Diseño e Implementación de un servidor de base de datos utilizando la FPGA Terasic DE10-Standard

Lino Ontano
Telematics
Guayaquil - Ecuador
lontano@espol.edu.ec

Jocelyn Miranda
Telematics
Guayaquil - Ecuador
jocammir@espol.edu.ec

***Resumen*—Diseño e implementación de un sistema que realiza el monitoreo a una base de datos implementada en el sistema operativo Linux del Hard Processor System (HPS) de la FPGA Terasic DE10-Standard y es mostrada por pantalla utilizando protocolo VGA.**

***Index Terms*—FPGA, HPS, VGA**

I. Introducción

El presente proyecto se basa en el diseño y la implementación de un sistema embebido que realice monitoreo a una base de datos. El servidor de la base de datos se lo implementará en el Hard Processor System (HPS) de la FPGA Terasic DE-10 Standard, la cuál será manejada vía Putty SSH conectada a una red local. La FPGA será la encargada de realizar las consultas por medio de sus periféricos de entrada al HPS el cual ejecutará scripts que devolverán valores y serán escritos en memoria para que la FPGA lea y muestre el resultado por pantalla por medio de su puerto VGA. El proyecto consta de tres etapas:

1. Montar el servidor de la base de datos en el HPS, habilitar el acceso vía SSH y configurar las interfaces de red para el acceso en la red local. La imagen del sistema operativo es proporcionada por la tarjeta desde el sitio web de Terasic.
2. Diseñar e implementar la arquitectura de la FPGA donde el procesador Nios II enviará datos para ser mostrados por medio del VGA. Además de proporcionar acceso a los periféricos de entrada, para nuestro caso, habilitar acceso a los *switches* que puedan ser leídos por el procesador y acceso a la *OnChip Memory* de la FPGA, para leer los valores que serán enviados por el HPS en la siguiente etapa.
3. Finalmente, la arquitectura desarrollada deberá ser levantada por el HPS, para ello, se utilizará el proyecto de demostración Control Panel QT y se modificará dicha arquitectura y se agregará los bloques que requerimos. Se utiliza ese proyecto ya que contiene lo necesario para el correcto funcionamiento del HPS junto a la FPGA

La primera etapa constará de montar el servidor de la base de datos en la HPS y poder acceder a ella vía Putty SSH, la imagen del sistema operativo será descargada desde la página de Terasic 1 y donde se levantará el servicio de base de datos MySQL. La segunda etapa será poder mostrar al servidor por pantalla utilizando la FPGA, donde el procesador NIOS manejará las respuestas del servidor para ser mostradas y procesadas. Y la última etapa será de poder, desde la FPGA, solicitar consultas al HPS y recibir las mismas respuestas, es decir una comunicación bidireccional.

II. Procedimiento

Para la interacción entre el Hard-Processor System (HPS) y la FPGA se utiliza el puente bidireccional que permite la comunicación entre ambas, de tal forma la FPGA actue como cliente del servidor de base de datos desarrollado en el HPS.

The core functions performed within the network device and the QoS behaviors they can provide are:

- Classifiers
- Policers
- Shapers
- Queues
- Schedulers

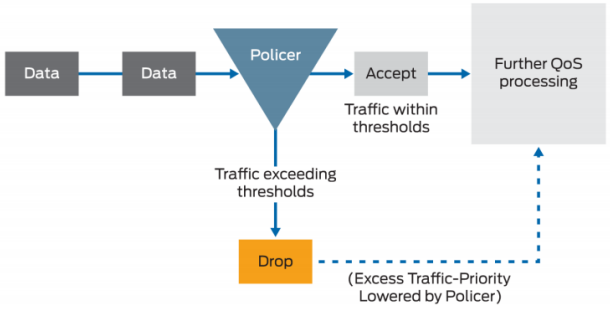


Figura 1: QoS Policers

In this project the function of the *policer* will be used intensively.
Policers control traffic bursts by ensuring that incoming or outgoing traffic conforms to a configured rate called the bandwidth limit. Policers provide the first line of congestion management by preventing incoming traffic from overloading the network. Figure 1 illustrates how policers manage traffic. In many QoS implementations, policers work in conjunction with *metering* and *color marking* tools to increase granularity.

Metering measures the traffic arrival rate and assigns different colors to the traffic according to that rate. Metering works by comparing the actual rate of the traffic with the following two configurable values:

- Committed information rate (CIR): This is the guaranteed rate.
- Peak information rate (PIR): This is te maximum allowed traffic rate.

Metering measures the traffic comparing these two values and marks the traffic with colors that identify whether the traffic is *in-contract* or *out-of-contract* as follows:

- Green: Traffic rate is below the CIR and is in-contract.
- Yellow: Traffic rate falls between the CIR and PIR and is out-of-contract.
- Red: Traffic is above the PIR and is out-of-contract.

You can configure whether you want the device to forward the traffic or discard it based on the color.
Metering has one input, which is the traffic arrival rate, and it has three possible outputs: green, yellow, or red.

In this project we represent the value of the PIR as a constant defined in our script. The measurements we make on our four sensors cannot reach that value, if they do, the *policer* will discard those values.

III. Implementation Design

III-A. ***Technology:***

To achieve the simulation of the *policer*, we will use an *Arduino UNO* with a group of sensors: two temperature sensors and two potentiometers that will give variable values to appreciate the work of the *policer*. The Arduino will process the reading of the sensors and display them in real time on graphs where it will be appreciated when a sensor reaches the PIR.

III-B. ***Topology:***

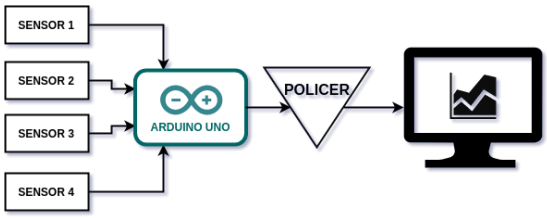


Figura 2: Topology

The sensor values are received by the Arduino and processed by a Python script through the serial port. The QoS parameters for metering and coloring are defined in the script. The sensor data is plotted by the same script in real time.

IV. Solution Development

The project consists of two parts, the data acquisition part and the processing through the policer.

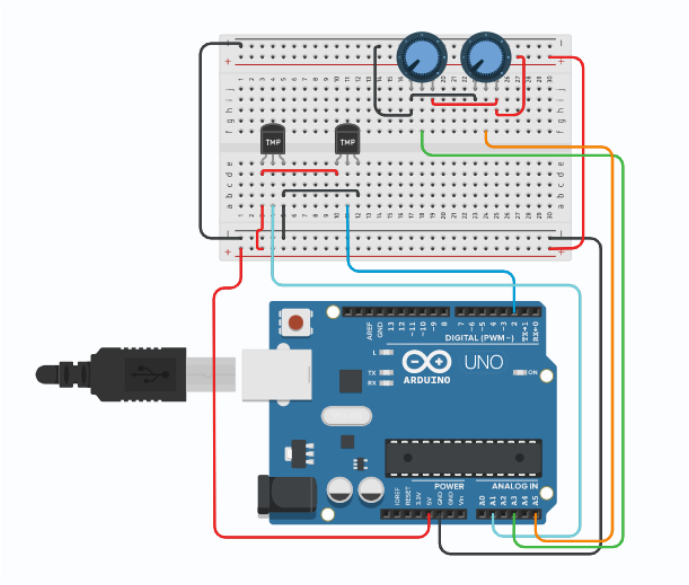


Figura 3: Schematic diagram

IV-A. Acquisition of data

Arduino UNO receives signals from temperature sensors and potentiometers through the following script:

```
#include <DHT.h>
#include <Thread.h>

#define DHTPIN 2
#define DHTPIN1 4

#define DHTTYPE DHT11

DHT dht1(DHTPIN, DHTTYPE);
DHT dht2(DHTPIN1, DHTTYPE);

Thread thrBlink1 = Thread();
Thread thrBlink2 = Thread();
Thread thrBlink3 = Thread();

float tempC;
int pinLM35 =A0;
int pinPot = A3;
int pinPot1 = A2;
int pinPot2 = A1;
int parpadeo;
int valorPot;
int parpadeo1;
int valorPot1;
int parpadeo2;
int valorPot2;


void setup() {

  Serial.begin(9600);


   dht1.begin();
   dht2.begin();

  thrBlink1.enabled = true;
  thrBlink1.setInterval(1000);
  thrBlink1.onRun(sensor1);
```

```
  thrBlink2.enabled = true;
  thrBlink2.setInterval(2000);
  thrBlink2.onRun(sensor2);

  thrBlink3.enabled = true;
  thrBlink3.setInterval(3000);
  thrBlink3.onRun(sensor3);

}
void blink() {

}
void loop() {

  thrBlink1.run();

  thrBlink3.run();
}
void sensor1(){

  valorPot = analogRead(pinPot);
  parpadeo = map(valorPot, 0, 1023, 100, 500);

  float t = dht2.readTemperature();
  if (isnan(t)) {
    Serial.println("Error obteniendo los datos del
    sensor DHT11");
    return;
  }

  Serial.print(t);
  Serial.print(",");
  Serial.print(parpadeo);
  delay(1);

}
void sensor2(){

  valorPot1 = analogRead(pinPot1);
  parpadeo1 = map(valorPot1, 0, 1023, 100, 500);
  float t = dht2.readTemperature();
  if (isnan(t)) {
    Serial.println("Error obteniendo los datos del
    sensor DHT11");
    return;
  }
  Serial.print(",");
  Serial.print(t);
  Serial.print(",");
  Serial.print(parpadeo1);
  delay(1);

}
void sensor3(){
  valorPot2 = analogRead(pinPot2);
  parpadeo2 = map(valorPot2, 0, 1023, 100, 500);
  tempC = analogRead(pinLM35);
  tempC = (5.0 * tempC * 100.0)/1024.0;
  Serial.print(",");
  Serial.print(tempC);
  Serial.print(",");
  Serial.println(parpadeo2);
  delay(100);
}
```

IV-B. Processing

With the information of the sensor data, the processing and the simulation of the policer are carried out, establishing a PIR value for each signal. A graph will show the values obtained and the effect of the policer. The python script is as follows:

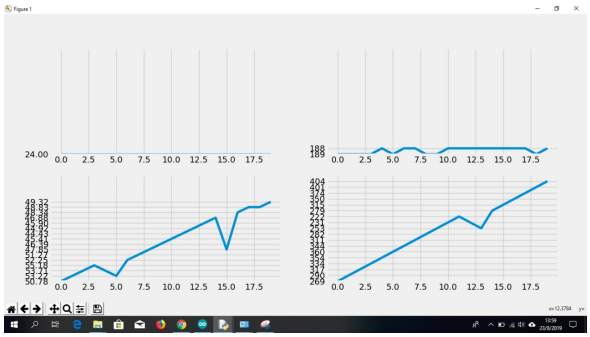
```
from matplotlib import pyplot as plt #Importa pyplot
     para realizar la grafica.
from matplotlib import animation  #Importa animation
     que permite actualizar la grafica en intervalos
     concretos
from matplotlib import style #Permite cambiar el
    estilo de nuestra grafica.
import serial #Importa libreria para trabajar con el
     puerto serie.


style.use('fivethirtyeight')  #Cambia el estilo de
    nuestra grafica.

fig = plt.figure() #Creamos un objeto para almacenar
     el la grafica.



ax1 = fig.add_subplot(2,2,1) #Anadimos una "
    subgrafica" a nuestra ventana.

ax2 = fig.add_subplot(2,2,2) #Anadimos una "
    subgrafica" a nuestra ventana.

ax3 = fig.add_subplot(2,2,3) #Anadimos una "
    subgrafica" a nuestra ventana.

ax4 = fig.add_subplot(2,2,4) #Anadimos una "
    subgrafica" a nuestra ventana.

ser = serial.Serial('COM3', 9600) #Abrimos puerto
    Serie, sustituir 'dev/ttyUSB0', por 'COM2', '
    COM3' o el puerto que use el Arduino en tu PC.
ser.timeout = 1
ser.setDTR(False)
ser.flushInput()
ser.setDTR(True)
ser.readline()
rojoPot1 = 230
rojoPot2 = 250
rojoTemp1 = 19.00
rojoTemp2 = 24.00

metering = input("Aplicar Calidad de Servicio?(s/n):
     ")
def plotea (i):
     temperatura = []
     pot = []
     temperatura1 = []
     pot1 = []
     muestras= 20
     for i in range(0,muestras): #Bucle for para
    recibir 100 valores anets de pintarlos.

          datoString = ser.readline()  #Leemos una
    linea enviada (hasta que se recibe el caracter \
    n).

          datos = str(datoString).split(",")
          if(metering == 's'):
               if(len(datos[0][2:])!=0):
                    if(float(datos[0][2:])>rojoTemp1
    ):
                         temperatura.append (str(
    rojoTemp1))

                    else:
                         temperatura.append (datos
    [0][2:])
               if(len(datos[1]) !=0 ):
                    if(int(datos[1])>rojoPot1):
                         pot.append (str(rojoPot1))
```

```
                    else:
                            pot.append (datos[1])
                if(len(datos[2]) !=0):
                    if(float(datos[2])>rojoTemp2):
                        temperatura1.append (str(
    rojoTemp2))

                    else:
                        temperatura1.append (datos
    [2])
                if( len(datos[3][:-5]) !=0):
                    if(int(datos[3][:-5])>rojoPot2):
                        pot1.append (str(rojoPot2))

                    else:
                        pot1.append (datos[3][:-5])
            else:
                temperatura.append (datos[0][2:])
                pot.append (datos[1])
                temperatura1.append (datos[2])
                pot1.append (datos[3][:-5])

        ax1.clear() #Limpiamos la grafica para volver a
        pintar.
        ax1.set_ylim([0,20]) #Ajustamos el limite
    vertical de la grafica.
        ax2.clear() #Limpiamos la grafica para volver a
        pintar.
        ax2.set_ylim([0,18]) #Ajustamos el limite
    vertical de la grafica.
        ax3.clear() #Limpiamos la grafica para volver a
        pintar.
        ax3.set_ylim([0,20]) #Ajustamos el limite
    vertical de la grafica.
        ax4.clear() #Limpiamos la grafica para volver a
        pintar.
        ax4.set_ylim([0,18]) #Ajustamos el limite
    vertical de la grafica.

        try:  #Nos permite comprobar si hay un error al
        ejecutar la siguiente instruccion.
            ax1.plot(range(0,muestras), temperatura) #
    Plotea los datos en x de 0 a 100.
            ax2.plot(range(0,muestras), pot) # Plotea
    los datos en x de 0 a 100.
            ax3.plot(range(0,muestras), temperatura1) #
     Plotea los datos en x de 0 a 100.
            ax4.plot(range(0,muestras), pot1) # Plotea
    los datos en x de 0 a 100.
        except UnicodeDecodeError: #Si se produce el
    error al plotear no hacemos nada y evitamos que
    el programa se pare.
            pass

ani = animation.FuncAnimation(fig, plotea, interval
    = 1) #Creamos animacion para que se ejecute la
    funcion plotea con un intervalo de 1ms.

plt.show() #Muestra la grafica.
```

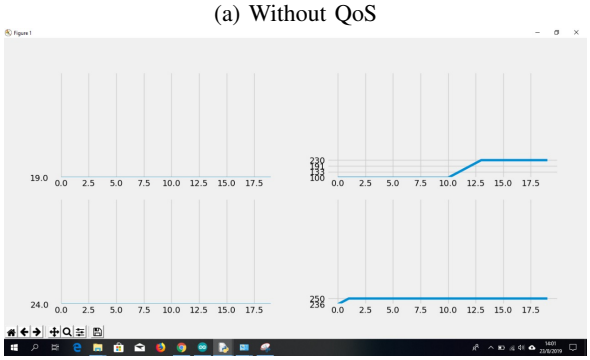
V. Results

The results were as expected, the graph of the normal values of the sensors with respect to time is observed first. In this case, there are no restrictions and the values are read as they are detected by the sensors.

In the second graph, the flag is raised to apply the QoS changes. It is observed that the graph is cut at the PIR points of each sensor, it does not allow values greater than those predefined in the script. In the figure 4 you can see the aforementioned.



(a) Without QoS



(b) With QoS

Figura 4: *Output graphs*

VI. Conclusions

- The project executes the policer correctly, and its operation is observed in real time.
- The developed application can be extended with more functions to be used in any telemetry field.
- It is necessary for better application performance to use another device because Arduino presents problems when measuring so much data at the same time, in this way we give scalability to the proposed solution.

Referencias

[1] Juniper Networks, “Learn About Quality of Services (QoS)” by Collen Feerick , 2015.