Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica
IEE2463 Sistema electrónicos programables

Laboratorio

Protocolo AXI - Parte 1

---

1. Objetivo

El presente documento tiene como objetivo ayudar al estudiante a comprender el funcionamiento del protocolo AXIs y su implementación como protocolo de comunicación entre la PL (Programmable Logic) y la PS (Processing Subsystem)(ARM Cortex-A9).

2. Implementación PL-PS y bloque ADC (VIVADO).

Inicialmente nos enfocamos en usar solo el lado del sistema de procesamiento (PS) del Zynq. El beneficio real de usar un dispositivo como el Zynq SoC, viene en la creación de un sistema programable que también utiliza el lado de la lógica programable (PL) del dispositivo y las macros dedicadas de IP dura en el Zynq SoC, incluido el subsistema analógico XADC. , los enlaces serie de alta velocidad (SerDes) y los puntos finales PCIe.

Para este laboratorio, iniciaremos utilizando el XADC para monitorear una cantidad de parámetros internos del dispositivo, para verificar la salud de su diseño. Además, para facilitar la verificación durante las primeras etapas dentro de un sistema basado en SoC Zynq, puede usar el XADC para medir la temperatura registrada por el sensor de temperatura en el chip, junto con los siguientes parámetros adicionales:

- VCCInt: El voltaje interno del núcleo PL.
- VCCAux: La tensión PL auxiliar.
- VRefP: El voltaje de referencia positivo XADC.
- VRefN: El voltaje de referencia negativo XADC.
- VCCBram: El voltaje PL BRAM.
- VCCPInt: El voltaje del núcleo interno del PS.
- VCCPaux: La tensión auxiliar PS.

- VCCDdr: El voltaje de funcionamiento de la RAM DDR conectada a la PS.

Lo primero que debe hacer, es integrar el ZYNQ7 y asegurarse de haber habilitado una de las interfaces AXI maestras de propósito general dentro del Zynq PS en la página de configuración de PS-PL dentro del ZYNQ7, figura 1.

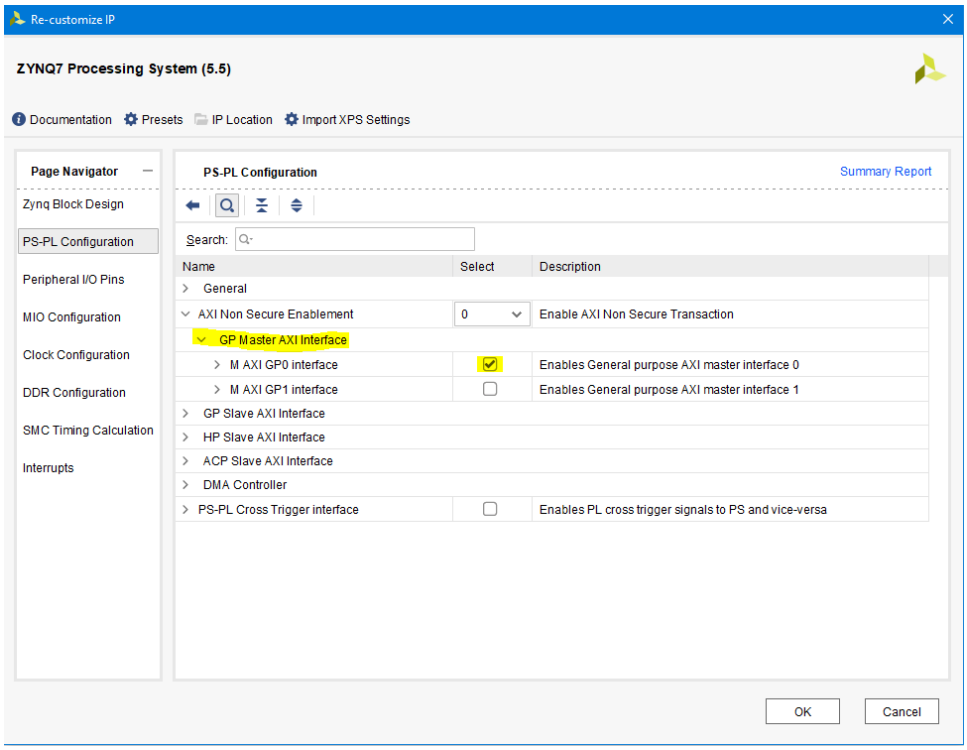

Figura 1: Configuración y habilitación de AXI Master.

Luego de agregar el ZYNQ7, lo que debe hacer es conectar este puerto AXI del ZYNQ7 dentro del PS a un bloque de interconexión AXI. Hacer esto permite que Zynq PS que se conecte a la interfaz AXI 4 lite de XADC. Agregue el bloque de interconexión AXI a su diagrama de bloques desde el catálogo IP de Vivado. Una vez que lo haya agregado al diagrama de bloques, conecte el puerto AXI master del PS al puerto esclavo de interconexión AXI. Conecte FCLK_CLK0 del PS al S00_ACLK y al ack maestro de la interconexión AXI. Del mismo modo, conecte los resets como se muestra en el diagrama en la figura 2.

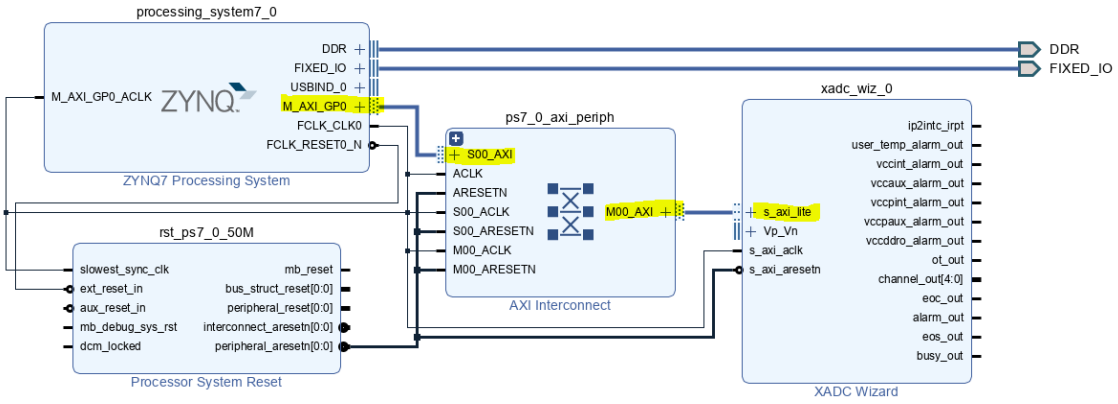

Figura 2: Visualización de conexión AXI master y AXI_S en diagrama de bloques.

Una vez que la interconexión AXI está conectada al PS, podemos personalizar la interconexión AXI para seleccionar la cantidad de puertos esclavos y maestros. Haga clic en el módulo de interconexión AXI y podrá seleccionar la cantidad de interfaces maestras y esclavas. En este ejemplo, seleccioné uno de cada uno.

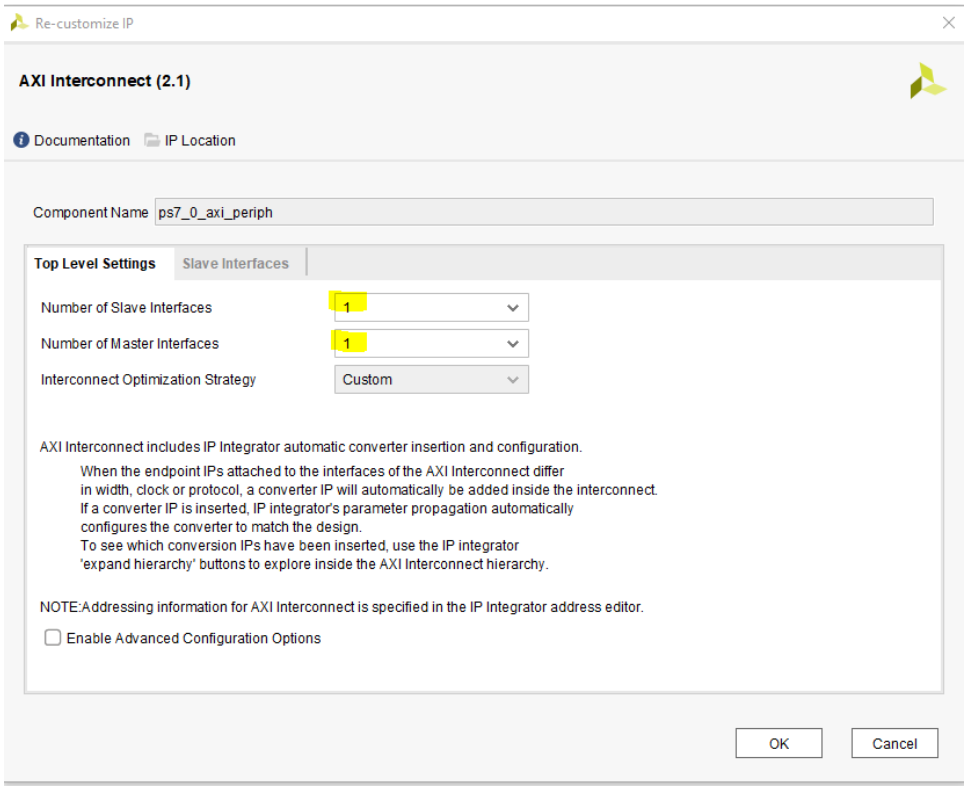

Figura 3: Configuración de AXI Interconnect.

Una vez que esto esté completo, puede incluir el XADC agregándolo desde el catálogo

de IP. Una vez que coloca el XADC en su diseño (como en la figura 2, puede personalizarlo para conectarse a través de la interfaz AXI4 usando el asistente XADC. Tenga en cuenta que en este ejemplo estaremos leyendo los voltajes y temperaturas internas del Zynq SoC. La configuración debe ser como en la figura 4.

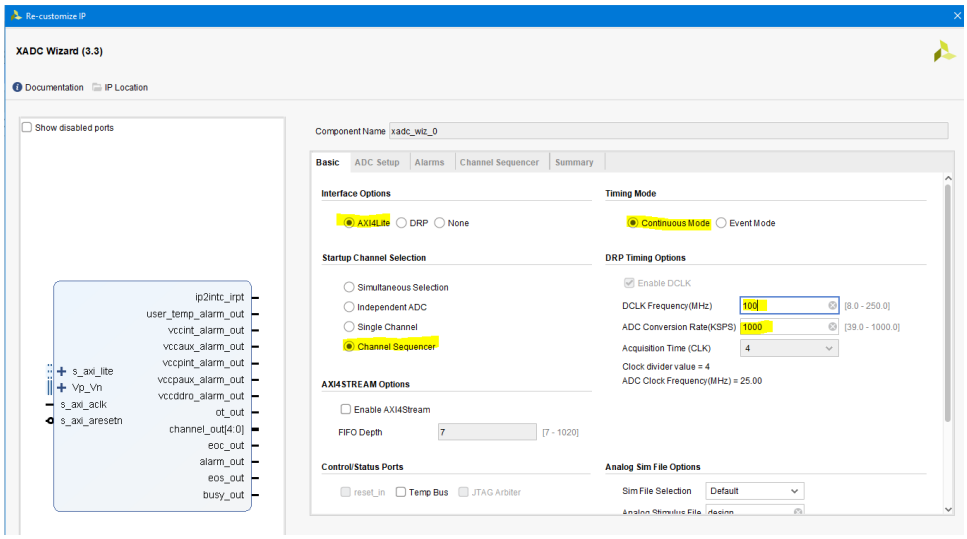

Figura 4: Configuración de AXI ADC.

En el caso de realizar el proceso de manera automática, ingrese los bloques axi interconnect y un bloque IP XADC Wizard, como se presenta en la figura 5, vaya al item Run connection Automation, seleccione todas las casillas y presione ok. Por defecto se integrará un nuevo bloque llamado Processor System Reset. Para realizar dicho proceso, procure mantener las configuraciones de bloque de las figuras; Fig. 3 y Fig. 4.

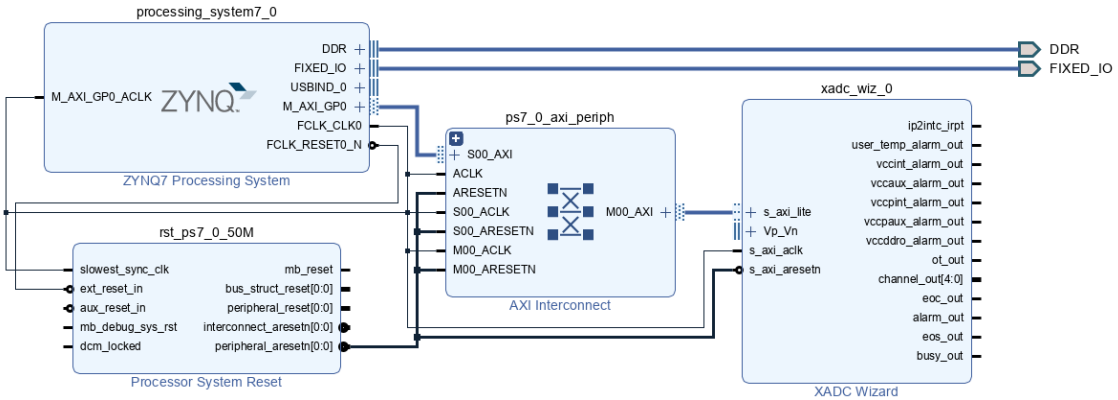

Figura 5: Diagrama de bloques del sistema diseñado.

Finalmente, valide el diseño, cree el HDL Wrapper sobre el diseño, aplique la función

Run Implementation y Generate bitstream. Una vez que haya compilado todo sin errores, vaya al menu de configuración File y seleccione Export Hardware.

3. Implementación en Vitis.

Abra el software Xilinx Vitis, seleccione la carpeta donde implementará el proyecto y genere una nuevo proyecto de aplicación. Por consiguiente, utilice el mismo metodo de los laboratorios anteriores e integre el hardware (archivo .xsa) en la nueva app.

Posteriormente, puede seleccionar el archivo de programa por defecto helloworld.c; ya que en dicho archivo pegaremos el código de programa a implementar. Una vez dentro del proyecto, debería visualizarse de la siguiente manera:

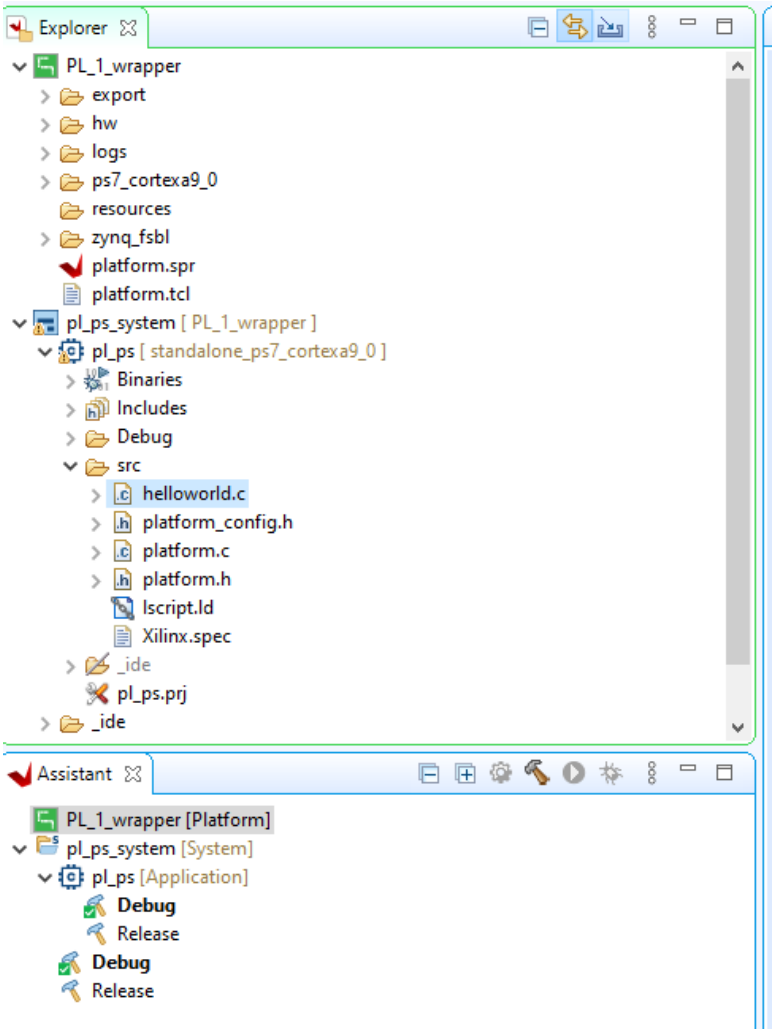

Figura 6: Archivos de programas.

En el archivo helloworld.c elimine el código e incluya el siguiente programa:

```
/*
 * Copyright (c) 2009 Xilinx, Inc.  All rights reserved.
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
 * COURTESY TO YOU.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
 * ONE POSSIBLE   IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
 * STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
 * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION.
 * XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO
 * THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO
 * ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE
 * FROM CLAIMS OF INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 * helloworld.c: simple test application
 */

#include <stdio.h>
#include "platform.h"
#include "xadcps.h"
#include "xil_types.h"
#define XPAR_AXI_XADC_0_DEVICE_ID 0

//void print(char *str);

static XAdcPs XADCMonInst;
int main()
{
XAdcPs_Config *ConfigPtr;
XAdcPs *XADCInstPtr = &XADCMonInst;

//status of initialisation
int Status_ADC;

//temperature readings
u32 TempRawData;
float TempData;
```

```
//Vcc Int readings
u32 VccIntRawData;
float VccIntData;

//Vcc Aux readings
u32 VccAuxRawData;
float VccAuxData;

//Vbram readings
u32 VBramRawData;
float VBramData;

//VccPInt readings
u32 VccPIntRawData;
float VccPIntData;

//VccPAux readings
u32 VccPAuxRawData;
float VccPAuxData;

//Vddr readings
u32 VDDRRawData;
float VDDRData;

init_platform();

    printf("Adam Edition MicroZed Using Vivado How To Printf \n\r");

    //XADC initilization

     ConfigPtr = XAdcPs_LookupConfig(XPAR_AXI_XADC_0_DEVICE_ID);
      if (ConfigPtr == NULL) {
      return XST_FAILURE;
      }

     Status_ADC = XAdcPs_CfgInitialize(XADCInstPtr,ConfigPtr,ConfigPtr->BaseAddress);
     if(XST_SUCCESS != Status_ADC){
         print("ADC INIT FAILED\n\r");
         return XST_FAILURE;
      }

     //self test
```

```
   Status_ADC = XAdcPs_SelfTest(XADCInstPtr);
if (Status_ADC != XST_SUCCESS) {
return XST_FAILURE;
}

//stop sequencer
XAdcPs_SetSequencerMode(XADCInstPtr,XADCPS_SEQ_MODE_SINGCHAN);

   //disable alarms
   XAdcPs_SetAlarmEnables(XADCInstPtr, 0x0);

   //configure sequencer to just sample internal on chip parameters
   XAdcPs_SetSeqInputMode(XADCInstPtr, XADCPS_SEQ_MODE_SAFE);

   //configure the channel enables we want to monitor
   XAdcPs_SetSeqChEnables(XADCInstPtr,XADCPS_CH_TEMP|XADCPS_CH_VCCINT|XADCPS_CH_VCCAUX
    XADCPS_CH_VCCPAUX|XADCPS_CH_VCCPDRO);

   while(1){
   TempRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_TEMP);
    TempData = XAdcPs_RawToTemperature(TempRawData);
    printf("Raw Temp %lu Real Temp %f \n\r", TempRawData, TempData);

    VccIntRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCINT);
    VccIntData = XAdcPs_RawToVoltage(VccIntRawData);
    printf("Raw VccInt %lu Real VccInt %f \n\r", VccIntRawData, VccIntData);

    VccAuxRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCAUX);
    VccAuxData = XAdcPs_RawToVoltage(VccAuxRawData);
    printf("Raw VccAux %lu Real VccAux %f \n\r", VccAuxRawData, VccAuxData);

//    VrefPRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VREFP);
//    VrefPData = XAdcPs_RawToVoltage(VrefPRawData);
//    printf("Raw VRefP %lu Real VRefP %f \n\r", VrefPRawData, VrefPData);

//    VrefNRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VREFN);
//    VrefNData = XAdcPs_RawToVoltage(VrefNRawData);
 //   printf("Raw VRefN %lu Real VRefN %f \n\r", VrefNRawData, VrefNData);

    VBramRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VBRAM);
    VBramData = XAdcPs_RawToVoltage(VBramRawData);
    printf("Raw VccBram %lu Real VccBram %f \n\r", VBramRawData, VBramData);
```

```
        VccPIntRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCPINT);
        VccPIntData = XAdcPs_RawToVoltage(VccPIntRawData);
        printf("Raw VccPInt %lu Real VccPInt %f \n\r", VccPIntRawData, VccPIntData);

        VccPAuxRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCPAUX);
        VccPAuxData = XAdcPs_RawToVoltage(VccPAuxRawData);
        printf("Raw VccPAux %lu Real VccPAux %f \n\r", VccPAuxRawData, VccPAuxData);

        VDDRRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCPDRO);
        VDDRData = XAdcPs_RawToVoltage(VDDRRawData);
        printf("Raw VccDDR %lu Real VccDDR %f \n\r", VDDRRawData, VDDRData);
       }
     return 0;
}
```

Finalmente, realice un build debug (martillo) del proyecto o app. creado y espere a que finalice sin errores.
Una vez que este todo ok con el programa, conecte el hardware al laptop y configure el puerto serial a una velocidad de 115200baud. Posteriormente en el software vitis, seleccione la opción run y escriba la aplicación sobre la zybo.

En la terminal del software utilizado, en este caso Tera Term, se debería comenzar a visualizar inmediatamente las variables medidas, como se puede apreciar en la figura 7

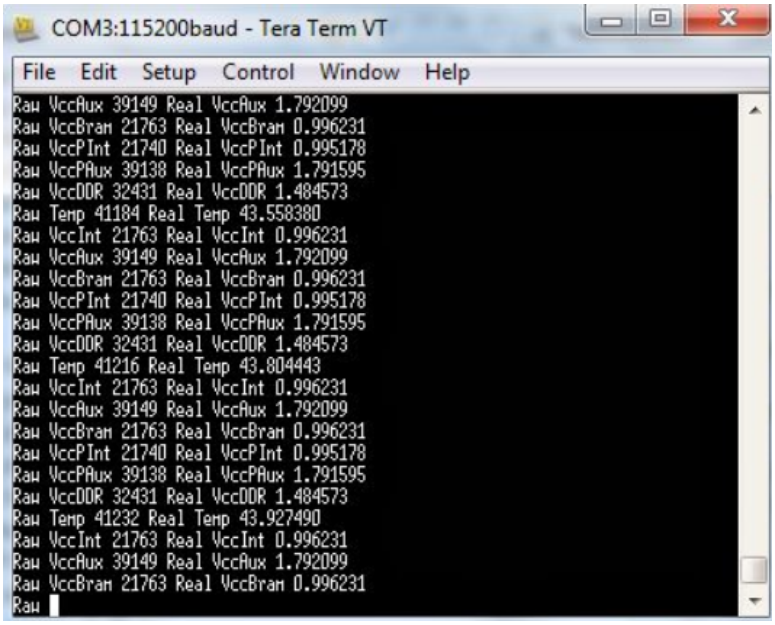

Figura 7: Variables medidas y presentadas en la terminal de Tera Term.

3.1. Explicación del código.

El encabezado xil_types.h define una serie de macros que puede usar para configurar, controlar y leer desde el XADC. Entre otras cosas, este archivo de encabezado contiene definiciones para los registros XADC, opciones de promedio de muestreo, opciones de secuencia de canales y modos de apagado. También contiene una serie de definiciones de tipos, macros y otras funciones.

Para el ejemplo, se leen los parámetros internos de temperatura y voltaje del Zynq SoC y enviarlos a través de un enlace RS232.

Lo primero que se debe hacer en el código es buscar la configuración del XADC a inicializar; esto requiere un puntero de tipo XAdcPs_Config. Usando la llamada de función XAdcPs_LookupConfig() junto con la ID del dispositivo, esta función devolverá 0 ya que solo hay un bloque XADC dentro del Zynq. La configuración (ID del dispositivo y dirección base del XADC que se está inicializando, vea xparameters.h para esto) se almacenará en el puntero. Si no se puede encontrar la configuración para el XADC con el ID del dispositivo, se devolverá null para permitir que se maneje el error.

El siguiente paso en el proceso de inicialización, es utilizar la información previamente obtenida y almacenada dentro del puntero de configuración, que requiere un puntero de tipo XAdcPs.

Llame al puntero de configuración ConfigPtr y a mi puntero de creación de instancias XADCInstPtr.

Habiendo inicializado el XADC, los siguientes pasos lo configuran para el ejemplo:

- Utilice la función XAdcPs_SelfTest() para realizar una auto comprobación para verificar que no haya problemas con el dispositivo. XAdcPs_SetSequencerMode() para evitar que el secuenciador realice su operación actual configurándolo en un solo canal.
- Use XAdcPs_SetAlarmEnables() para deshabilitar cualquier alarma que pueda configurarse.
- Utilice XAdcPs_SetSeqInputMode() para reiniciar el secuenciador con la secuencia deseada.
- Use XAdcPs_SetSeqChEnables() para configurar las habilitaciones para los canales que deseo muestrear.

Leer una muestra del XADC puede ser tan simple como llamar a la función XAdcPs_GetAdcData(). Para los parámetros internos de temperatura y voltaje, luego usé dos de las macros provistas, XAdcPs_RawToTemperature() y XAdcPs_RawToVoltage(), para convertir los valores XADC sin procesar en sus equivalentes de temperatura o voltaje del mundo real.

4. Implementación bloque AXI4.

Generando el mismo diagrama de bloques de la figura 5, implementaremos un nuevo bloque AXI. Para ello, en la barra de opciones superior seleccione tools y posteriormente Create and Package New IP.

Esto abrirá un cuadro de diálogo que le permitirá crear periféricos AXI4. La primera página real del cuadro de diálogo presenta una serie de opciones para crear un nuevo bloque de IP o convertir su diseño actual o un directorio en un módulo de IP. En la primera pestaña informativa presione next y luego seleccione la opción Create AXI4 Peripheral (Figura 8).

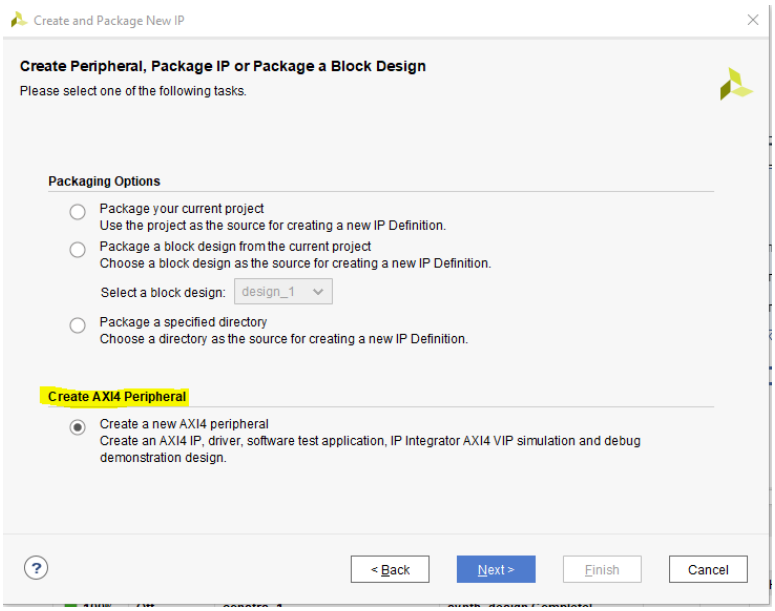

Figura 8: Opción crear un nuevo periférico AXI4.

En la pestaña Peripheral Details, puede dejar los valores por defecto, pero en este punto se ingresa el nombre y la descripción del nuevo periférico AXI4.

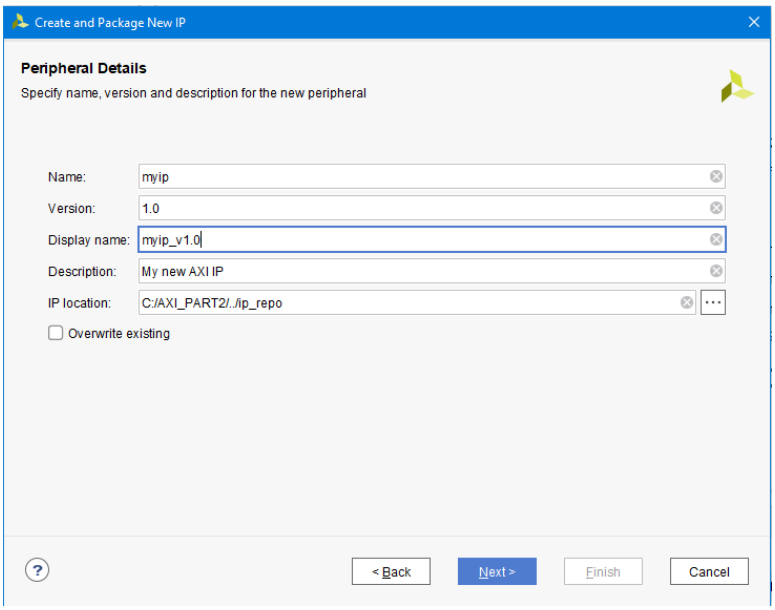

Figura 9: Descripción del nuevo periférico AXI4.

El cuadro de diálogo que sigue es la ventana principal donde podemos definir el tipo de interfaz AXI4 que deseamos especificar:

- Maestro o esclavo
- Tipo de interfaz: Lite, Streaming o Burst
- Ancho de bus 32 o 64 bits
- Tamaño de la memoria
- Número de registros

Este ejemplo inicial va a ser muy simple solo para que pueda demostrar el flujo necesario para crear el periférico, implementarlo dentro de Vivado y luego exportarlo a Vitis. Por esta razón, se usará una interfaz AXI4-Lite con solo cuatro registros que luego podemos abordar mediante software. Estos registros podrían usarse para controlar la operación de funciones dentro del lado de la lógica programable del diseño Figura 10.

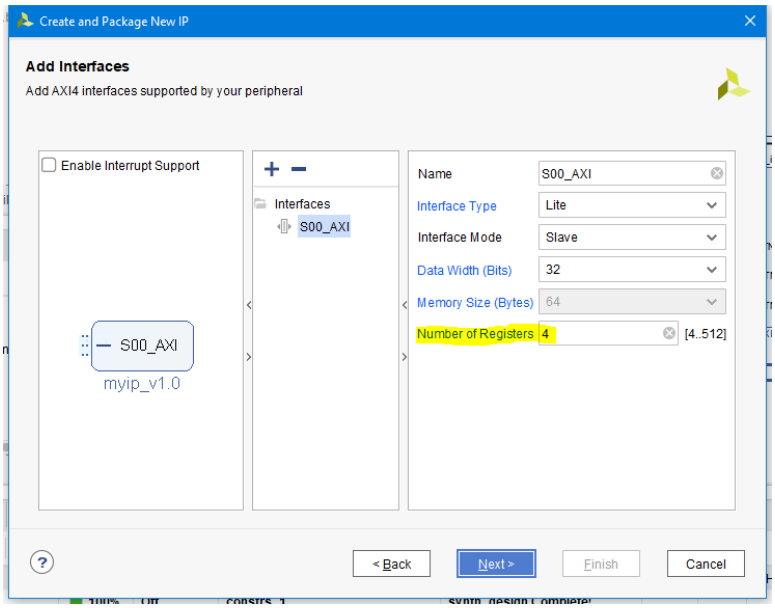

Figura 10: Configuración e interfaz del nuevo periférico AXI4.

Al terminar de configurar, solo debemos agregar la IP al repositorio y finalizar la creación del nuevo periférico.

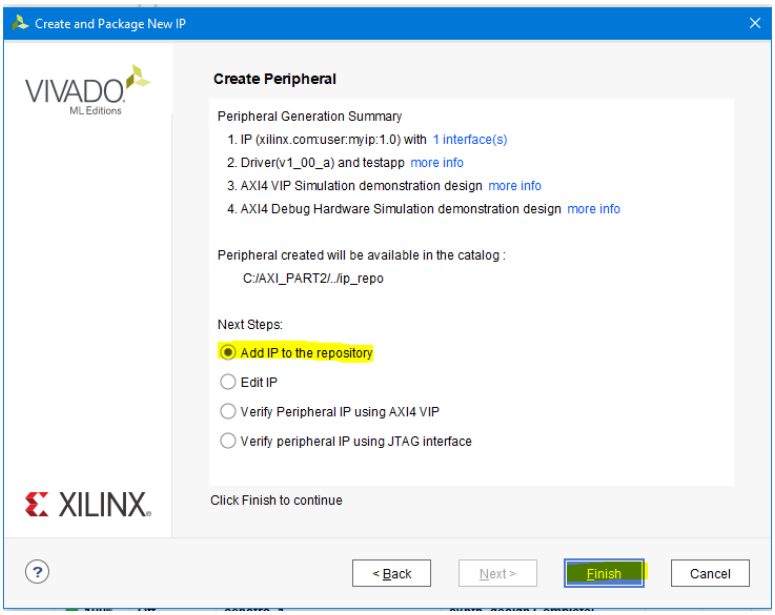

Figura 11: Termino de configuración de periférico AXI4.

Arrastre este bloque IP al diseño y luego conéctelo al bus AXI GP.

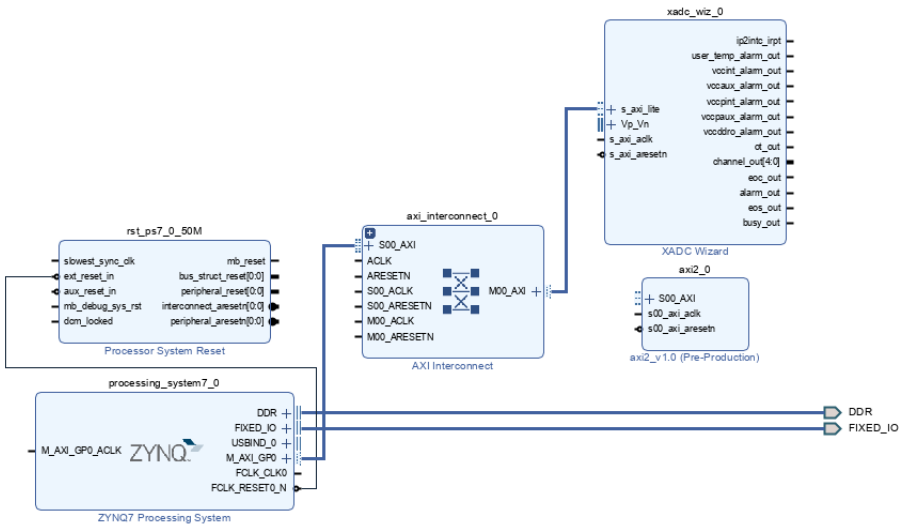

Figura 12: Nuevo bloque IP AXI4 en diagrama de bloques.

Vivado ofrece asistencia de diseño para conectar el nuevo periférico automáticamente. Puede ver esta asistencia en la barra verde que aparece en la parte superior de la imagen a continuación, donde Vivado ofrece ejecutar la herramienta de automatización de conexión. Ejecutar la herramienta rápidamente da como resultado un diseño que podemos implementar (Figura 13.

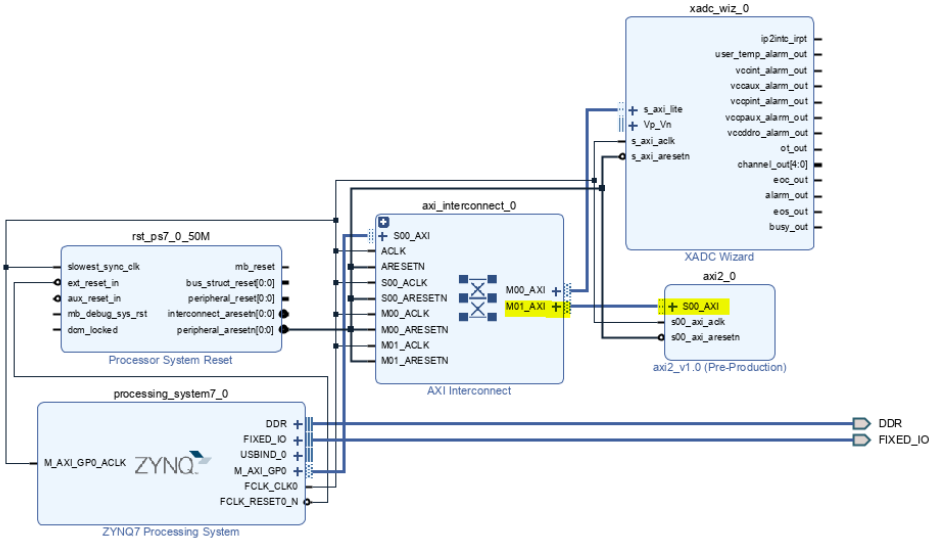

Figura 13: Diagrama de bloques final.

Una vez terminada la configuración del bloque AXI4 e integrada al diagrama del proyecto xADC, validaremos el proyecto, crearemos el HDL Wrapper sobre el diseño, aplicaremos Run

Implementation y finalmente Generate Bitstream.

Exporte el archivo .xsa y se procederá a la etapa 2 del proyecto AXI.