Protocolos de Comunicación

Leslie P. Brenes, Diego Villafuerte

Resumen—En esta práctica se implementaron distintos protocolos de comunicación: I^2C, UART, y SPI. Con ellos, se generó comunicación maestro-esclavo entre arduinos utilizando un acelerómetro, y comunicación entre XBees inalámbricamente.

1. INTRODUCCIÓN

Para el proyecto final, necesitamos aprender a utilizar sensores y comunicación entre 2 carritos, para ello es necesario conocer y probar distintos protocolos de comunicación, en esta práctica exploramos I^2C, UART, y SPI. Estos protocolos funcionan de distinta forma y es necesario familiarizarnos con los protocolos, ver sus ventajas y desventajas. En particular, nos interesa aprender a generar comunicación inalámbrica entre dispositivos y comunicación serial maestro-esclavo entre dos arduinos. También nos interesa aprender a utilizar un acelerómetro.

2. MARCO TEÓRICO

Para el intercambio de información, se debe implementar un protocolo de comunicación común y estos pueden separarse en una de dos categorías: paralelo o en serie.

Las interfaces paralelas transfieren múltiples bits al mismo tiempo. Por lo general, requieren buses de datos. Los datos se transfieren en enormes olas de 1 y 0. En cambio, las interfaces seriales transmiten sus datos, un bit a la vez. La principal desventaja de las interfaces paralelas es que requieren múltiples líneas de entrada / salida y por lo general contamos con recursos limitados.

Entre las interfaces seriales más comunes se encuentran SPI, I^2C. Cada una de estas interfaces seriales puede clasificarse en uno de dos grupos: síncrono o asíncrono.

Una interfaz serie síncrona siempre empareja sus líneas de datos con una señal de reloj, por lo que todos los dispositivos comparten un reloj común. Esto hace que que la transferencia en serie sea más directa, y a menudo más rápida, pero también requiere al menos un cable entre los dispositivos SPI e I^2C son dos ejemplos de interfaces síncronas.

Asíncrono significa que los datos se transfieren sin necesidad de una señal de reloj externa. Funciona para minimizar los cables necesarios y el uso de pines de I/O, pero necesitamos más cuidado para transferir y recibir datos de manera confiable.[1]

Philips diseñó el bus I^2C a principios de los años 80 para permitir una fácil comunicación entre componentes en la misma placa de circuitos. El nombre I^2C se traduce en Ïnter IC". I^2C no solo se utiliza en placas individuales, sino también para conectar componentes que están conectados mediante un cable. [2]

Las características más significativas incluyen:

- Sólo se requieren dos líneas de bus.
- Existen relaciones simples maestro / esclavo entre todos los componentes.
- Cada dispositivo conectado al bus es direccionable por software por una dirección única.
- I^2C es un verdadero bus multi-maestro que proporciona arbitraje y detección de colisiones.

Los módulos XBee son soluciones integradas que brindan un medio inalámbrico para la interconexión y comunicación entre dispositivos. Estos módulos utilizan el protocolo de red IEEE 802.15.4 para crear redes FAST POINT-TO-MULTIPOINT; o para redes PEER-TO-PEER. Fueron diseñados para aplicaciones que requieren de un alto tráfico de datos, baja latencia y una sincronización de comunicación predecible. Por lo que básicamente XBee es propiedad de Digi basado en el protocolo Zigbee. En términos simples, los XBee son módulos inalámbricos fáciles de usar.[**?**]

3. DESARROLLO

3.1. I^2C

Para esta sección, fue necesario trabajar con otro equipo para poder generar la comunicación maestro-esclavo entre dos arduinos. Nosotros generamos el rol de esclavo, por lo que fue necesario generar un programa que recibiera eventos y a partir del evento, ejecutara funciones específicas. Nos basamos principalmente en un tutorial de Arduino.[3]

Primero, antes de implementar el protocolo de comunicación, generamos un programa para probar las distintas funciones a implementar: Freno pasivo, rotar en sentido de las manecillas, rotar en contrasentido de las manecillas y freno activo, además configuramos el puente H con dos fuentes de voltaje.

```
int speedPin = 3;
int motor1APin = 36;
int motor2APin = 37;
int ledPin = 13;
int speed_value_motor1;
char sentido = '1';

void setup(){
  pinMode(speedPin, OUTPUT);
```

```
  pinMode(motor1APin, OUTPUT);
  pinMode(motor2APin, OUTPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  Wire.begin(25);
}
void loop(){
  if (Wire.available() > 0) {
    sentido = Wire.read();
  }
  if(sentido == '1'){
    digitalWrite(motor1APin, HIGH);
    digitalWrite(motor2APin, LOW);
  }
  else if(sentido == '2'){
    digitalWrite(motor1APin, LOW);
    digitalWrite(motor2APin, HIGH);
  }
  else if(sentido == '3'){
    digitalWrite(motor1APin, LOW);
    digitalWrite(motor2APin, LOW);
  }
  else{
    digitalWrite(motor1APin, HIGH);
    digitalWrite(motor2APin, HIGH);
  }

speed_value_motor1 = 127;
analogWrite(speedPin, speed_value_motor1);

}
```

Posteriormente solo se modificó el programa para la parte de comunicación.

```
#include <Wire.h>
int speedPin = 3;
int motor1APin = 36;
int motor2APin = 37;
int ledPin = 13;
int speed_value_motor1;
int sentido = 1;

void setup(){
  pinMode(speedPin, OUTPUT);
  pinMode(motor1APin, OUTPUT);
  pinMode(motor2APin, OUTPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  Wire.begin(25);
  Wire.onReceive(receiveEvent);

}
void loop(){
  delay(100);
}

  void receiveEvent(int sentido)
{
  sentido = Wire.read();
// receive byte as an integer
  Serial.println(sentido);
  if(sentido == 1){
    digitalWrite(motor1APin, HIGH);
    digitalWrite(motor2APin, LOW);
  }
  else if(sentido == 2){
    digitalWrite(motor1APin, LOW);
    digitalWrite(motor2APin, HIGH);
  }
  else if(sentido == 3){
    digitalWrite(motor1APin, LOW);
    digitalWrite(motor2APin, LOW);
  }
  else{
    digitalWrite(motor1APin, HIGH);
    digitalWrite(motor2APin, HIGH);
  }
  speed_value_motor1 = 127;
  analogWrite(speedPin, speed_value_motor1);
}
```

3.2. SPI y acelerómetro

En esta sección, primero configuramos los puertos de acuerdo a la hoja de especificaciones del acelerómetro en el archivo *accelrw.vhd*.

```
port map
(
clock    => CLK,     --system clock
reset_n  => reset_n, --asynchronous reset
enable   => enable ,-initiate transaction
cpol     => '1', --spi clock polarity
cpha     => '1',--spi clock phase
clk_div  => 5,--system clock cycles per 1/2 period of
addr     => 0,--address of slave
rw       => rw_buffer,--'0' for read, '1' for write
tx_cmd   => cmd_to_spi,--command to transmit
tx_data  => TX_IN,--data to transmit
sclk     => I2C_SCLK,--SPI SCLK
ss_n(0)  => G_SENSOR_CS_N,--Sensor SSN
sdio     => I2C_SDAT,--SPI SDIO
busy     => busy,--busy / data ready signal
rx_data  => RX_out--data received
);
```

3.3. UART y XBee

Para este protocolo, primero fue importante que el número de serie High y Low coincidiera en ambos XBee (esto se configuró en XCTU). Luego, agregamos la librería SoftwareSerial en Arduino.

```
#include <SoftwareSerial.h>

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

void setup() {
  pinMode(7, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(4, INPUT);
```

```
  // Open serial communications and wait
  //for port to open:
  xbeeSerial.begin(9600);
  // set the data rate for
  //the SoftwareSerial port
}

void loop() { // run over and over
  delay(500);
  char var = xbeeSerial.read();
  if(digitalRead(4)){
    xbeeSerial.println("Nada impresiona a Gary");
  }
  if(var=='1'){
    digitalWrite(7, HIGH);
    digitalWrite(6, LOW);
    digitalWrite(5, LOW);
  }
  else if(var=='2'){
    digitalWrite(6, HIGH);
    digitalWrite(7, LOW);
    digitalWrite(5, LOW);
  }
  else if(var=='3'){
    digitalWrite(5, HIGH);
    digitalWrite(6, LOW);
    digitalWrite(7, LOW);
  }
  else if(var=='4'){
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
    digitalWrite(7, HIGH);
  }
  else{
    digitalWrite(7, LOW);
    digitalWrite(6, LOW);
    digitalWrite(5, LOW);
  }

}
```

4. RESULTADOS

La primera sección con el protoclo I^2C pudimos configurar perfectamente el puerto H, y antes de generar la comunicación entre los microcontroladores, pudimos probar las distintas funcionalidades a implementar. Gracias al tutorial de comunicación Maestro-Esclavo de Arduino, generar el programa de esclavo fue muy sencillo. Resaltamos que en esta sección no quedaba claro por qué era necesario un delay() en la sección loop, pero sin el delay, el programa no funcionaba.

Para la sección de SPI, no pudimos pasar del punto 2 señalado en la práctica, que era "Leer el registro que contiene elIDdel acelerómetro y verificar que el valor es correcto". Dado que trabajamos sobre un repositorio, y que los parámetros a configurar los obtuvimos de acuerdo a la hoja de especificaciones, no pudimos continuar con la práctica. El programa compilaba correctamente pero no mostraba el ID correcto, en realidad solo se quedaron prendidos todos los leds.

Finalmente, en la sección de UART y Xbee, al principio tuvimos problemas compilando el programa ya que no sabíamos que debíamos compilar y posteriormente conectar el XBee, además no teníamos configurados correctamente los XBee, fue hasta que les asignamos los identificadores correctos que pudimos pasar a la parte de UART. En esta sección, lo más importante fue configurar correctamente los puertos de transmisión, recepción y correspondieran con los puertos físicos. Finalmente, pudimos configurar todo correctamente, y generamos comunicación en ambos sentidos.

5. CONCLUSIONES

- Leslie: Fue muy bonito poder configurar correctamente los microcontroladores y comunicarlos mediante I^2C, las cosas funcionaron de acuerdo a la teoría, pudimos trabajar de forma modular y obtuvimos el resultado esperado, que sin duda necesitaremos para el proyecto final. En el protocolo de SPI, aunque entendí como funcionaba de manera teórica no pudimos realmente compararlo contra I^2C, dado que ambos utilizan comunicación serial síncrona. En cuanto XBee y UART, al principio tuvimos muchos problemas para configurar correctamente el XBee en XCTU, pero después de eso, pudimos configurar correctamente la comunicación mediante UART.
- Diego: En esta práctica pudimos aprender a configurar el puente H que necesitaremos luego en el proyecto. También pudimos aprender de forma teórica como funciona SPI pero al final no pudimos avanzar con la práctica porque no supimos si el repositorio que usamos requería otras modificaciones adicionales a la asignación de puertos en la configuración del acelerómetro. Finalmente para la comunicación de los XBee, aunque al principio fue complicado, el protocolo en sí mismo (UART) funcionó como debía. Tal vez en el futuro sea importante tener un manual detallado de la configuración de XBees con XCTU.

6. ROL O PAPEL

- Leslie: Investigación de comunicación maestro-esclavo en arduino, ayuda a buscar los parámetros correctos en la hoja de especificaciones del acelerómetro y investigación sobre XBee. Marco teórico.
- Diego: Implementación del puente H y generación de las funcionalidades del motor, configuración del acelerómetro, configuración de la comunicación con XBee. Limpieza de programas.

7. FUENTES CONSULTADAS

REFERENCIAS

[1] "Serial communication." [Online]. Available: https://learn.sparkfun.com/tutorials/serial-communication
[2] "I2c – what's that?" [Online]. Available: https://www.i2c-bus.org/
[3] "Master writer/slave receiver." [Online]. Available: https://www.arduino.cc/en/Tutorial/MasterWriter