadores para evitar el desborde

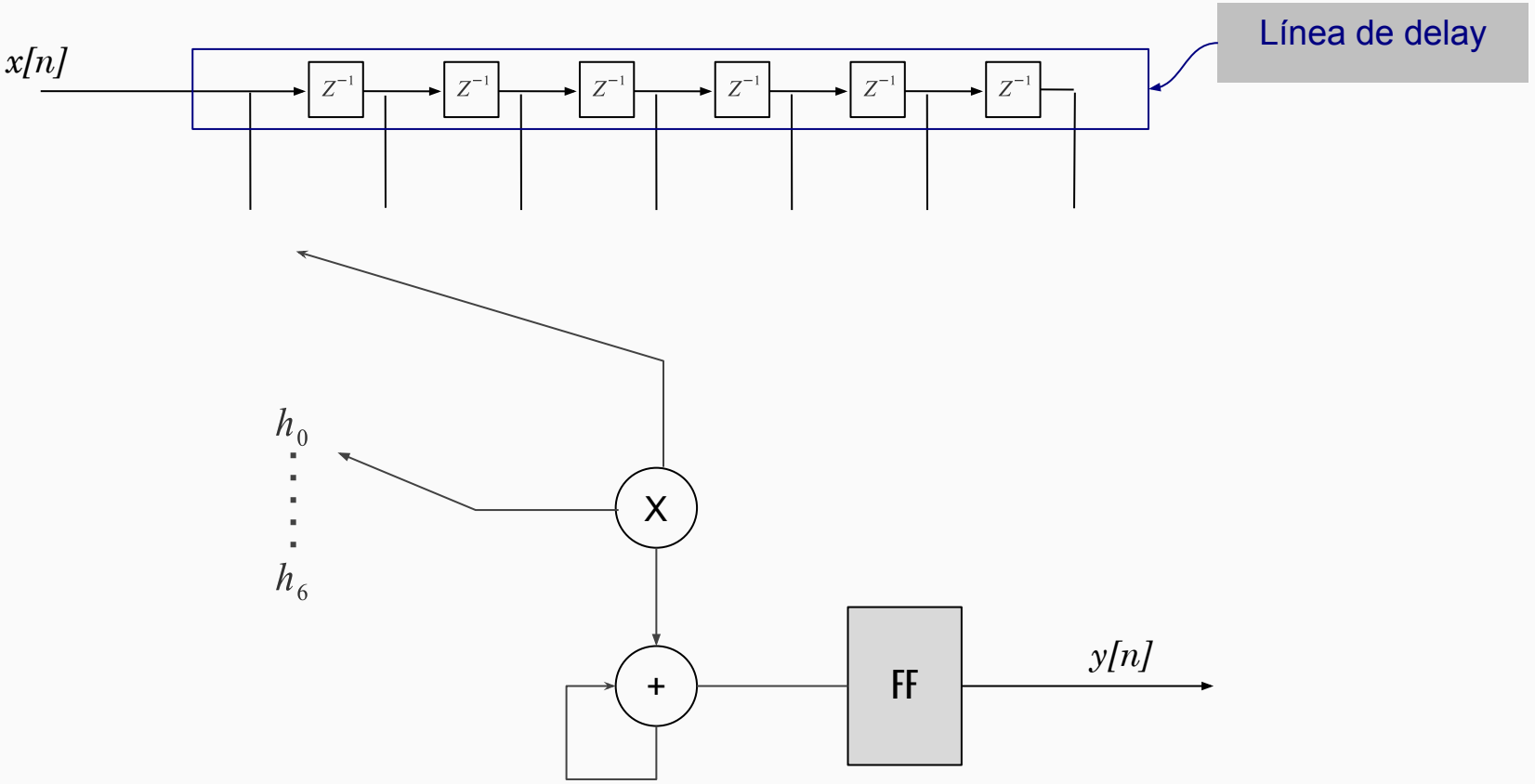


A veces es necesario ajustar el tamaño de la salida del filtro a un tamaño similar al de la entrada. Se puede redondear o truncar.



Implementación: PMACD & PMACI

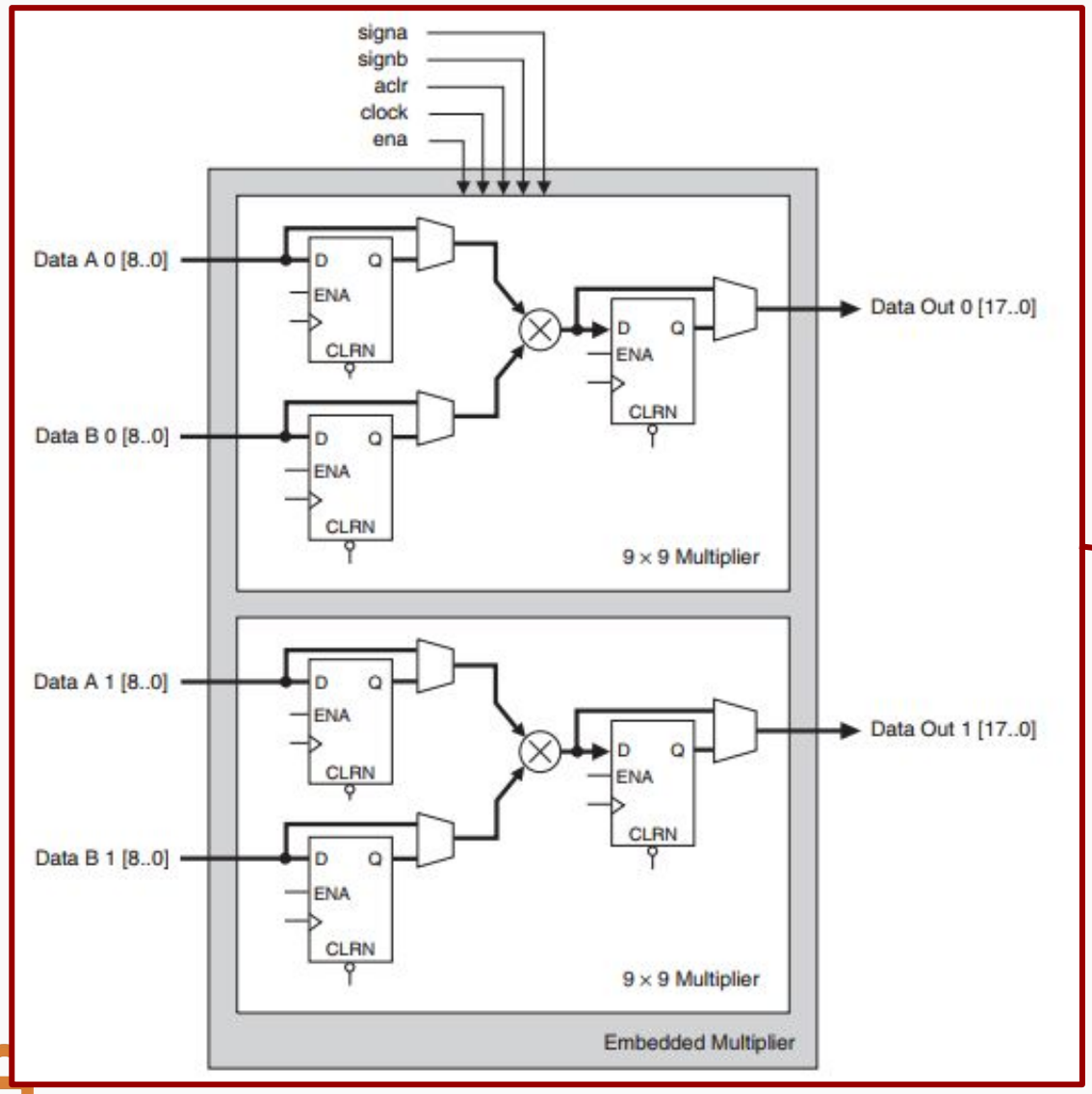
Filtros FIR: Reducción de multiplicadores

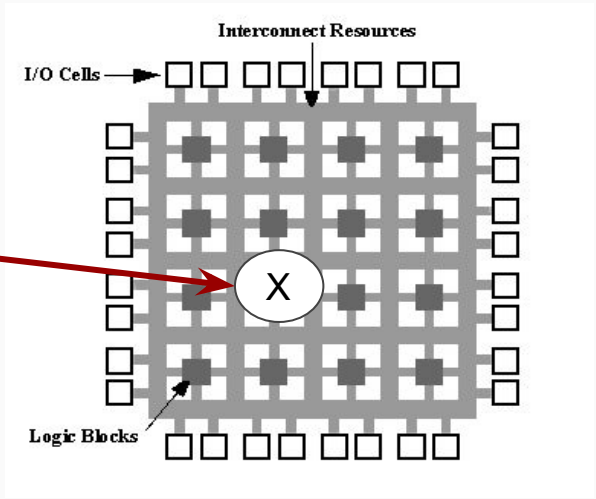


Lo mismo se hace para la arquitectura serie transpuesta

Implementación: SMACD & SMACI

Filtros FIR: Uso de multiplicadores embebidos





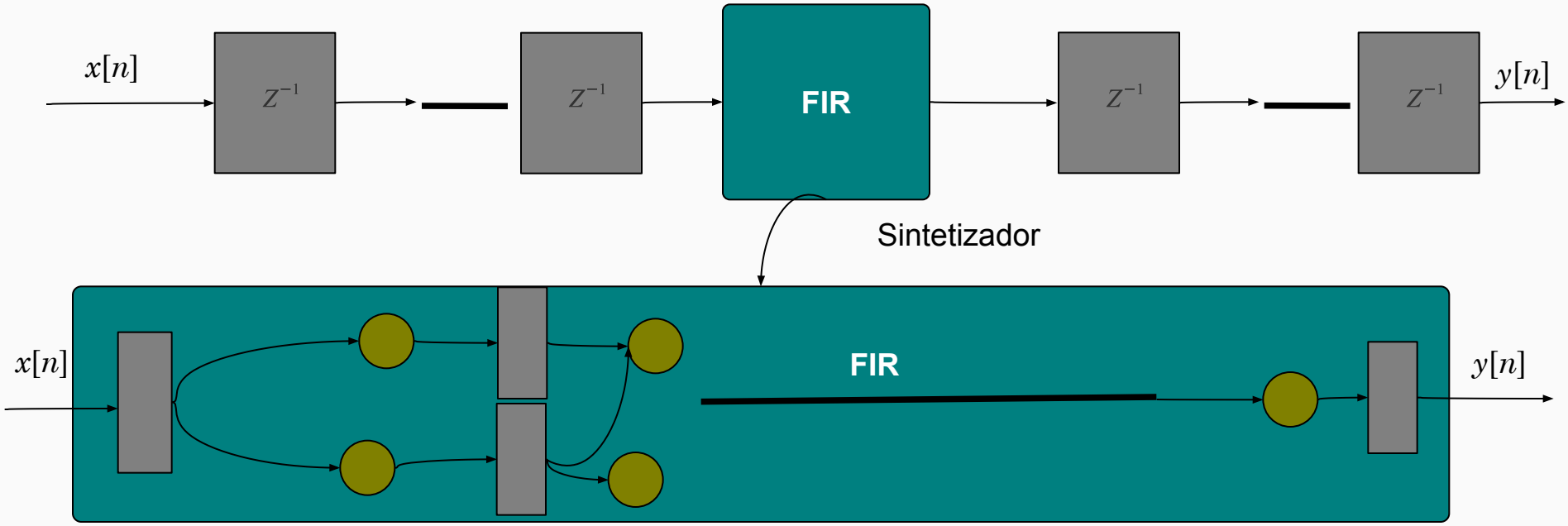
Implementación: con multiplicadores hardcore

Detalles para la Implementación

Concepto de Re-timing

Intenta mejorar, de forma automática, el **timing** de un dispositivo en una FPGA

Consiste en introducir estados de pipeline en el dispositivo, pero la ubicación de los mismos es controlada por el sintetizador.



El punto óptimo se consigue cuando se parte al circuito con tantos pipeline hasta que la demora es la del sistema indivisible más lento

Conclusiones

- Las FPGA son una herramienta muy poderosa para el procesamiento de señales digitales
- El uso de herramientas en Python acelera y mejora la profundidad de testeo y automatización de los procesos (CoCoTB y HDLMake)
- La Cuantización de los coeficientes, puede traer problemas
- Las arquitecturas de filtros FIR tipo inversa son las más adecuadas para el trabajo con FPGA
- La técnica de re-timing es útil para aumentar el rendimiento del filtro
- Debe considerarse muy bien los tamaños de las señales intermedias
- La optimización de los multiplicadores tiene grandes beneficios:
  - Reducción del consumo de primitivas/macros
  - Aumento de la frecuencia de operación por reducción de la latencia

- **Desarrollo con FPGAs en GNU/Linux**, Ing. S. Tropea, Ing. D. Brengi e Ing. R. Melo, *SASE 2011*l
- **Implementación de DSP en FPGAs,** Ing. M. Cervetto, Ing. E. Marchi, *SASE 2012*
- **COCOTB:** http://cocotb.readthedocs.io/en/latest/
- **HDLMake:** https://www.ohwr.org/projects/hdl-make
- **GTKWave**: http://gtkwave.sourceforge.net/
- **GHDL**: http://ghdl.free.fr/