XADC

Tyrone Novillo
Facultad de ingeniería
Universidad de Cuenca
Cuenca, Ecuador
tyrone.novillo@ucuenca.edu.ec

***Resumen*—This report presents the implementation and configuration of the XADC (Xilinx Analog-to-Digital Converter) on the Basys 3 board. The XADC enables the conversion of analog signals into digital data, facilitating the integration of real-world signals into FPGA projects. The setup involves configuring various parameters in Vivado's LogicCORE IP, followed by integrating the XADC component into the main program. The advantages of using the pre-built XADC over custom-built ADCs are highlighted, emphasizing its high precision and ease of use. This implementation serves as a robust foundation for developing complex digital systems requiring analog signal processing.**

***Keywords*—** XADC, Analog-to-Digital Converter, Basys 3, FPGA, Vivado, signal conversion.

I. Introducción

Un Conversor Analógico-Digital (ADC, por sus siglas en inglés) es un dispositivo que transforma señales analógicas continuas en señales digitales discretas, permitiendo que sistemas digitales, como las FPGAs, puedan procesar y manipular datos del mundo real. Los ADCs son componentes fundamentales en una variedad de aplicaciones, incluyendo sistemas de control, procesamiento de señales y comunicaciones, donde es crucial convertir señales del mundo físico, como temperatura, presión o sonido, en datos digitales que puedan ser analizados y utilizados por microcontroladores y otros circuitos digitales.

La tarjeta Basys 3, desarrollada por Digilent, integra un ADC de alta precisión conocido como XADC (Xilinx Analog-to-Digital Converter). Este conversor está diseñado específicamente para funcionar con las FPGAs de Xilinx, proporcionando una solución eficiente y de alto rendimiento para la conversión de señales analógicas. El XADC incluye múltiples canales de entrada, tanto diferenciales como de un solo extremo, permitiendo la captura simultánea de varias señales. Además, ofrece características avanzadas como la detección de alarmas y la monitorización de temperaturas internas, lo que lo convierte en una herramienta poderosa para el desarrollo de aplicaciones digitales que requieren la integración de señales analógicas. En este documento, se detalla el proceso de configuración y uso del XADC en la Basys 3.

II. Desarrollo

Se requiere hacer uso del conversor anaógico digital proporcionado por la tarjeta Basys 3. Para ellos se utilizará el conversor que viene por defecto en la FPGA. Para esto se debe configurar desde el setup wizard que tiene integrado Vivado.

Para configurar el XADC, se deben seguir los siguientes pasos: Primero dirigirse al menú de *LogicCORE IP* (Propiedad intelectual) proporcionado por Vivado, como muestra la Figura 1

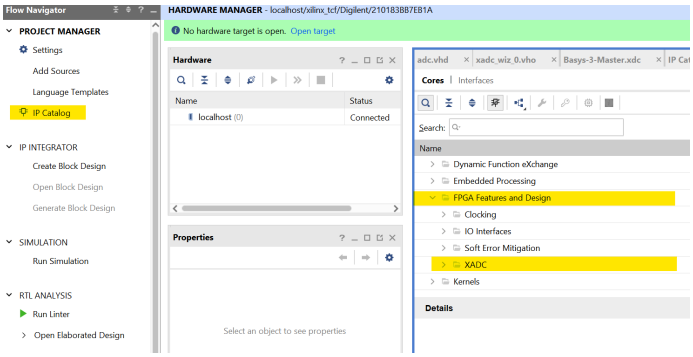

Figura 1: Menú del LogicCORE IP

Una vez en el wizard, se deben colocar los siguientes datos:

- **Interface options:** DRP
- **Timing mode:** Event mode
- **Startup Channel Selection:** Single Channel
- **DCLK Frequency:** 100MHz
- **Control/Status Port:** reset in
- **Event mode Trigger:** convst_in
- **Sequencer mode:** off
- **Todas las alarmas apagadas**
- **Selected Channel:** VAUXP5 VAUXN5
- **Bipolar:** off

Una vez configurados estos parámetros, el componente que proporciona Vivado es uno como de la Figura 2

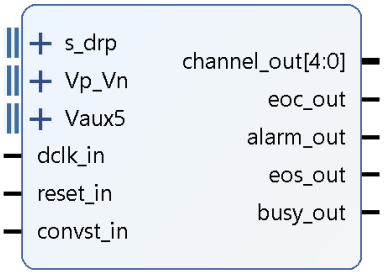

Figura 2: Componente ADC

A continuación, desde el programa principal, se debe llamar a este componente de la siguiente manera:

Para ello, se utilizarán componentes para cada unidad, tanto la aritmética como la lógica. Para esta práctica se evitará el uso de la librería **ieee.std_logic_1164.all**. Los componentes que se implementarán serán los siguientes:

- Componente para dividir frecuencia definido en prácticas anteriores.

- Componente que describe la operación de la unidad lógica.
- Componente que describe la operación de la unidad aritmética.
- Componente que define la suma de dos valores en binario.
- Componente de multiplexación de unidades.

Este documento describe un módulo llamado `adc`, diseñado para ser implementado en una tarjeta Basys 3, utilizando un conversor analógico-digital XADC (Xilinx Analog-to-Digital Converter). El diseño se estructura en una entidad y una arquitectura, donde se declaran y configuran las señales necesarias para el funcionamiento del XADC.

III. Entidad `adc`

La entidad `adc` define las entradas y salidas del módulo. Tiene las siguientes señales:

- `clk`: Señal de entrada de reloj (`STD_LOGIC`).
- `sw`: Señal de entrada de conmutador (`STD_LOGIC`).
- `led`: Señal de salida para controlar un conjunto de LEDs (`STD_LOGIC_VECTOR (15 downto 0)`).
- `JA`: Señal de entrada que recibe datos desde el conector JA de la tarjeta (`STD_LOGIC_VECTOR (7 downto 0)`).

III-A. Arquitectura `Behavioral` de `adc`

Dentro de la arquitectura `Behavioral`, se declara un componente llamado `xadc_wiz_0`, que representa el módulo XADC proporcionado por Xilinx. Este componente tiene varias señales de entrada y salida:

III-B. Señales del Componente `xadc_wiz_0`

- `di_in`: Entrada de datos (`STD_LOGIC_VECTOR(15 DOWNTO 0)`).
- `daddr_in`: Dirección de entrada (`STD_LOGIC_VECTOR(6 DOWNTO 0)`).
- `den_in`: Habilitador de entrada (`STD_LOGIC`).
- `dwe_in`: Habilitador de escritura (`STD_LOGIC`).
- `drdy_out`: Señal de salida lista para datos (`STD_LOGIC`).
- `do_out`: Datos de salida (`STD_LOGIC_VECTOR(15 DOWNTO 0)`).
- `dclk_in`: Señal de entrada de reloj (`STD_LOGIC`).
- `reset_in`: Señal de entrada de reinicio (`STD_LOGIC`).
- `convst_in`: Señal de inicio de conversión (`STD_LOGIC`).
- `vp_in`: Entrada positiva del canal analógico (`STD_LOGIC`).
- `vn_in`: Entrada negativa del canal analógico (`STD_LOGIC`).
- `vauxp5`: Entrada auxiliar positiva del canal 5 (`STD_LOGIC`).
- `vauxn5`: Entrada auxiliar negativa del canal 5 (`STD_LOGIC`).
- `channel_out`: Salida del canal (`STD_LOGIC_VECTOR(4 DOWNTO 0)`).
- `eoc_out`: Señal de fin de conversión (`STD_LOGIC`).
- `alarm_out`: Señal de alarma (`STD_LOGIC`).
- `eos_out`: Señal de fin de secuencia (`STD_LOGIC`).
- `busy_out`: Señal de ocupado (`STD_LOGIC`).

III-C. Señales Internas y Proceso de Control

Además de las señales del componente, se declaran varias señales internas:

- `channel_out`: Señal para el canal de salida (`STD_LOGIC_VECTOR(4 DOWNTO 0)`).
- `daddr_in`: Dirección de entrada (`STD_LOGIC_VECTOR(6 DOWNTO 0)`).
- `eoc_out`: Señal de fin de conversión (`STD_LOGIC`).
- `do_out`: Datos de salida (`STD_LOGIC_VECTOR(15 DOWNTO 0)`).
- `anal_p` y `anal_n`: Señales analógicas positivas y negativas (`STD_LOGIC`).
- `count`: Contador para el proceso de control (`INTEGER`).
- `convst`: Señal de inicio de conversión (`STD_LOGIC`).

III-D. Mapeo de Señales

El mapeo de señales conecta las entradas y salidas del componente `xadc_wiz_0` con las señales internas y externas definidas en la entidad y arquitectura. Por ejemplo, `di_in` se fija en un valor constante, `daddr_in` se asigna a `daddr_in`, y así sucesivamente.

III-E. Proceso de Control

Se define un proceso para controlar la señal de inicio de conversión (`convst`). Este proceso se ejecuta en cada flanco ascendente del reloj (`clk`). Incrementa el contador y, cuando el contador alcanza 999, se reinicia a 0 y activa la señal `convst` a '1' para iniciar una nueva conversión. Este proceso asegura que la conversión se realice a una frecuencia de muestreo de 100 kHz, dado un reloj de 100 MHz.

III-F. Constraints

En las restricciones (.xdc) se define el pin físico W5 para la señal de reloj (`clk`) con el estándar de señal LVCMOS33. Además, se crea un reloj con un periodo de 10.00 ns. Se define el pin físico V17 para el primer switch (`sw`). Los demás switches están comentados pero se pueden definir de manera similar descomentando las líneas correspondientes. También se definen los pines físicos U16 y E19 para los primeros dos LEDs (`led[0]` y `led[1]`). Se definen los pines físicos J1 y L2 para los primeros dos pines del conector Pmod JA (`JA[0]` y `JA[1]`), estas son las entradas de la señal a convertir que son entradas diferenciales positivas del ADC.

IV. Conclusiones

El uso del XADC proporcionado por la tarjeta Basys 3 presenta múltiples ventajas sobre la implementación manual de un conversor analógico-digital. Este módulo preconfigurado y optimizado por Xilinx garantiza un alto rendimiento y precisión en la conversión de señales analógicas a digitales, reduciendo significativamente el tiempo de desarrollo y minimizando errores. El XADC facilita la integración con otros componentes del sistema, permitiendo una configuración rápida a través del wizard de Vivado. En cuanto al funcionamiento del ADC, este componente convierte señales analógicas en digitales al muestrear y cuantificar las entradas analógicas en valores binarios, procesando las señales a una frecuencia de muestreo determinada. La configuración y mapeo de señales, junto con el control preciso del inicio de la conversión, aseguran que las lecturas digitales reflejen fielmente las variaciones en las señales analógicas.

V. Anexos

V-A. Archivo leccion2.vhd

```
----------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 07/10/2024 02:17:44 PM
-- Design Name:
-- Module Name: adc - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------


library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity adc is
  Port (
  clk : in STD_LOGIC;
  sw : in std_logic;
  led : out STD_LOGIC_VECTOR (15 downto 0);
  JA : in STD_LOGIC_VECTOR (7 downto 0)
  );
end adc;

architecture Behavioral of adc is

COMPONENT xadc_wiz_0
  PORT (
    di_in : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    daddr_in : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    den_in : IN STD_LOGIC;
    dwe_in : IN STD_LOGIC;
    drdy_out : OUT STD_LOGIC;
    do_out : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
    dclk_in : IN STD_LOGIC;
    reset_in : IN STD_LOGIC;
    convst_in : IN STD_LOGIC;
    vp_in : IN STD_LOGIC;
    vn_in : IN STD_LOGIC;
    vauxp5 : IN STD_LOGIC;
    vauxn5 : IN STD_LOGIC;
    channel_out : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
    eoc_out : OUT STD_LOGIC;
    alarm_out : OUT STD_LOGIC;
    eos_out : OUT STD_LOGIC;
    busy_out : OUT STD_LOGIC
  );
END COMPONENT;

signal channel_out : std_logic_vector(4 downto 0);
signal daddr_in  : std_logic_vector(6 downto 0);
signal eoc_out : std_logic;
signal do_out  : std_logic_vector(15 downto 0);
signal anal_p, anal_n : std_logic;

signal count: integer;
signal convst: std_logic;

begin

conversor : xadc_wiz_0
  PORT MAP (
    di_in => "0000000000000000",
    daddr_in => daddr_in,
    den_in => eoc_out,
    dwe_in => '0',
    drdy_out => open,
    do_out => do_out,
    dclk_in => clk,
    reset_in => sw,
    convst_in => convst,
    vp_in => '0',
    vn_in => '0',
    vauxp5 => anal_p,
    vauxn5 => anal_n,
    channel_out => channel_out,
    eoc_out => eoc_out,
    alarm_out => open,
    eos_out => open,
    busy_out => open
  );

 daddr_in <= "00" & channel_out;
 anal_p <= JA(4);
 anal_n <= JA(0);
 led <= do_out;

   -- for a sampling freq of 100 kHz, with a clk freq
      process(clk)
      begin
          if rising_edge(clk) then
              count <= count + 1;
              convst <= '0';
              if count = 999 then
                  count <= 0;
                  convst <= '1';
              end if;
          end if;
      end process;


end Behavioral;
```

V-B. Archivo de Constraints Basys-3-Master.xdc

```
#Clock signal
set_property -dict { PACKAGE_PIN W5
↪  IOSTANDARD LVCMOS33 } [get_ports clk]
#create_clock -add -name sys_clk_pin -period
↪  10.00 -waveform {0 5} [get_ports clk]


# Switches
set_property -dict { PACKAGE_PIN V17
↪  IOSTANDARD LVCMOS33 } [get_ports {sw}]

# LEDs
set_property -dict { PACKAGE_PIN U16
↪  IOSTANDARD LVCMOS33 } [get_ports {led[0]}]
set_property -dict { PACKAGE_PIN E19
↪  IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
set_property -dict { PACKAGE_PIN U19
↪  IOSTANDARD LVCMOS33 } [get_ports {led[2]}]
set_property -dict { PACKAGE_PIN V19
↪  IOSTANDARD LVCMOS33 } [get_ports {led[3]}]
set_property -dict { PACKAGE_PIN W18
↪  IOSTANDARD LVCMOS33 } [get_ports {led[4]}]
set_property -dict { PACKAGE_PIN U15
↪  IOSTANDARD LVCMOS33 } [get_ports {led[5]}]
set_property -dict { PACKAGE_PIN U14
↪  IOSTANDARD LVCMOS33 } [get_ports {led[6]}]
set_property -dict { PACKAGE_PIN V14
↪  IOSTANDARD LVCMOS33 } [get_ports {led[7]}]
set_property -dict { PACKAGE_PIN V13
↪  IOSTANDARD LVCMOS33 } [get_ports {led[8]}]
```

```
set_property -dict { PACKAGE_PIN V3
↪  IOSTANDARD LVCMOS33 } [get_ports {led[9]}]
set_property -dict { PACKAGE_PIN W3
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {led[10]}]
set_property -dict { PACKAGE_PIN U3
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {led[11]}]
set_property -dict { PACKAGE_PIN P3
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {led[12]}]
set_property -dict { PACKAGE_PIN N3
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {led[13]}]
set_property -dict { PACKAGE_PIN P1
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {led[14]}]
set_property -dict { PACKAGE_PIN L1
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {led[15]}]

#Pmod Header JA
set_property -dict { PACKAGE_PIN J1
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {JA[0]}];#Sch name = JA1
set_property -dict { PACKAGE_PIN L2
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {JA[1]}];#Sch name = JA2
set_property -dict { PACKAGE_PIN J2
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {JA[2]}];#Sch name = JA3
set_property -dict { PACKAGE_PIN G2
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {JA[3]}];#Sch name = JA4
set_property -dict { PACKAGE_PIN H1
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {JA[4]}];#Sch name = JA7
set_property -dict { PACKAGE_PIN K2
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {JA[5]}];#Sch name = JA8
set_property -dict { PACKAGE_PIN H2
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {JA[6]}];#Sch name = JA9
set_property -dict { PACKAGE_PIN G3
↪  IOSTANDARD LVCMOS33 } [get_ports
↪  {JA[7]}];#Sch name = JA10


## Configuration options, can be used for all
↪  designs
set_property CONFIG_VOLTAGE 3.3
↪  [current_design]
set_property CFGBVS VCCO [current_design]

## SPI configuration mode options for QSPI
↪  boot, can be used for all designs
set_property BITSTREAM.GENERAL.COMPRESS TRUE
↪  [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33
↪  [current_design]
set_property CONFIG_MODE SPIx4
↪  [current_design]
```