Reporte de la Práctica 3: Protocolos de Comunicación

Andrea Marín Alarcón
Luis Felipe Landa Lizarralde
Miguel González Borja

Resumen—Dado que existen varios protocolos de comunicación entre microcontroladores y dispositivos, se buscó resaltar las diferencias entre estos. Para esto, se diseñaron tres sistemas distintos utilizando I^2C, *SPI*, *UART* y *Zigbee*. Se vio que el protocolo I^2C es una buena manera de establecer comunicación síncrona entre dos microcontroladores; *SPI* permite comunicación serial entre un microcontrolador y un *peripheral*; la combinación de *UART* y *Zigbee* otorga una comunicación asíncrona e inalámbrica entre un microcontrolador y una computadora.

1. INTRODUCCIÓN

La comunicación entre microcontroladores y dispositivos externos permite una variedad de usos. El microcontrolador puede recibir datos de un sistema externo o enviar señales de control a un dispositivo. La diversidad de hardware, software y usos exige la implementación de protocolos estándar para compartir información entre los dispositivos.

La comunicación síncrona es fundamental cuando se requiere un control rígido sobre el orden de ejecución de un sistema. Si un dispositivo requiere cierta información antes de realizar una operación, es necesario que reciba estos datos antes de que se ejecute la operación. Compartir el reloj entre dispositivos deja que los datos enviados sean leídos en el orden deseado con el formato preestablecido. Las aplicaciones que necesitan este tipo de comunicación pueden ser sistemas que implementan retroalimentación. La comunicación síncrona entonces permite un mayor control sobre la transferencia y lectura de mensajes, sin embargo, al compartir el reloj, se debe enviar una mayor cantidad de información que conlleva mayores costos.

La comunicación asíncrona a su vez tiene otras ventajas y desventajas. Esta manera de transmitir y recibir mensajes carece de un reloj compartido entre los sistemas. Por esta razón los mensajes deben cambiar su estructura para brindar la información que permita su lectura correcta. Este tipo de comunicación es el que se utiliza en el internet y entre computadoras en general.

Los distintos tipos de comunicación permean ámbitos diversos en el mundo de las aplicaciones. No existe un único protocolo que resuelva todas las necesidades con las restricciones que impone el problema particular. Por esto mismo hay tal variedad en estos protocolos y se vuelve de interés analizar en qué situación el uso de uno sobre otro brinda beneficios o costos.

A continuación se presenta el Marco Teórico que ofrece una explicación a mayor detalle de los elementos y protocolos utilizados en el experimento. Seguido de eso, la sección de Desarrollo resalta los aspectos importantes de la implementación del experimento junto con la manera en que se utilizaron los elementos mencionados en el marco teórico. El apartado de Resultados discute los valores obtenidos en el experimento a su vez que menciona errores o fallos presentados durante el desarrollo. Por último, las Conclusiones resaltan las perspectivas de los miembros del equipo y lo que se consideró de mayor relevancia en la ejecución del experimento.

2. MARCO TEÓRICO

En esta práctica se implementaron tres protocolos de comunicación: I^2C, UART y SPI. Asímismo se trabajó con XBees, con el acelerómetro del FPGA y con un puente H para controlar un motor.

Los dispositivos electrónicos se comunican entre sí enviando bits de datos por medio de cables físicos que conectan dichos dispositivos. Estos bits pueden ser transmitidos de manera serial o paralela. Cuando se hace de manera paralela se envían múltiples bits al mismo tiempo a través de distintos cables. Por el otro lado, en la comunicación serial, los bits se envían uno por uno mediante un solo cable [1].

Universal Asynchronous Receiver/Transmitter, o UART, no es un protocolo se comunicación, si no un circuito físico cuyo propósito es implementar una comunicación serial. Aquí no se tiene una arquitectura de maestro-esclavo, si no que tenemos el emisor y el receptor. Para comunicarse entre dos UARTs sólo se necesitan dos cables, Tx (*transmitting*) y Rx (*receiving*).

Para transmitir los mensajes se crean paquetes los cuales contienen [2]:

- **Bit de inicio**: También se le conoce como bit de sincronización y se coloca antes de los datos. Para iniciar la transmisión, el UART emisor baja la línea de datos, lo que indica al receptor que empieza el flujo de información.
- **Bits de datos**: Tras iniciar la transmisión se comienzan a enviar los bits del mensaje. Generalmente son de 5 a 9.

- *Bit de paridad*: Permite que el receptor verifique si los datos obtenidos están bien o no. Se usan de 0 a 1 bits de paridad.
- **Bit de fin**: Se coloca al final del paquete de datos y pueden ser de 1 a 2 bits. Para notificarle al receptor que el mensaje terminó, el emisor sube la línea de datos y la mantiene arriba.

Esta estructura se puede ver en la Fig. 1

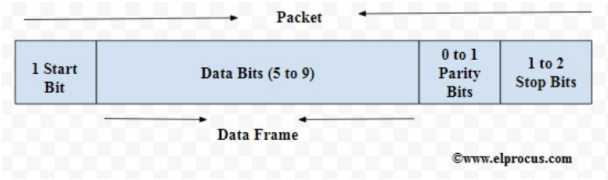

Figura 1. Estructura de un paquete enviado por UART

La característica principal de la comunicación por medio de UART es que ésta es asíncrona. Esto quiere decir que no existe una señal de reloj para sincronizar los bits de salida del UART emisor con los bits de muestreo del UART receptor. Esta es la razón por la cual se agregan bits de inicio y de fin a los paquetes de datos [3].

Es importante que ambos UARTS tengan la misma velocidad de transmisión para que puedan comunicarse correctamente. Pues, cuando el UART receptor detecta el bit de inicio, éste comienza a leer los bits de entrada a la velocidad de transmisión establecida. Además, ambos UARTS deben de estar configurados para leer y enviar la misma estrictura del paquete de datos [1].

Serial Peripheral Interface, o SPI, es un bus de interfaz que normalmente se utiliza para enviar datos entre microcontroladores y periféricos pequeños tales como sensores, tarjetas SD, resgistros de desplazamiento y otros. Utiliza líneas de reloj y datos separadas además de una línea con la que se elige con cuál dispositivo se quiere hablar.

Como se utilizan dos líneas diferentes para los datos y para el reloj, SPI es un bus de datos síncrono, es decir, ambas partes de la comunicación están en sincronía [4]. Un beneficio del protocolo SPI es que los datos se pueden enviar sin interrupciones, a diferencia de UART e I^2C donde la información se envía en paquetes de un número de bits determinado. Además, con SPI se pueden tener múltiples esclavos.[1]

La comunicación entre SPI puede ser a través de 3 o 4 cables. En ambos casos se tiene la línea *Slave Select* (SS) y la línea del reloj *Serial Clock (SCKL)* que es controlada por el maestro. Sin embargo, en el primer caso, la otra lína es *Serial Data In/Out* (SDIO) [5], que es por donde el maestro envía y recibe información. Con 4 cables se tiene una línea *Master Out, Slave In* (MOSI) por la cual envía datos el maestro y otra línea *Master In, Slave Out*(MISO), por la que recibe datos el maestro. En esta práctica se utilizó la primera versión del SPI.

Para iniciar la transimisión, el maestro baja la línea SS. Cuando se tienen múltiples esclavos cada uno tiene su línea SS, por lo que el maestro baja la línea del esclavo con el cual se quiere comunicar. Si la operación es de lectura entonces el esclavo comienza a enviar información por la línea SDIO; si la operación es de escritura, el maestro es quien envía datos por SDIO.

SPI tiene cuatro modos de operación dependiendo de la configuración de dos parámetros: *clock polarity* (CPOL) y *clock phase* (CPHA). El primer parámetro determina el estado inicial de SCLK, así si CPOL es cero, SCKL comienza en 0 y su primer flanco es ascendiente; si CPOL es uno, entonces el primer flanco de SCKL es descendiente. Por el toro lado, CPHA define cómo se van a leer los datos. Si CPHA es cero, entonces el primer bit se escribe en el flanco decreciente de SS y se lee en el primer flanco de SCKL. Si CPHA es uno, entonces los datos se escriben en el primer flanco de SCKL y se leen en el segundo [5].

La ventaja que se tiene con la configuración de 4 cables de SPI es que se puede enviar y recibir información al mismo tiempo pues se tiene una línea asignada a cada tipo de operación. Una de las desventajas de SPI es que no hay manera de saber si los datos llegaron al destino de manera exitosa.

Inter-Integrated Circuit, o I^2C, es un protocolo serial para interfaces con dos cables utilizado para conectar dos dispositivos de baja velocidad, tales como: microcontroladores, EEPROMs, convertidores A/D o D/A y otros periféricos en sistemas embebidos [6]. Con este protocolo se pueden conectar múltiples esclavos a un maestro y también se pueden tener múltiples maestros que controlan a uno o más esclavos.

Como ya se mencionó, este protocolo sólo utiliza dos cables para transmitir información:

- **Serial Data (SDA)**: Es la línea por la cual tanto el maestro como el esclavo transmiten información.
- **Serial Clock (SCL)**: La línea que transmite la señal del reloj

Al ser un protocolo de comunicación serial, los datos se transmiten bit por bit a través de SDA [7].

Cada esclavo necesita tener una dirección única de 7 bits para que el o los maestros se pueda comunicar con él. Por otro lado, el maestro no necesita ninguna dirección y además es quien genera la señal del reloj (SCL) [6].

Para comenzar la comunicación, el dispositivo maestro mantiene SCL arriba y baja SDA. De esta manera notifica a todos los esclavos que va a comenzar la transmisión, a esto se le llama *start condition*. Posteriormente se envía la dirección de 7 bits del esclavo con quien se quiere hablar, seguido de un bit llamado R/W. Con este bit se indica si la operación a realizar es de lectura, 1 (el maestro pide datos del esclavo) o de escritura, 0 (el maestro envía datos al escalvo).

Después de que se envía la dirección, se empiezan a transmitir los datos en paquetes de 8 bits. La información se carga al SDA después de un flanco descendiente del SCL y se muestra tras un flanco ascendiente. Para terminar la transmisión se envía una *stop condition*, la cual consiste en una transición de 0 a 1 del SDA, después de una transición de 0 a 1 del SCL donde el SCL se mantiene arriba [8].

Entre cada paquete del mensaje se manda un bit de *acknowledge/no-acknowledge*. Así, si el paquete fue recibido exitosamente, se regresa un bit ACK al emisor del mensaje [7]. En la Fig. 2 se puede ver la estructura de un mensaje.

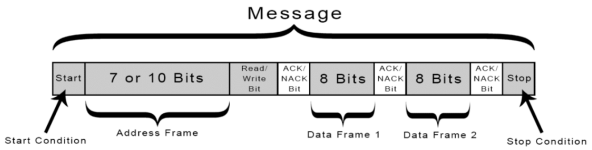

Figura 2. Estructura de un mensaje enviado por I^2C

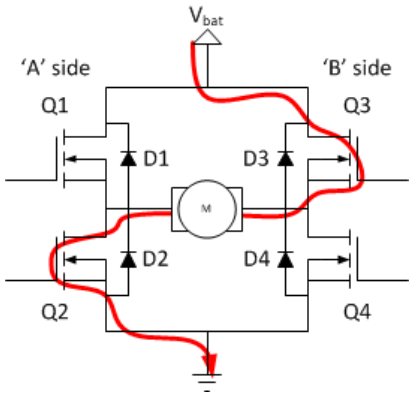

Figura 3. Diagrama de un puente H

Para la comunicación por medio de UART se utilizó un dispositivo conocido como XBee. Un XBee son chips que se pueden comunicar de manera inalámbrica entre ellos. De esta manera puedes evitar colocar un cable físico que una los dos dispositivos. Fueron diseñados para aplicaciones de requieren un alto tráfico de datos, baja latencia y una sincronización de comunicación predecible. Los XBee están basados en el protocolo Zigbee [9].

Para dos de los sistemas implementados se utilizó un un puente H para controlar un motor. Un puente H es un circuito electrónico que permite que un motor eléctrico DC gire a la izquierda, a la derecha, entre en freno pasivo y en freno activo. Este circuito se construye con 4 interruptores. Así, dependiendo de cuáles interruptores estén abiertos/cerrados el motor se mueve en una dirección o se frena. En la Fig. 3 se puede ver el diagrama de un puente H y cómo fluye el voltaje a través de éste.

Por último se utilizó el acelerómetro del FPGA. Un FPGA, o *Field Programmable Gate Array*, es un dispositivo cuyas interconexión y funcionalidad pueden ser programadas. Estos dispositivos te permiten diseñar circuitos digitales y reconfigurarlos cuantas veces sean [10].

El FPGA utilizado cuenta con un acelerómetro, el cual es un dispositivo electromecánico que detecta las fuerzas de aceleración, ya sea estática o dinámica. Las primeras incluyen la gravedad, y las segundas vibraciones y movimiento [11].

3. DESARROLLO

3.1. I^2C

Se conectaron los pines para SCL y SDA de ambos arduinos. Después, Se dio de alta el esclavo con dirección 9 y espero instrucciones del maestro. El maestro estaba conectado a dos botones y, dependiendo de cuales estuvieran presionados, enviaba mensajes al esclavo a tiempos periódicos.

El esclavo estaba conectado a un motor a través de un puente H. Al recibir cada byte, el esclavo decidía si dejar el motor en freno pasivo, freno activo, girarlo a la derecha o girarlo a la izquierda. Imágenes de los sistemas relevantes pueden ser observadas en las Figuras 5 y 6.

3.2. SPI

Para poder lograr la comunicación entre el FPGA y el acelerómetro, se utilizo SPI a 3 cables (*Clock, Slave Select, y Serial Data I/O*), con el FPGA como maestro y el acelerómetro como esclavo 0. En cuanto al reloj del protocolo, se utilizo una señal a 5MHz con polaridad y fase 1, por lo que el reloj iniciaba con un flanco hacia abajo cuando se seleccionaba algún esclavo para transmisión de datos, escribiendo datos en los flancos hacia abajo del reloj y leyendo en flancos hacia arriba.

Una vez que se configuró la comunicación, se modificaron las siguientes propiedades de la configuración del acelerómetro.

- Deshabilitar las funciones de TAP y DOUBLE TAP
- Establecer BANDWIDTH de 200Hz
- Deshabilitar el modo LINK
- Deshabilitar el modo AUTOSLEEP
- Habilitar el modo MEASURE
- Establecer el acelerómetro en modo NORMAL

Finalmente, se configuraron los dip switches del FPGA como se muestra en la Tabla 1, y se conectaron dos pines del FPGA para controlar el puente H mostrado en la Figura 4 de tal modo que el motor avanzara cuando el bit mas significativo del valor desplegado fuese 1, y retrocediera si fuese 0. Una imagen del sistema puede ser observada en la Figura 7, y una corta demostración puede ser vista aquí.

3.3. $Zigbee$

Primero, se configuraron las direcciones del par de XBees para establecer la comunicación a 9600 baudios. Una vez que el par XBee se podía comunicar, dejamos uno conectado a una computadora y el otro se conectó al ATmega2560 como se muestra en la Figura 8.

Luego, se configuro el microcontrolador para prender el LED verde si recibía 0x01, el amarillo si recibía 0x02 y el rojo si recibía 0x04. Además, si se presiona el botón, el ATmega debe enviar un mensaje al otro XBee con el byte 0xFF.

Dip Switches	Valor a Desplegar en LEDs
0000	ID
0001	X_l
0010	Y_l
0100	Z_l
1001	X_h
1010	Y_h
1100	Z_h

Cuadro 1
Configuración de Dip Switches del FPGA para el sistema 4.2

4. RESULTADOS

4.1. I^2C

Se conectaron las los pines SCL y SDA de ambos Arduinos. Luego, el esclavo se dio de alta con dirección 9, esperando transmisiones del maestro. Una imagen del maestro puede ser observada en la Figura 6. En este sistema, dependiendo de los botones presionados, se envian datos distintos al esclavo a tiempos periodicos.

En la Tabla 2 se puede observar la configuración del nodo esclavo. Este esta conectado a un motor a través de un puente H, visto en la Figura 4. Una imagen del esclavo puede ser vista en la Figura 5. Al recibir un byte, el esclavo lo interpretaba como un entero sin signo de 8 bits. Si su valor era 1, el motor gira a la derecha. Si el valor es 2, gira a izquierda. Si el valor es 3, se usa el freno activo, y para cualquier otro valor el motor tiene freno pasivo.

4.2. SPI

En la Tabla 3 se pueden observar los valores de los puertos del nodo maestro de SPI a 3 cables. Notar que se utilizo un valor de $clk\_div = 5$ para reducir la frecuencia original del FPGA (50MHz) a la frecuencia del acelerómetro ADXL345 (5 MHz). La configuración del ADXL345 puede ser vista en el Cuadro 4, y la de los pines del FPGA en el Cuadro 5. Una imagen del sistema puede ser observada en la Figura 7, y una corta demostración puede ser vista aquí. Cuando el bit mas significativo del valor leído es 1 (representado por el LED 7 del FPGA), el motor gira hacia el LED amarillo. De otro modo, gira hacia el LED rojo.

4.3. $Zigbee$

Primero, se configuraron los XBee como se muestra en el Cuadro 6 y se probo la comunicación entre ambos módulos desde XCTU.

Luego, se conecto el ATmega2560 como se muestra en el Cuadro 7 y la figura 4.3. Al recibir un byte a través del XBee, el micro-controlador toma los 3 bits menos significativos del byte como nuevos valores para los LEDS (b_2 = Rojo, b_1 = Amarillo, b_0 = Verde). Además, cuando el botón del protoboard es presionado, se genera una interrupción que manda el byte *0xFF* al otro XBee.

Pin	I/O	Elemento
21	Input	SCL
20	I/O	SDA
11	Output	Puente H - Izq
12	Output	Puente H - Der

Cuadro 2
Configuración del ATmega2560 para el sistema 4.1

Puerto	Valor
clk_div	5
cpol	1
cpha	1
addr	0

Cuadro 3
Configuración del nodo maestro de SPI para el sistema 4.2

Registro	Valor
DUR	0x00
BW_RATE	0x14
POWER_CTL	0x08

Cuadro 4
Configuración de registros del ADXL345 para el sistema 4.2

Pin	I/O	Elemento
GPIO_00	Output	Puente H - Der
GPIO_01	Output	Puente H - Izq

Cuadro 5
Configuración de pines del FPGA para el sistema 4.2

5. CONCLUSIONES

Aunque los protocolos de comunicación SPI e I^2C presentan grandes ventajas para recolectar información de sensores directamente, el uso de UART y Zigbee para la comunicación entre nodos XBee resulta mas útil para transmitir estos datos al microcontrolador central, pues permite hacerlo de manera inalámbrica.

-Miguel González Borja

Queda claro con la práctica que diferentes protocolos son útiles en situaciones diferentes. En la interacción entre la FPGA y el acelerómetro un protocolo inalámbrico sería superfluo pero en la transmisión de información entre computadoras es necesario. Las aplicaciones dictan el protocolo que se debe utilizar.

-Luis Felipe Landa Lizarralde

asdf

-Andrea Marín Alarcón

6. ROL

Miguel González Borja: Armado de circuitos para sistemas 4.1, 4.2 y 4.3. Programación del sistema 4.2. Secciones de Desarrollo y Resultados.

Luis Felipe Landa Lizarralde: Implementación de sistemas 4.1 y 4.2. Secciones de Abstract e Introducción.

Andrea Marín Alarcón: Implementación de sistemas 4.1 y 4.3. Sección de Marco Teórico.

7. FUENTES CONSULTADAS

[1] C. Basics, "Basics of the i2c communication protocol," accessed 2019-03-14. [Online]. Available: http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/

[2] E. . P. . Focus, "Basics of uart communication, block diagram, applications," accessed 2019-03-14. [Online]. Available: https://www.elprocus.com/basics-of-uart-communication-block-diagram-applications/

[3] C. Basics, "Basics of uart communication," accessed 2019-03-14. [Online]. Available: http://www.circuitbasics.com/basics-uart-communication/

[4] MIKEGRUSIN, "Serial peripheral interface (spi)," accessed 2019-03-14. [Online]. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

[5] S. Larson, "Spi 3-wire master (vhdl)," accessed 2019-03-14. [Online]. Available: https://www.digikey.com/eewiki/pages/viewpage.action?pageId=27754638

[6] i2c, "I2c info – i2c bus, interface and protocol," accessed 2019-03-14. [Online]. Available: https://i2c.info/

Propiedad	XBee 1	XBee 2
ID	0x3332	0x3332
DH	0x0000	0x0000
DL	0x0030	0x0032
MY	0x0032	0x0030
Baud Rate	9,600	9,600

Cuadro 6
Configuración de XBee para el sistema 4.3

Pin	I/O	Elemento
52	Input	XBee Tx
53	Output	XBee Rx
22	Output	LED Verde
23	Output	LED Amarillo
24	Output	LED Rojo
20	Input	Botón

Cuadro 7
Configuración del ATmega2560 para el sistema 4.3

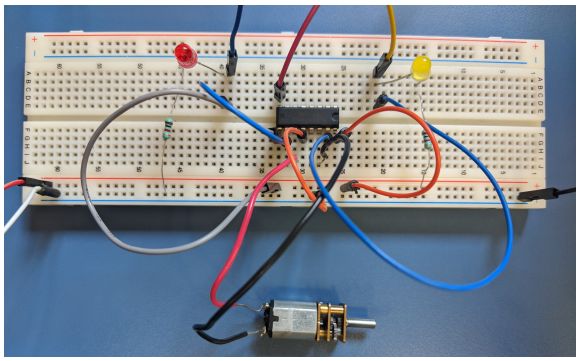

Figura 4. Motor conectado a través de un puente H

[7] C. Basics, "Basics of the i2c communication protocol," accessed 2019-03-14. [Online]. Available: http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

[8] SFUPTOWNMAKER, "I2c," accessed 2019-03-14. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c/all

[9] XBee.cl, "¿qué es xbee?" accessed 2019-03-14. [Online]. Available: https://xbee.cl/que-es-xbee/

[10] J. Rajewski, "What is an fpga?" accessed 2019-03-14. [Online]. Available: https://alchitry.com/blogs/tutorials/what-is-an-fpga

[11] H. Electrónica, "Abc del acelerometro," accessed 2019-03-14. [Online]. Available: https://www.5hertz.com/index.php?route=tutoriales/tutorial&tutorial_id=2

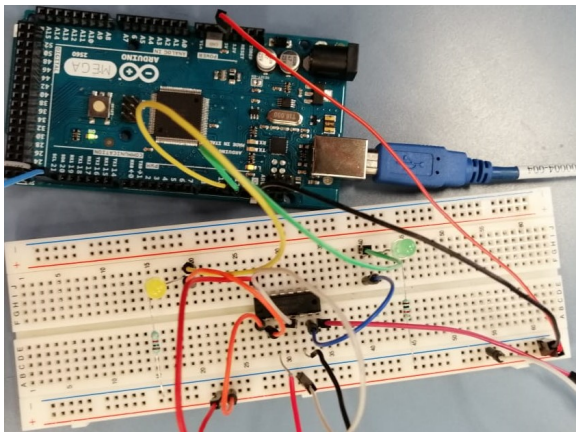

Figura 5. Sistema esclavo 4.1

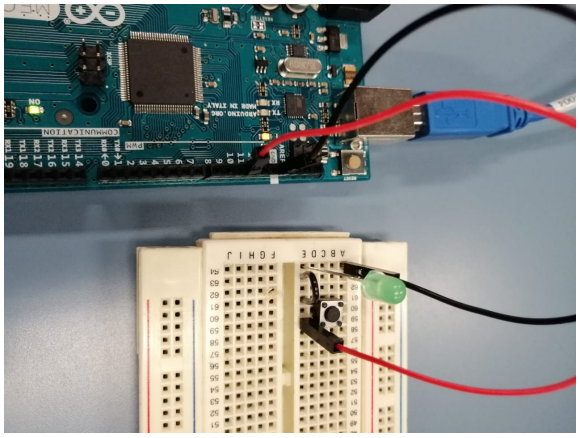

Figura 6. Sistema maestro 4.1

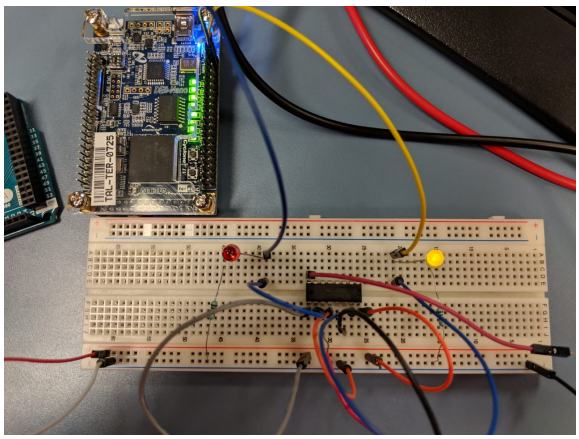

Figura 7. Sistema 4.2

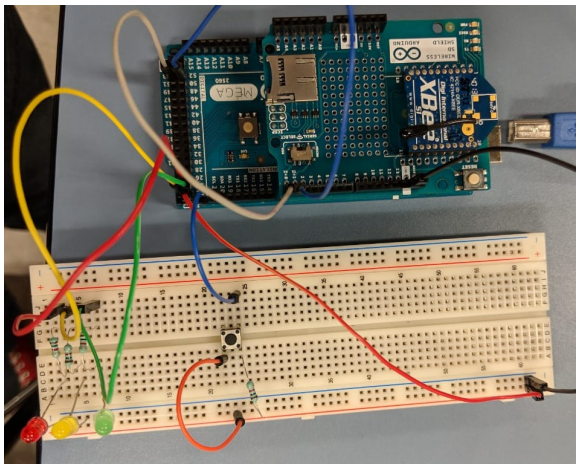

Figura 8. Sistema 4.3