

Reporte de práctica 3: protocolos de comunicación

(Marzo 15, 2019)

Thomas, Manuel & Ruz, Pablo. *Principios de Mecatrónica, ITAM*

Abstract— A lo largo de tres sesiones, se buscó entender y programar con protocolos de comunicación UART, SPI e I2C. Así mismo, se programó el XBee para poder establecer comunicación entre dos de estos aparatos, uno de ellos por medio del Arduino.

Index Terms—XBee, UART, SPI, I2C, Protocolos de comunicación, Arduino, acelerómetro.

I. INTRODUCCIÓN

La transmisión de datos de un aparato electrónico a otro es algo por demás común. No obstante, llevarlo a cabo no es tan sencillo como podría parecer. Para lograrlo, es necesario contar con protocolos de comunicación e interfaces. Por medio de la interfaz, es posible definir a uno de los aparatos como el *emisor* (el que transmite el mensaje) y a otro como el *receptor* (el que recibe el mensaje).

La interfaz mediante la cual operan los protocolos está basada en estándares que permiten establecer el canal de comunicación entre los aparatos en cuestión. La obediencia a estos estándares es lo que permite establecer la comunicación entre aparatos electrónicos, desde pequeños XBee hasta impresoras y hardware más complejo.

Existen varios tipos de comunicación, entre los cuales destacan la comunicación en serie y la comunicación en paralelo. La primera de ellas se caracteriza por la transmisión de datos a través de un único canal, mientras que la que es en paralelo se lleva a cabo mediante múltiples canales de manera simultánea.

II. MARCO TEÓRICO

La comunicación serial y la comunicación en paralelo varían notablemente en velocidad. En efecto, en tanto que aquella que es en paralelo es capaz de transmitir varios paquetes de datos al mismo tiempo, la serial únicamente puede enviar uno por uno, como se muestra en las siguientes imágenes.

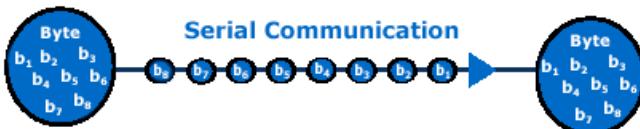


Fig. 1

Como puede observarse, los datos son enviados de un punto A a uno B uno tras otro.

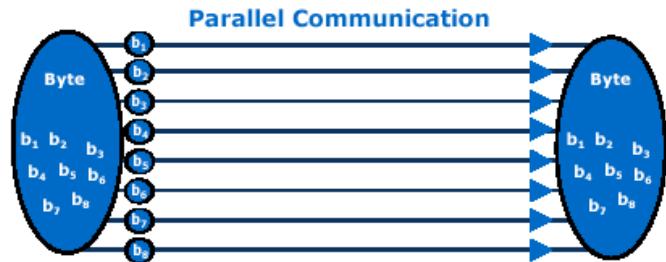


Fig. 2

En la comunicación paralela, los datos pueden ser enviados del punto A al punto B por medio de los diversos canales, permitiendo así que se haga de manera simultánea.

Aunque en un inicio parecería no haber ventaja para la comunicación serial, lo cierto es que la comunicación paralela puede generar problemas de interferencia entre los diversos canales. Esto es, cuando una señal está muy cerca de la otra, pueden afectarse entre sí, generando distorsión en los datos.

A su vez, la comunicación en serie presenta dos tipos de “subcomunicación”, la síncrona y la asíncrona. En el caso de la primera, tanto el emisor como el receptor funcionan por medio de la misma señal de reloj, por lo que se utiliza siempre una configuración *maestro-esclavo*. Por su parte, la segunda no requiere de una señal de reloj entre ambas partes, sino que el emisor, previo a enviar los datos, manda una señal de sincronización que permite que el receptor “acomode” los datos de manera adecuada cuando estos se reciben para lograr la comunicación efectiva.

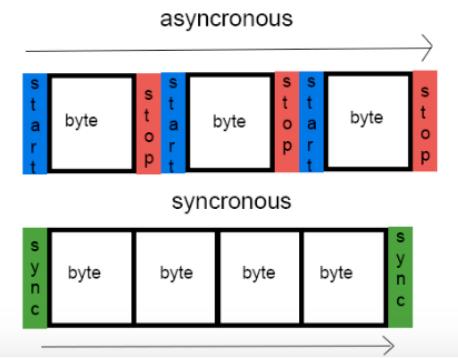


Fig. 3

Obsérvese que la comunicación síncrona manda todos los datos en una única transmisión, lo que la hace más veloz. Por su parte, la asíncrona manda uno por uno, haciendo un alto cada vez que un nuevo dato será enviado.

La comunicación UART (Universal Asynchronous Receiver Transmitter) es un tipo de protocolo serial asíncrono que se adoptó como estándar en los años 60. Es un estándar de baja velocidad, que transmite alrededor de 1Mbit/s.

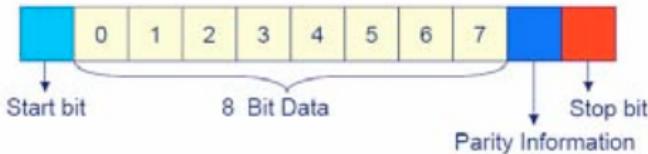


Fig. 4

Obsérvese que los bits no contienen información alguna acerca de la frecuencia de reloj, por tratarse de una comunicación asíncrona.

El "start bit" y el "stop bit" indican el inicio y el fin de la transmisión, respectivamente.

El protocolo SPI (Serial Peripheral Interface) es síncrono y tiene cuatro componentes principales, a saber: (i) SCLK (serial clock), (ii) MOSI (master out slave in), (iii) MISO (master in slave out), (iv) SS (slave select). El primero de ellos es el reloj que define la frecuencia a la que ambos, emisor y receptor, han de operar. El segundo define los datos que se envían del maestro al esclavo, el tercero los que van del esclavo al maestro, y el cuarto es el que permite hacer la selección del esclavo. Este protocolo es generalmente usado para conectar los periféricos entre sí con los microporcesadores.

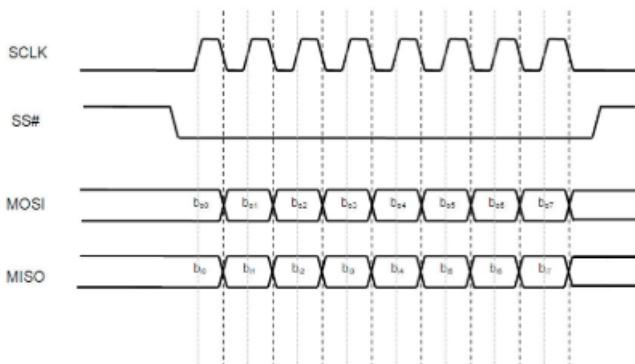


Fig. 5

En esta figura se aprecia un protocolo SPI simple. Obsérvese que tanto MOSI como MISO operan en la caída del reloj (ambos con la misma frecuencia). El SS es el responsable de indicar si es en la caída o en la subida del ciclo del reloj cuando los datos son apalancados (toggle).

Finalmente, el protocolo I2C (Inter-Integrated Circuits) fue originalmente diseñado por Phillips para comunicación entre sistemas periféricos y sistemas embebidos. Al igual que SPI, es completamente síncrono. Opera por medio de dos cables, denominados SCL y SDA. El primero de ellos es por donde se transmite la señal del reloj, mientras que el segundo es por el que se transmiten los datos. Cuando se conectan varios dispositivos a las mismas líneas, es importante saber la señal que se está enviando, a cuál de ellos debería de afectar, por lo que se cuenta con un sistema de direcciones. Así mismo, existe una referencia común: todos están conectados a la misma tierra.

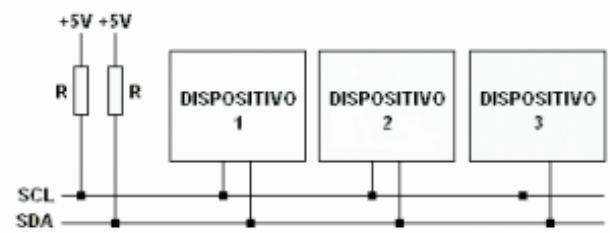


Fig. 6

Esquema de dispositivos con configuración I2C.

Véase que todos ellos están conectados a los dos canales SCL y SDA, permitiendo que exista una sincronización con el reloj de SCL y que todos puedan recibir los datos del SDA.

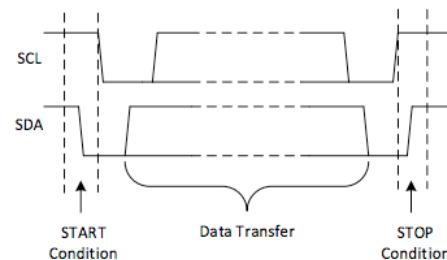


Fig. 7

La transferencia de datos se lleva a cabo entre las condiciones de inicio y de fin.

Como se mencionó anteriormente, todos los protocolos de comunicación serial síncrona funcionan bajo el esquema de maestro-esclavo. Este esquema se basa en el principio de que un único componente (maestro) es capaz de recolectar la información procesada por cada uno de los elementos secundarios (esclavos) conectados a él y redistribuirlo a cada uno de ellos. Es importante notar que este tipo de configuración permite el trabajo en paralelo (aunque aquí los protocolos son síncronos, por lo que esto no aplica), dado que cada esclavo es independiente de otro. Cuando no existe trabajo en paralelo, se conoce como comunicación punto a punto, mientras que cuando sí, se le conoce como comunicación colectiva.

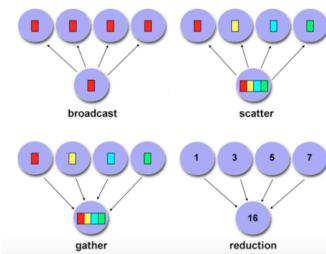


Fig. 8

Muestra de comunicación colectiva. La comunicación colectiva por medio de la estructura maestro-esclavo puede permitir al maestro enviar a cada esclavo la misma información (broadcast), distribuir una parte a cada uno (scatter), almacenar la de cada uno (gather) o bien operar con ella para convertirla en sólo un pedazo de información (reduction).

En muchas ocasiones, los datos que se quieren transmitir de un aparato a otro dependen de las condiciones en las que se encuentre el emisor, y el procesamiento de los mismos, de las del receptor. Una de estas condiciones es la aceleración. Esto es, es posible que de acuerdo con la aceleración que esté experimentando un objeto en un momento

dado, los datos deban de ser procesados de manera distinta, o incluso que la fórmula de la operación dependa directamente de la aceleración. Así, muchos elementos electrónicos, incluyendo la FPGA, tienen acelerómetros dentro de ellos mismos para poder realizar diversas funciones.

Como su nombre lo indica, un acelerómetro es un instrumento que mide la aceleración del cuerpo físico al que está unido, mediante la cuantificación de la masa inercial interna, de acuerdo con la fórmula siguiente:

$$A_m(s) = s^2 \cdot X(s) + \frac{\omega_n}{Q} s \cdot X(s) + \omega_n^2 X(s)$$

Fig. 9

Fórmula del acelerómetro una vez que se ha aplicado una transformada de Laplace. En efecto, obsérvese que la función es dependiente de $s = \sigma + j\omega$, y no del tiempo t , con ω = frecuencia de resonancia del sistema [Hz].

III. DESARROLLO

PARTE 1

En la primer parte de la práctica comenzamos implementando el protocolo de comunicación I2C para la comunicación serial síncrona maestro-esclavo, la cual utilizamos para comunicar dos Arduino en el cual uno enviasse información que pudiéramos propiamente ingresar directo a la terminal y el otro la recibiera. Lo que se construyó fue un programa que operara de la siguiente forma: en el Arduino que se definió como maestro definimos que se conectara mediante el ya mencionado protocolo con otro arduino, siendo este primero el que enviaría datos. Se enviaría una combinación de 2 correspondiente a 00, 01, 10, 11.

Por otro lado, el Arduino definido como esclavo se encargaría de leer los datos que eran enviados por su correspondiente maestro. En caso de recibir las combinaciones 00 o 11, el Arduino esclavo debía de hacer que un motor conectado a él, se mantuviera en reposo; para la combinación 01, dicho motor debía girar hacia la derecha y en caso contrario, para la combinación 10, girar hacia la izquierda.

```
#include<Wire.h>
void setup() {
  pinMode(13,OUTPUT);
  pinMode(11,OUTPUT);
  Wire.begin();
  Wire.onReceive(evento);
}
void evento() {
  int z=Wire.read();
  if(z==0){
    digitalWrite(11,HIGH);
    digitalWrite(13,HIGH);
  }
  else if(z==1){
    digitalWrite(11,LOW);
    digitalWrite(13,HIGH);
  }
  else if(z==2){
    digitalWrite(11,HIGH);
    digitalWrite(13,LOW);
  }
  else if(z==3){
    digitalWrite(11,LOW);
    digitalWrite(13,LOW);
    Serial.println(z);
  }
}
void loop(){}
```

Fig. 10

Código de recepción (esclavo) correspondiente a la primera parte de la conexión maestro-esclavo entre dos arduinos. Véase que de acuerdo con el valor recibido se activa el movimiento del motor hacia un lado o hacia otro, o bien, se detiene.

El circuito del esclavo era sencillo pues literalmente se conectó de un lado del Arduino el motor y por otro lado una fuente generadora de voltaje utilizando un puente H que posibilitaría el cambio de dirección de giro del motor.

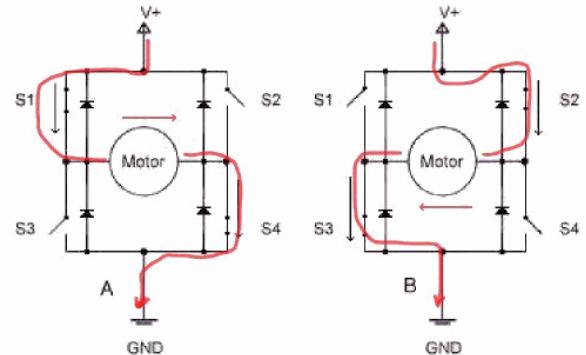


Fig. 11

Esquema de funcionamiento del puente H para lograr el cambio de dirección en el giro del motor. Para el circuito de la primera parte.

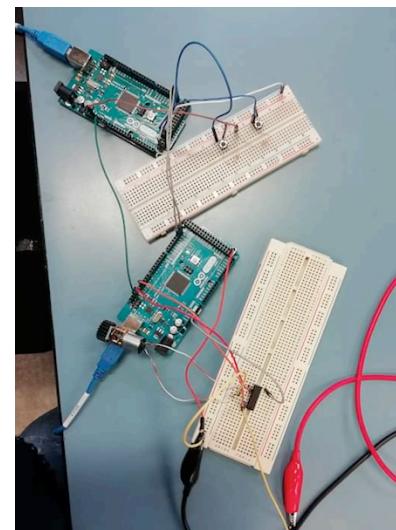


Fig. 12

Aquí se muestra la conexión entre la FPGA que contiene el código del maestro y la que contiene el código del esclavo. Véase que el motor está conectado a la parte del esclavo, que las tierras son comunes y que el voltaje proviene de ambas partes, aunque el que impulsa al motor es el del VCC.

PARTE 2

Esta fue la parte que no logramos completar. El problema no consistió en si en el concepto de la comunicación SPI en la que utilizaríamos el acelerómetro de la FPGA, sino en propiamente utilizar la tarjeta y el lenguaje VHDL pues nuestras bases del semestre pasado de circuitos lógicos no son muy sólidas y no logramos hallar un manual que explicara de forma tan clara los conocimientos pertinentes para realizar esta parte.

Lo que se pretendía hacer era utilizar el acelerómetro como fuente de datos de tal modo que al detectar un cambio en x y o z, según lo definiríamos, lograremos escribir esa información en los registros de la FPGA para posteriormente darle un significado(codificarlo) y enviar esa información un circuito construido por LEDs, los cuales se encenderían según se programara para cada dirección de cambio que indicara el acelerómetro.

PARTE 3

En esta sección implementamos el protocolo Zigbee para la comunicación inalámbrica entre dos dispositivos Xbee, uno conectado al software XCTU y otro en Arduino (montados sobre un shield para pc y para Arduino, respectivamente).

El objetivo era lograr esta comunicación inalámbrica punto a punto. Para ello definimos un canal de conexión (medio de transmisión de datos) y enlazamos un Xbee con otro definiendo específicamente a qué dispositivo se enviarían los datos mediante la configuración mostrada en la figuras 12 y 13.

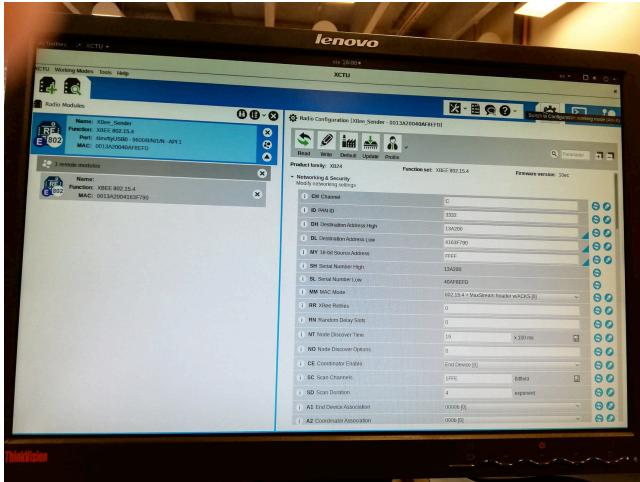


Fig. 13

Configuración en XCTU para la comunicación entre dispositivos Xbee. La parte más relevante de esta configuración fueron los campos DH (dirección alta del destino), DL (dirección baja del destino), y el canal.

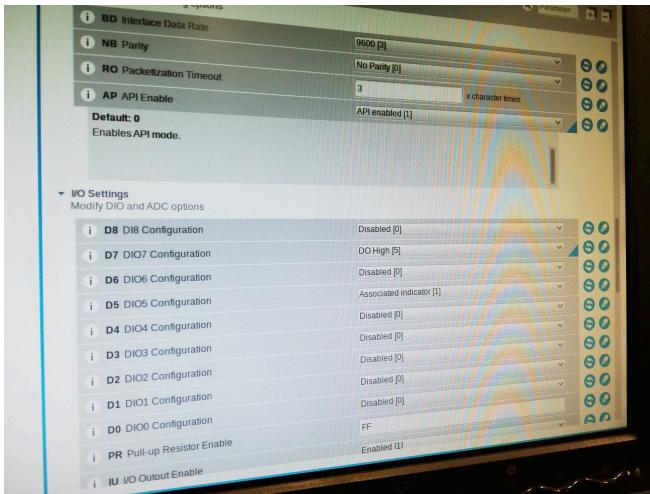


Fig. 14

Configuración en XCTU para la comunicación entre dispositivos Xbee. Véase que la configuración del campo AP API Enable, que se encuentra en 1. XCTU cuenta con dos valores posibles, 0 y 1, donde el primero de ellos representa el protocolo de comunicación transparente y el segundo el que permite que sea de punto a punto.

Una vez lograda esta configuración, recurrimos a la herramienta “API frames generator” la cual servía para realizar una codificación en binario o decimal de ciertos datos con los que podríamos enviar al dispositivo receptor. Lo hicimos de la siguiente manera:

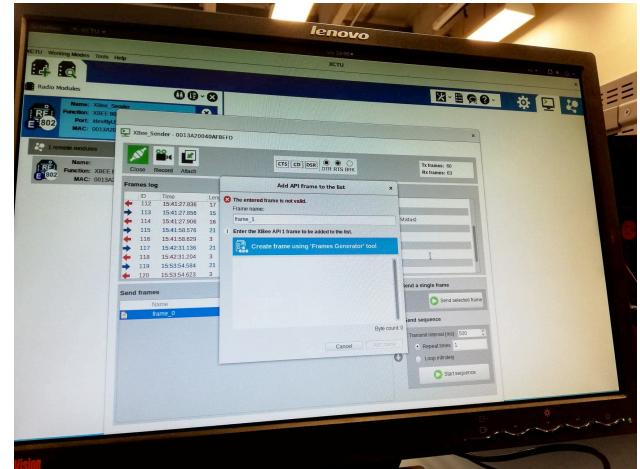


Fig. 15

Generador de marcos. Esta característica está activa únicamente cuando el API Enable se encuentra en el valor de 1, como se mostró en la figura pasada.

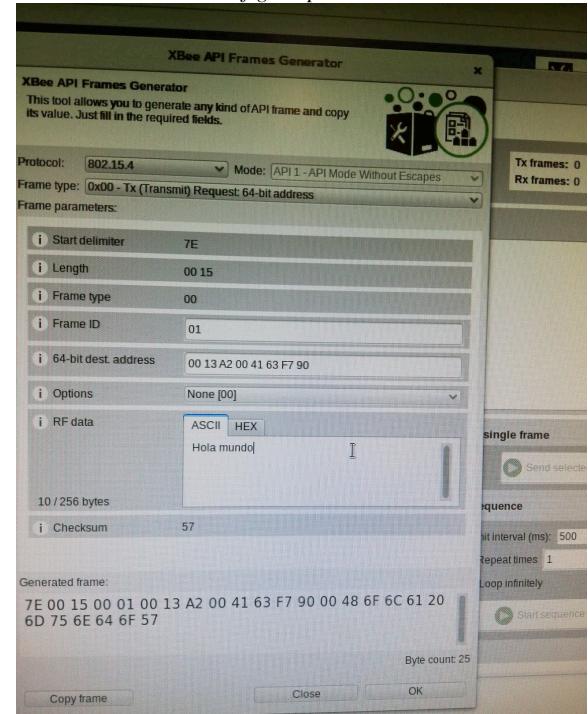


Fig. 16

Obsérvese la dirección “64.bit dest. address”, que está conformada por la concatenación de DH y DL de la parte anterior. En la caja RF-data simplemente se ingresó un mensaje cualquiera para verificar la llegada del mismo al XBee conectado con Arduino.

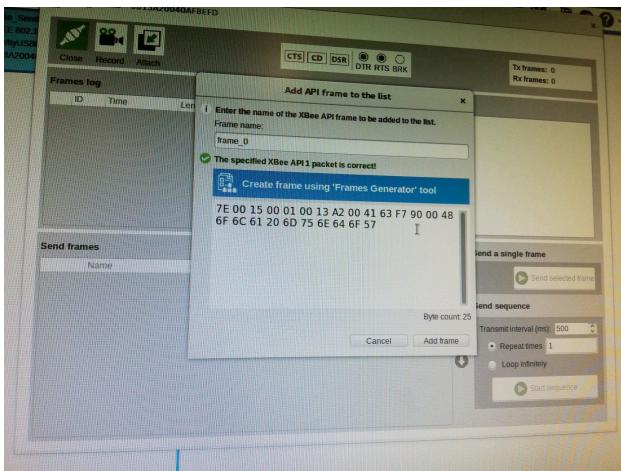


Fig. 17

Finalización de la configuración del XCTU y los canales de transmisión del XBee.

Para saber si la conexión fue o no exitosa, se mandó el mensaje con ayuda del botón “Send selected”. En un inicio, la respuesta era negativa. Esto es, mostraba un mensaje en el que se indicaba que el receptor no estaba activo o bien que no había recibido el mensaje. Para solucionarlo, fue necesario montar en el Arduino el código que se muestra en la siguiente figura.

```
XBee_Arduino
#include <SoftwareSerial.h>

SoftwareSerial XBee(10, 11); // RX, TX

void setup()
{
    // Set up both ports at 9600 baud. This value is most important
    // for the XBee. Make sure the baud rate matches the config
    // setting of your XBee.
    XBee.begin(9600);
    Serial.begin(9600);
}

void loop()
{
    if (Serial.available())
    {
        XBee.write(Serial.read());
        delay(1000);
    }
    if (XBee.available())
    {
        Serial.write(XBee.read());
        Serial.println(XBee.read());
        delay(1000);
    }
}
```

Fig. 18

El código de recepción del Arduino para el XBee. Nótese que en la parte del `setup()`, se inicializa el receptor mediante `XBee.begin(9600)`, y que en la parte del `loop()` es donde se pregunta si se está o no recibiendo la señal.

Una vez que esto fue implementado y cargado en el Arduino, se logró recibir el “success” en la pantalla del XCTU al mandar el mensaje y que el LED prendiera.

IV. RESULTADOS

En esta práctica no hay resultados matemáticos. No obstante, es importante reconocer que hubo partes que no lograron resolverse al 100%, como la programación del acelerómetro con la FPGA. Lo demás, no obstante, se logró hacer adecuadamente y con los resultados esperados.

V. CONCLUSIONES

Pablo A. Ruz Salmones: los protocolos de comunicación son de gran utilidad para poder llevar a cabo funciones en las que un aparato dé una orden y otro la reciba y actúe de acuerdo con lo que se le indicó. Todos los protocolos usados resultaron ser efectivos, y habría que hacer pruebas más a fondo para poder conocer las diferencias en velocidad de transmisión, por ejemplo, dado que aquí únicamente se probó la efectividad más no la eficiencia.

Victor M. Thomas Ruiz: La presente práctica fue sin duda la más compleja hasta ahora, pues no teníamos noción y jamás tuvimos una introducción previa al tema; es decir, nunca antes nos habían introducido a los protocolos de comunicación, por lo que, desde un inicio, tuvimos que investigar por cuenta propia lo que significaban los protocolos I2C, UART y SPI para comunicar dispositivos, como funcionaba un xbee y que era la comunicación punto a punto y a usar el acelerómetro de la fpga.

El resultado final no fue exactamente lo que esperábamos porque, a pesar de haber concluido con éxito la primera y tercera parte, no logramos concluir la parte dos de la fpga; además, la tercera sección referente al xbee nos costó mucho trabajo.

Logramos comprender el protocolo de comunicación I2C haciendo referencia a la comunicación maestro esclavo así como el UART y el Zigbee mediante la comunicación inalámbrica y logramos implementar ese conocimiento en la primera y tercer práctica. Es bastante útil poder comunicar entre diferentes dispositivos pues es el principio básico para poder integrar sistemas que funcionen con mayor grado de complejidad y que realicen más tareas.

VI. ROLES

Pablo A. Ruz Salmones: tanto la práctica como el reporte fueron realizados en equipo. Igual que en la práctica pasada, la diferencia estriba en una mayor contribución de Manuel en la segunda de la práctica y una mía en el reporte. Fue una práctica complicada, pero pudimos organizarnos para terminar satisfactoriamente la mayor parte de ella.

Manuel Thomas: En esta ocasión, tanto Pablo como yo, trabajamos de igual modo en ambas partes de la práctica, tanto en la sesión de laboratorio como en la elaboración del reporte escrito. Logramos una mejor distribución del trabajo en donde ambos nos desenvolvimos de igual modo.

VII. REFERENCIAS

[I] “Comunicación Serie”. Universidad de Buenos Aires; Facultad de Ingeniería. Disponible en: <http://web.fi.uba.ar/~fferrari/tps/%5B66.02%20-%20Laboratorio%5D%20Monograf%C3%ADA%20-%20Comunicaci%C3%B3n%20Serie.pdf>

[II] Valdez, J & Becker, J . “Understanding the I2C Bus”. Texas Instruments. Disponible en: <http://www.ti.com/lit/an/slva704/slva704.pdf>

[III] “Esquemas de Comunicación”. Benemérita Universidad Autónoma de Puebla. Disponible en: <https://www.cs.buap.mx/~mtovar/doc/PCPA/ComunicacionPP.pdf>

[IV] “Bus de Comunicación I2C”. Universidad de Sevilla; Biblioteca de la Escuela Superior de Ingenieros de Sevilla. Disponible en: <http://bibing.us.es/proyectos/abreproy/70099/fichero/Trabajo+Fin+de+Master+-+Vicente+Madero%252F14+-+Ap%C3%A9ndice+H+-+I2C.pdf>

[V] "Synchronous vs Asynchronous". Indiana University; Purdue School of Engineering and Technology. Disponible en:
http://www.engr.iupui.edu/~skoskie/ECE362/lecture_notes/LNB25_html/ext12.html

[VI] Gómez, P. "Interfaces" Simposio Argentino de Sistemas Embebidos. Disponible en:
http://www.sase.com.ar/2011/files/2010/11/SASE2011-Introduccion_Interfaces.pdf

[VII] Bouziane, H. "Proyecto final de carrera". Universidad Politécnica de Catalunya. Disponible en:
<https://upcommons.upc.edu/bitstream/handle/2099.1/7998/Mem%C3%B2ria.pdf>