E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Implementación de un sistema *Pick to light* con interface en Labview y comunicación ethernet

Grado en Ingeniería Eléctrica y Electrónica

Trabajo Fin de Grado

Aitor Redondo Ruiz

Cesar Elosua (Tutor UPNA)

Aitor Itoiz (Tutor AZKOYEN)

Pamplona, Junio de 2017

AZKOYEN GROUP

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ÍNDICE

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mis padres y a mi hermana su incondicional apoyo durante estos últimos cuatro años, en las buenas y en las no tan buenas. Gracias de verdad.

También me gustaría mencionar a la gente que me ha apoyado y ayudado durante mi estancia en las prácticas en Azkoyen. A Manolo Alvarado y Javier Elcuaz por su disponibilidad siempre que lo he necesitado aun teniendo mucho trabajo y quehaceres. A Aitor Itoiz por la confianza depositada en mí desde el principio en este proyecto, aun siendo un estudiante. A Daniel Jiménez por su amistad y por hacer la estancia de prácticas más amena en el día a día.

Gracias a mi tutor universitario del proyecto, César Elosua, por su disponibilidad, sus correcciones y su apoyo. No puedo dejar de mencionar también a Santiago Tainta, el cual siempre que lo he necesitado para diversas dudas ha estado ahí siempre proporcionándome otro punto de vista e, incluso, soluciones. También a todos esos profesores que, gracias a sus ganas por la enseñanza, han sembrado en mi esa curiosidad por este “mundillo”.

Por último, me gustaría nombrar a todos los amigos, tanto universitarios como no, por haberme hecho pasar unos años increíbles en todos los sentidos. Gracias a todos (Xabis, Felix, Lucas, Goratz, Yamal, Jon, Otazu, , …).

RESUMEN Y LISTA DE PALABRAS CLAVE

Con la nueva reestructuración en la zona de producción de la empresa AZKOYEN (medios de pago), se implementará un sistema *pick to light* en cada puesto de trabajo manual (32 en total) para la mejora de eficiencia en la producción.

Actualmente, el tiempo que transcurre en la fabricación de cualquier producto tiene una alta influencia en el precio final de este. Con la intención de reducir el precio en beneficencia de la empresa, se pretende diseñar un sistema el cual ayude a los operarios en su labor de una forma visual.

Este proyecto se centrará en el diseño electrónico de los LEDs, sensores...etc., así como en la programación del *Arduino* que controle todo el sistema y la interface que utilizará el operario para cambiar la secuencia en caso de requerirlo.

La interface se ha hecho con el programa *LabView*, así como la comunicación con el A*rduino* vía *ethernet*. El control de los LEDs estará programado a través de *Arduino*, el cual recibe una secuencia desde *LabView* por cable *ethernet*, la divide y la interpreta.

Palabras clave:

LabView, arduino, interface, ethernet, C++, LED, sensor, transistor, demultiplexor, layout, EEPROM

1. INTRODUCCIÓN

Los sistemas “pick to light” son muy utilizados hoy en día en almacenes para la preparación de pedidos. En ellos se guía al operario el producto que corresponde coger y la cantidad. De esta forma se ahorra tiempo en la preparación del pedido y se facilita al operario el trabajo.

En Azkoyen, aprovechando la nueva reestructuración de toda la fábrica de AMPASA (medios de pago), se ha introducido en cada uno de los 32 puestos de trabajo este sistema.

Utilizando un *Arduino* y la conexión ethernet, se conectará a un ordenador que dispone de una *interface* para poder manipular y ajustar la secuencia deseada.

Son varios los objetivos que concierne a este proyecto:

- Simplificar los procesos de producción
- Acortar los tiempos de trabajo
- Eliminar errores humanos
- Ahorrar en costes
- Facilitar su implantación
- Agilizar el flujo de procesos y productos

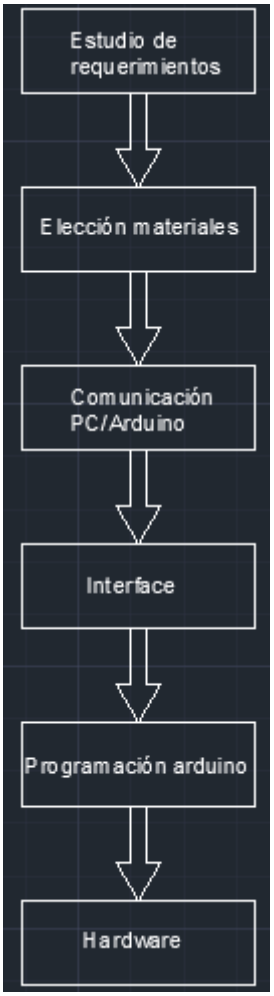
2. CONTENIDOS

Se ha diseñado el circuito electrónico de los LEDs y los sensores, así como la programación de la interface en *LabView* y la comunicación *Arduino*/*LabView* que gestionará los encendidos/apagados de los LEDs.

Como mejoras del proyecto, se ha implementado un botón en la interface para guardar la secuencia tecleada (en caso de querer guardarla) y un botón de escritura de una secuencia ya almacenada anteriormente y que se desee reutilizar. También se ha implementado en el código de *Arduino* la capacidad de medida de tiempos de ciclo de trabajo, lo cual es muy útil a nivel de organización industrial, y se visualizarán en *LabView* y se almacenan en un Excel estos tiempos.

Para el cumplimiento de estos objetivos se han realizado las siguientes tareas:

- Estudio de los requerimientos del proyecto.
- Elección de los materiales a utilizar (*arduino*, LEDs, sensores, demultiplexores…etc.).
- Desarrollo de las comunicaciones PC/*arduino*.
- Programación de la interface en *LabView*
- Programación de *Arduino*
- Programación de los sensores y LEDs



Todo ello ha sido desarrollado durante la estancia de prácticas en la empresa Azkoyen en el departamento de Ingeniería de producción junto a los tutores Manuel Alvarado, Javier Elcuaz y Aitor Itoiz.

El horario de estas prácticas ha sido de 8:00h-17:30h de lunes a jueves y de 8:00h-14:00h los viernes (del 15 de febrero al 23 de junio).

3. ESTUDIO PREVIO

El proyecto tiene que implementarse en los puestos de trabajo nuevos que va adquirir Azkoyen para su nuevo *layout*. Estos eran de las siguientes dimensiones:

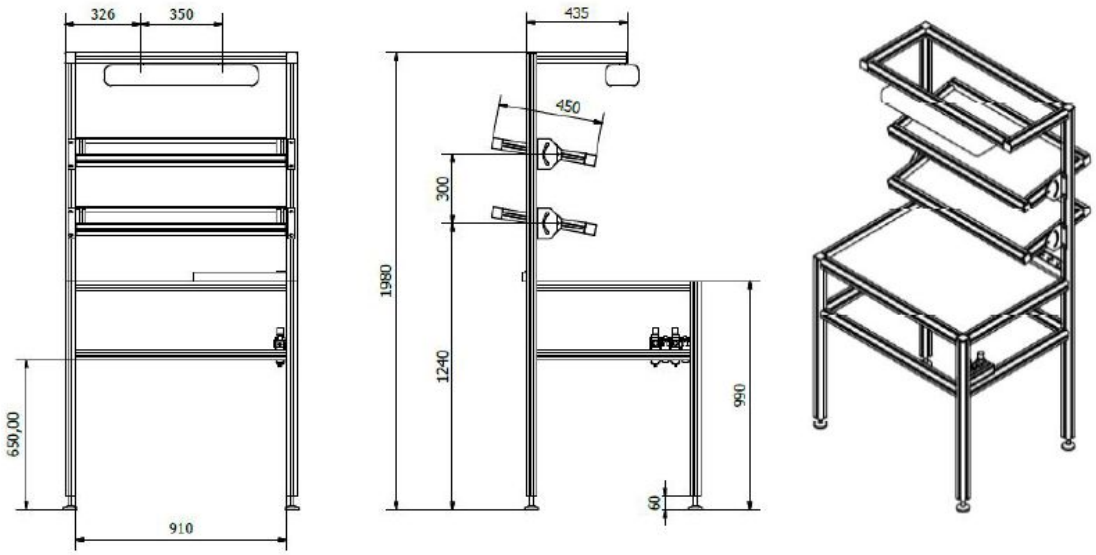


Fig. 3.1 Puesto de trabajo donde se implementará el sistema Pick to light

El sistema tiene que ir incluido en cada una de las baldas que dispone, la cual cada una de ellas puede tener como máximo 9 cajas con material para montar (en total 9x3=27 cajas).

La cantidad de cajas con material tiene que ser flexible, ya que las hay de distintas dimensiones y además algunos productos pueden requerir de menos material que otros. Por ello el sistema también tiene que ser flexible, con capacidad de introducir/quitar sensores y LEDs y cambiar la programación de una forma sencilla.

Tras fijar el proyecto detalladamente de cómo se va hacer, la empresa propuso hacerlo mediante el protocolo de comunicación estándar en domótica KNX.

3.1 Valoración del sistema *pick to light* mediante KNX

En primera instancia, por parte d la empresa se decidió el desarrollo de este sistema mediante un protocolo de comunicación estándar en domótica ya que en la empresa se cuenta con la instalación de las oficinas de esta forma.

La idea principal era poder interconectar todo el sistema de oficinas con la parte de producción (*pick to light*).

Tras varias semanas de análisis en aspectos como el precio, la compatibilidad de conexión, el funcionamiento del protocolo de comunicación, etc… se llegó a la conclusión, como más adelante se podrá observar, de no realizar el proyecto de esta forma. Tras proponer y explicar detalladamente a la empresa de los pros y contras, la empresa accedió a cambiar el diseño del *pick to light*.

TOPOLOGÍAS DE CONEXIÓN KNX

- Centralizado: este tipo de topología centraliza toda la información en un único punto. Generalmente se cablea en estrella y el centro de esta es el "cerebro". No existe comunicación alguna entre los sensores y actuadores, por lo que se podría decir que son elementos sin inteligencia propia.

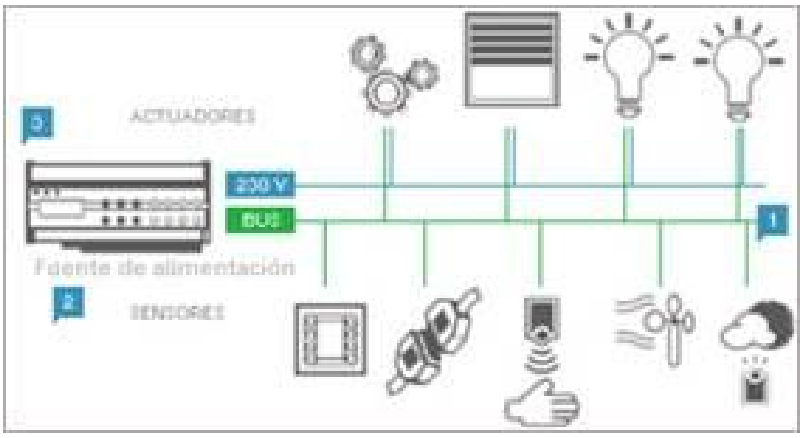


Fig. 3.1.1 Esquema conexión KNX

La información circularía del siguiente modo:

sensor→cerebro→actuador o actuador→cerebro→sensor

Este tipo de topologías disponen de varias ventajas:

-Bajo coste
-Puesta en marcha sencilla
-Utilización de elementos comerciales

Pero, al igual que todo, tiene sus partes negativas:

-Flexibilidad limitada
-Reconfiguración costosa
-Poco robusto
-Más cableado

- Distribuidos: En este tipo de topologías cada elemento es capaz de tratar la información y actuar en consecuencia. Lo más común es el conexionado de tipo bus y para ellos es necesario un protocolo de comunicaciones. La idea es conectar todos elementos al bus principal.

Fig. 3.1.2 Conexión bus KNX y elementos

En cuanto a las partes positivas de esta topología:

-Alta flexibilidad

-Ahorro de cableado respecto al centralizado

Y su parte negativa:

-Precio elevado (5-10 veces MÁS)

-Tiene que existir compatibilidad entre equipos

-Oferta de productos limitada

KNX es un sistema distribuido ya que cada dispositivo puede tratar la información y actuar en consecuencia.

PROGRAMACIÓN

En cuanto a la programación de este tipo de dispositivos, se vio que cuenta con un entorno de programación gratuito llamado ETS (*Engineering Tool Software*). Tras ver varios videos de su uso y leer algunos tutoriales, se creyó que no era el entorno ideal para la aplicación a desarrollar, ya que en el sistema *pick to light* es necesario una interface fácil manipulable y, aunque en ETS la programación fuese sencilla, no era lo deseado para esta aplicación.

Cabe destacar que ETS es un software de descarga gratuito pero que requiere de una licencia de pago para la programación de instalaciones que tengan más de 2 dispositivos.

PRODUCTOS NECESARIOS

Para la realización de este proyecto y de esta forma eran necesarios varios productos:

- Fuente de alimentación (una por cada 64 dispositivos, suponiendo un consumo de 10 mA por cada uno de ellos)
- 27 sensores de detección por cada puesto
- 27 LEDs por cada puesto
- Actuadores para encender/apagar los LEDs

Este era el material necesario para realizar el proyecto, contando en que por cada 64 dispositivos haría falta otra fuente de alimentación y que cada puesto de trabajo debería contar con 27 sensores y 27 LEDs.

PRECIOS

Uno de los temas importantes para cualquier proyecto reside en el precio o coste total de este.

Al empezar a buscar ofertas de productos con protocolo de comunicación KNX, se vio que estos productos eran demasiado caros para realizar este proyecto. Como se ha podido comprobar en el apartado anterior, la cantidad de material necesario era muy elevado y por consiguiente el precio también (sabiendo que cada aparato con protocolo KNX es muy caro).

Se han buscado precios para poder confirmar el alto coste. Estos son algunos de los ejemplos encontrados (y que serían necesarios en este proyecto):

Fig. 3.1.3 Actuador con protocolo KNX

Fig. 3.1.4 Fuente de alimentación

Como se puede observar, los precios de estos elementos son muy elevados para el objetivo final que se desea. Además, en el caso de los actuadores, el que se muestra en la imagen es de 6 entradas y 4 salidas. Para este proyecto serían necesarias 27 entradas y 54 salidas, por lo que el precio subiría mucho.

CONCLUSIONES

Tras analizar toda esta información durante varios días, se propuso a la empresa **NO** realizar el proyecto de esta forma, argumentando todos los criterios mencionados anteriormente.

Como la empresa estaba muy interesada en realizar este proyecto por aspectos estéticos (muchos comerciales visitan la empresa) como por aspectos de producción, se pensaron diferentes ideas para llevarla a cabo.

Una de las ideas inmediatas fue utilizar *Arduino* para el control de los sensores y los LEDs debido a su coste y su fácil manejabilidad. Además, existen muchos tipos de *Arduinos*, por lo que la cantidad de entradas/salidas necesarias se podría suplir sin ningún problema.

3.2 Sistema mediante KNX y arduino

Tras haber analizado el sistema KNX y conociendo el entorno *arduino* desde hace tiempo, se pensó en conseguir juntar estos dos sistemas mediante un *TPUART.*

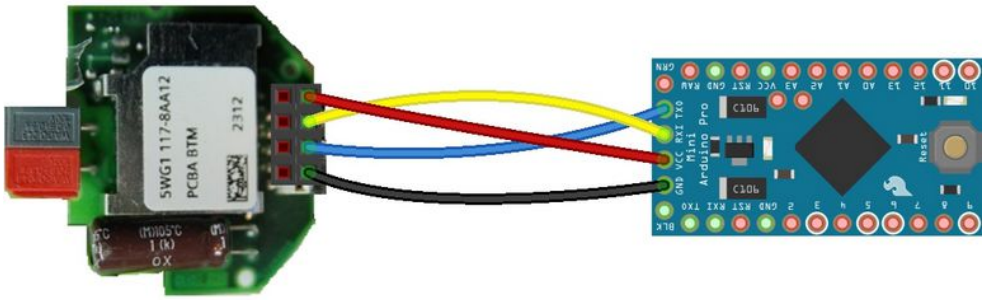


Fig. 3.2.1 Conexión entre Arduino y TPUART para la comunicación con el bus KNX

Sabiendo la sencillez para programar los sensores y LEDs mediante *Arduino*, la idea fue conectarlo al bus KNX para poder manipularlo mediante ETS.

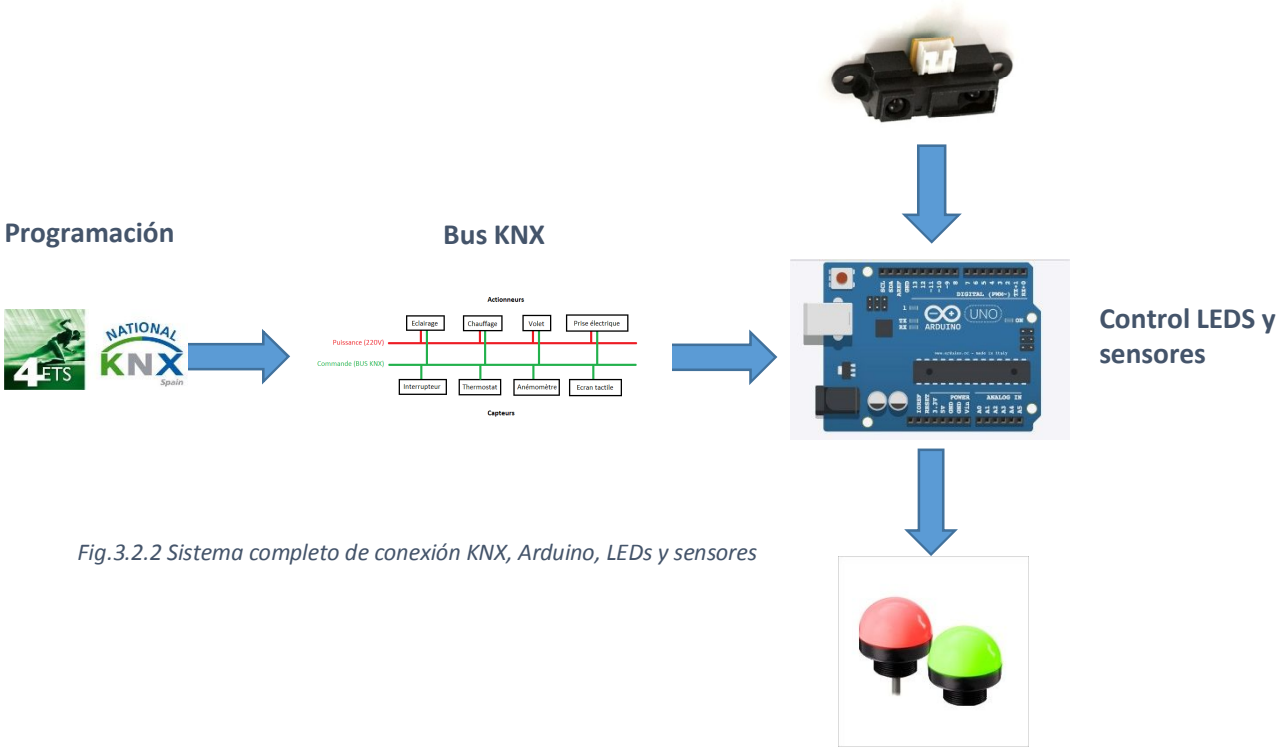


Fig.3.2.2 Sistema completo de conexión KNX, Arduino, LEDs y sensores

Esta forma de desarrollar el proyecto era una alternativa a la anterior, disminuyendo los costes.

De esta forma se tendría que pagar la licencia de KNX la cual es cara para solo realizar este proyecto.

Versión	Precio	Observaciones
ETS5 Demo	¡Gratis!	Máximo 5 dispositivos KNX por proyecto.
ETS5 Lite	€ 200,-	Máximo 20 dispositivos KNX por proyecto.
ETS5 Professional	€ 1000,-	Funcionalidad completa.
ETS5 Supplementary	€ 150,-	Se permiten máximo 2 licencias Suplementario por cada licencia Professional.

Fig.3.2.3 Precios licencia software KNX

En este caso correspondería al precio de *ETS5 Lite* o incluso al *ETS5 Professional* por lo que sería bastante costoso.

De todas formas, no es tanto el problema del precio como el de la complejidad de desarrollarlo de esta forma. KNX se trata de un protocolo de comunicación muy cerrado, por lo que conectar un dispositivo sin este protocolo al bus es algo bastante difícil. Esta conexión debería hacerse mediante algún dispositivo y tener conocimientos avanzados en temas de comunicaciones. Este ha sido el gran motivo por el cual se ha declinado esta opción.

Finalmente, se buscó una nueva alternativa: utilizando *Arduino*, manejar todo el programa de LEDs y sensores y que, en caso de querer manipular la secuencia de estos (cambiar el programa de *Arduino*) hacerlo con una conexión ethernet a un ordenador principal que tenga una interface muy intuitiva y manipulable.

3.3 Sistema mediante *Arduino, ethernet* y *LabView*

Tras el análisis de varias posibilidades surge la idea de conectar *Arduino* mediante *ethernet* a un ordenador donde se pueda manipular la secuencia de los LEDs fácilmente.

COMUNICACIÓN

Sabiendo que existen dispositivos para simplificar la comunicación ethernet para *Arduino* y una librería para poder controlarlo, es una buena idea ya que el entorno de A*rduino* es de sobra conocido. De esta manera la parte de comunicaciones es posible suplirla sin problemas.

I/O DE ARDUINO

Mediante las salidas de *Arduino* se programarán los encendidos de los LEDs tanto rojos como verdes y mediante las entradas se leerá el estado de los sensores para poder tomar decisiones de los encendidos y apagados.

INTERFACE

En un principio la interface se ha pensado en hacerla mediante HTML, comenzando a estudiar sobre este lenguaje de programación. Simplemente con este lenguaje, la página web que se crea queda algo pobre por lo que se ha buscado una nueva alternativa.

LabView es la alternativa escogida debido a su sencillez de construir una interface muy visual, lo cual es el objetivo de esta parte del proyecto, disponer de una interface visual e intuitiva para poder ser manipulada por cualquier persona.

La idea general de la interface es disponer de varios dibujos de LEDs para ir seleccionándolos cuando se quiera y en el orden deseado, generando de esta forma una

secuencia que será mandada en forma de *String* a *Arduino*, el cual deberá de separarla e interpretarla para ejecutar su programa principal de encender/apagar LEDs.

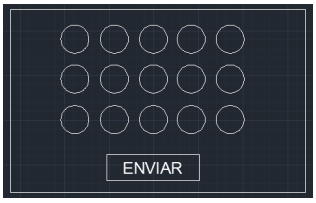


Fig. 3.3.1Ilustración simplificada posible interface

HARDWARE

En cuanto al hardware necesario para el control de todos LEDs y sensores se diseñarán 3 tarjetas electrónicas:

1- La primera que se comportará como un *shield* de *Arduino* para poder sacar fácilmente los pines de *Arduino* sin problemas y poder pasarlos a una tarjeta mayor donde estará el *hardware* principal.

   Arduino cuenta con unos dispositivos llamados *shields* los cuales se acoplan a los pines de *Arduino* para poder desempeñar otras funciones. Como ejemplo, dispone de *ethernet shield* el cual se encarga de poder realizar comunicaciones vía ethernet. También existen *shields* como *WiFi shield* para comunicaciones vía *WiFi* o *bluetooth shield.*

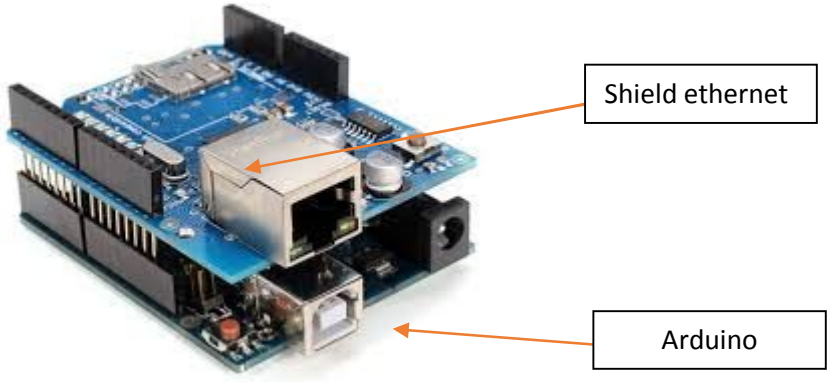


Fig. 3.3.2 Foto arduino y ethernet shield

2- La tarjeta principal será la que reciba todas las señales desde *Arduino* y desde cada sensor y las envíe tanto a *Arduino* para poder tomar decisiones como a los LEDs para encenderlos o apagarlos.

3- Por último, se diseñará una pequeña tarjeta para incluirla en la caja donde se acoplarán los LEDs y el sensor para poder controlar estos.

CAJA DE LEDS Y SENSORES

Se elegirá una pequeña caja donde poder acoplar tanto la tarjeta de LEDs y los sensores como la tarjeta electrónica diseñada. Esto se ha pensado en hacer para que a la hora de quitar/poner los LEDs se pueda hacer de una forma rápida y cómoda.

Se elegirá una caja a poder ser comercial para evitar el diseño y el coste de fabricación que pudiese suponer esto. La caja no deberá ser mayor a la anchura de las cajas donde se encontrará el material a montar y deberá ser lo más pequeña posible en cuanto a altura.

Fig. 3.3.3 Caja donde se incluirán LEDs, sensor y tarjeta electrónica

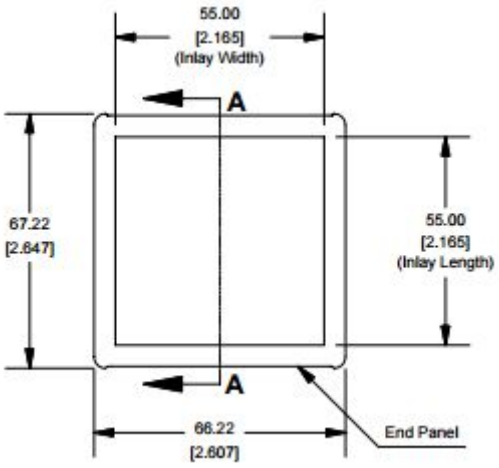


Fig. 3.3.4 Medidas caja elegida (TOP view)

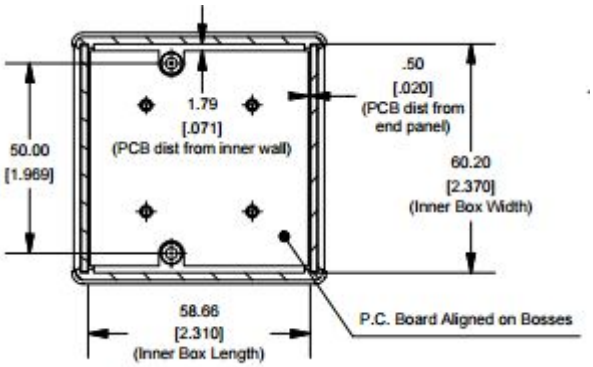


Fig. 3.3.5 Medidas caja elegida (Inside view)

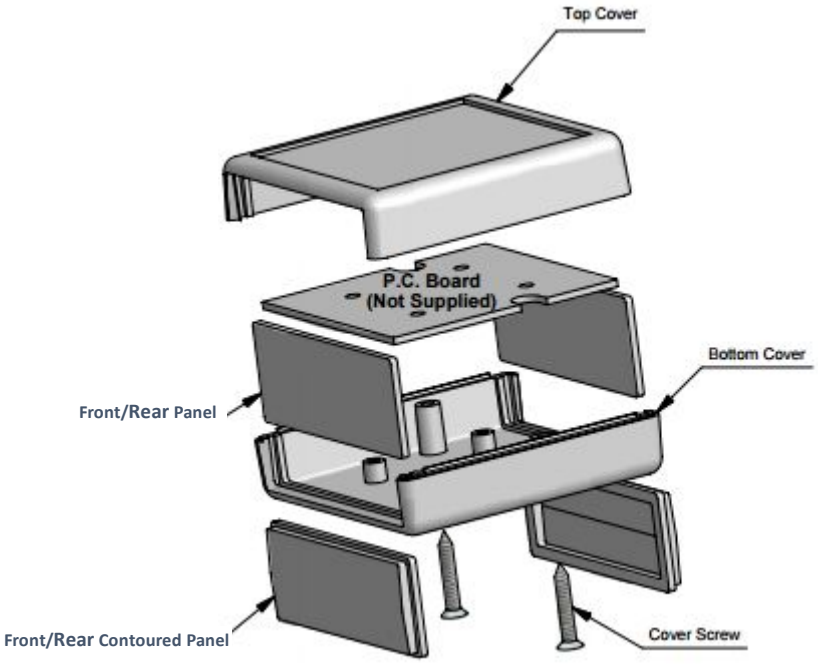


Fig. 3.3.6 Despiece de caja elegida

4. ENTORNO DE DESARROLLO

Con los objetivos bien marcados, se va a presentar el material necesario para este proyecto los cuales son el entorno *Arduino*, el programa *LabView*, sensores, LEDs, electrónica empleada y la comunicación *ethernet*.

4.1 Entorno *Arduino*

Arduino es una compañía de hardware libre la cual diseña y vende placas electrónicas las cuales van enfocadas a facilitar el uso de la electrónica y la programación a la gente.

Fig. 4.1.1 Logo de arduino

Estas placas se componen generalmente por un microprocesador *Atmel*, unos puertos digitales y analógicos que pueden ser tratados como entradas/salidas, un conector para la alimentación de la placa y otro conector USB para la programación del programa (y alimentación en caso de ser requerido).

Fig. 4.1.2 Ejemplo de placa arduino con todos los componentes

En este caso se ha elegido el modelo de **ARDUINO MEGA 2560**. Este modelo de *Arduino* consta de las siguientes características:

- Microcontrolador ATmega1280
- Voltaje de funcionamiento 5V
- Voltaje de entrada 7-12V (recomendado)
- 54 pines digitales I/O de las cuales 15 se pueden usar como salida PWM
- 14 pines analógicos de entrada
- Memoria flash de 128KB de los cuales 4KB utilizados por la placa
- EEPROM de 4KB
- Velocidad del reloj de 16MHz

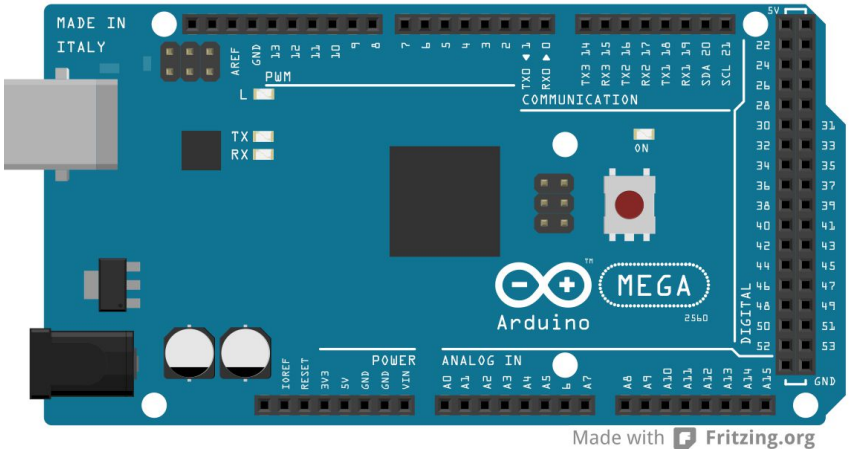


Fig. 4.1.3 Placa ARDUINO MEGA 2560

En cuanto al entorno de programación, *Arduino* dispone de su propio entorno el cual es gratuito para su descarga y el lenguaje es muy similar a C++. La totalidad del programa consta de una ventana con *SetUp* y *Loop* donde se pueden definir las variables a utilizar, así como el programa entero.

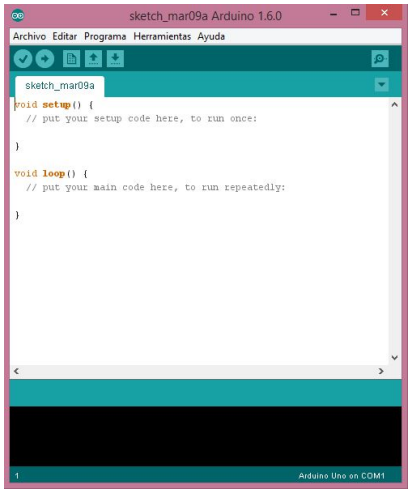


Fig. 4.1.4 Entorno programación arduino

4.2 *LabView*

LabView es un entorno de desarrollo para diseñar sistemas mediante un lenguaje visual/gráfico. Se utiliza el lenguaje G, llamado así ya que la G simboliza que es un lenguaje gráfico.

Es un programa creado por *National Instruments*, el cual lo creó para funcionar sobre máquinas MAC.

Puede ser utilizado con varios protocolos de comunicación (TCP/IP, GPIB, puerto serie, puerto paralelo...), puede interactuar con otros lenguajes de programación (.NET, DLL, Matlab/Simulink…), es sencillo de crear interfaces visuales y de fácil manejo, etc….

Contiene dos ventanas: una para la programación gráfica del programa (utilizando bloques) y la otra para generar la interface y visualizar los datos.

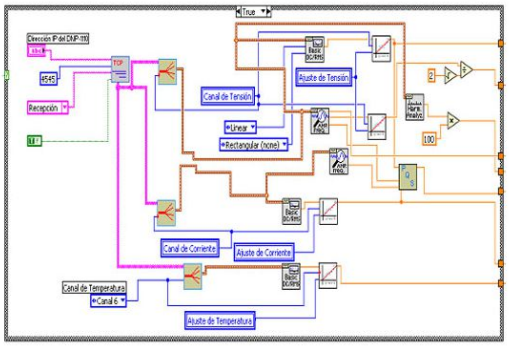


Fig. 4.2.1 Ventana de bloques y programación

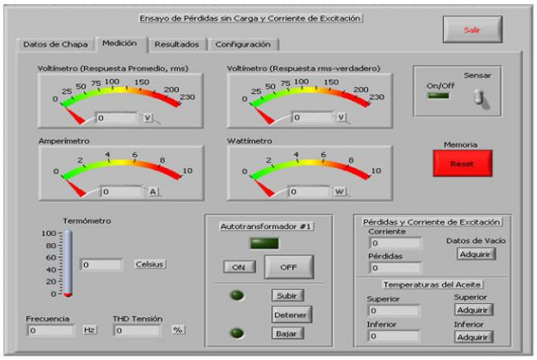


Fig. 4.2.2 Ventana de control y visualización de datos

4.3 Sensores

Uno de los temas más importantes de este proyecto son los sensores de detección ya que es muy importante que no den falsas detecciones o que no se les pase detectar cuando el operario ha introducido la mano.

Por ello se barajaron diversas opciones:

1- <u>Sensores de barreras:</u>

Se tratan de sensores infrarrojos los cuales una de las barreras emite una luz infrarroja y la otra barrera la recibe. En el momento de que la señal de la barrera receptora no recibe todo el haz de luz quiere decir que algo se ha interpuesto de por medio (la mano del operario) por lo que se trata de una detección.

Esta opción, a priori, era la mejor en cuanto a fiabilidad. El problema de estos sensores es su alto precio, por lo que se descartaron.

Fig. 4.3.1 Foto sensor de barrera infrarroja

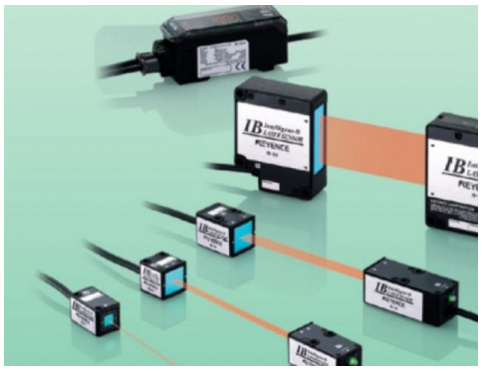


Fig. 4.3.2 Ejemplo sensor comercial Bitmakers

2- Sensores de infrarrojos:

Estos sensores funcionan igual que los anteriores, pero en vez de abarcar un área tan grande como los anteriores mandan y reciben el haz de luz desde un mismo dispositivo (reciben el rebote del infrarrojo). Debido a las pocas entradas analógicas que dispone *Arduino*, se ha buscado un sensor de estas características con señal de detección digital, sin éxito alguno ya que la gran mayoría se encuentran obsoletos. En consecuencia, se ha elegido un sensor analógico el cual con un comparador se podrá conseguir una salida digital.

El elegido ha sido el SHARP GP2Y0A41SK0F ya que cumple con las especificaciones requeridas y el precio de este es mucho menor que el anterior.

Fig. 4.3.2 Sensor SHARP analógico

Uno de los orificios se trata de un fotodiodo que emite una señal infrarroja y el otro es un fototransistor que se activa al recibir la señal de vuelta.

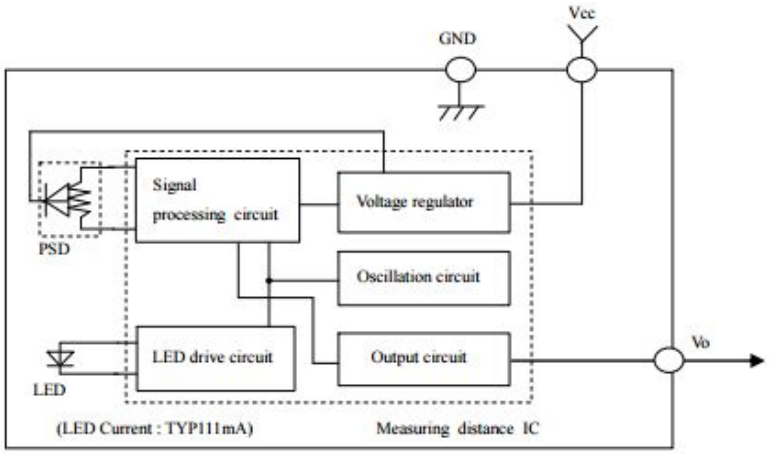


Fig. 4.3.3 Esquema funcionamiento de sensor SHARP analógico

Contiene 3 patillas de conexión: VCC (5V), GND y Vout. La salida (Vout) tiene el siguiente comportamiento respecto a la distancia:

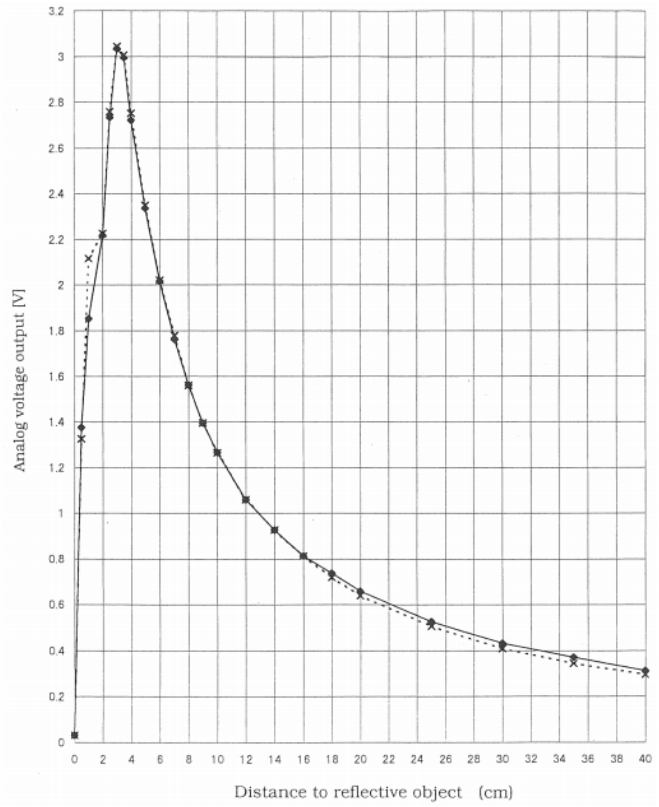


Fig. 4.3.4 Comportamiento sensor en función de la distancia

En función de la distancia del rebote de la luz infrarroja, en la salida del sensor tendremos distintos valores de tensión.

Enviando esta señal a nuestro *arduino*, podremos tomar decisiones de encendido/apagado de los LEDs.

4.4 LEDs

Los LEDs utilizados en el proyecto son los mismos que se emplean en el producto final POS1500 de AZKOYEN.

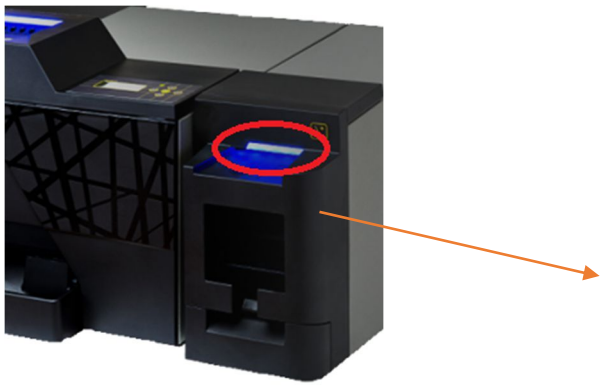


Fig. 4.4.1 Producto Cashlogy POS1500 de AZKOYEN

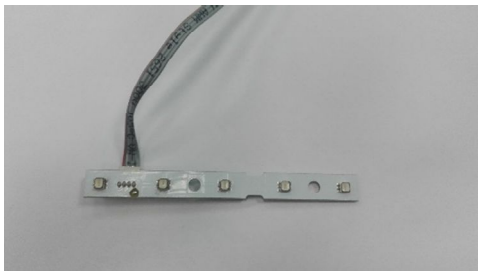


Fig. 4.4.2 Tira de LEDs utilizada en POS1500

Se trata de una tarjeta diseñada con los LEDs *LRTBGFTG*

Fig. 4.4.3 LED LRTBGFTG

El diseño de esta tarjeta electrónica, aunque se comentara más detalladamente en el apartado *hardware*, está compuesta del siguiente circuito electrónico:

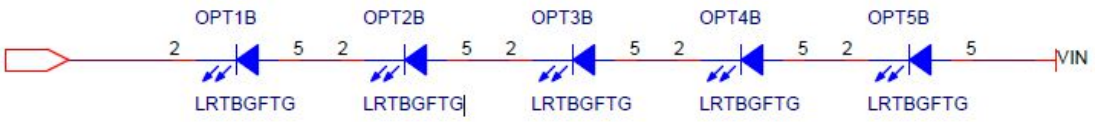
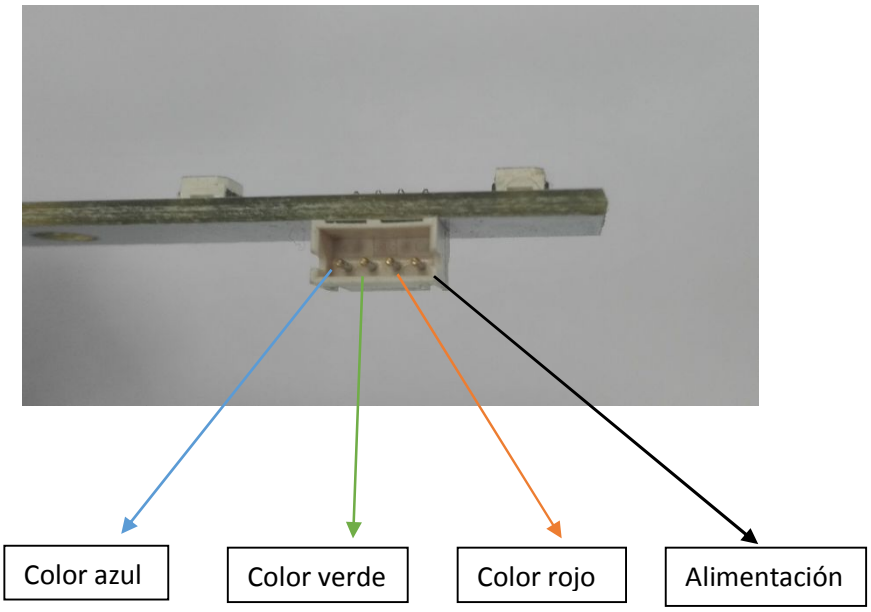


Fig. 4.4.4 Circuito electrónico LEDs

De esta forma se consigue reutilizar uno de los materiales que ya se utilizan en la empresa, evitando tener que cambiar de proveedor (algo importante a nivel de empresa) y dándole uso a la gran cantidad que hay en la fábrica.

Estas tiras de LEDs están diseñadas por el departamento de I+D de la empresa. Cuenta con 4 conectores (alimentación, color rojo, color verde y color azul) los cuales alimentándolos de una forma u otra se pueden generar diversos colores (aplicando *PWM* por ejemplo).

Fig. 4.4.5 Conectores tarjeta LEDs



4.5 Electrónica empleada

Para el desarrollo de la parte *hardware* de los LEDs y los sensores, se han barajado varias alternativas:

- Conexión de todos los componentes (27 LEDs y 27 sensores) a cada uno de los puertos que dispone nuestro *Arduino Mega*. En este caso, como este *Arduino* tiene 54 pines digitales nos bastaría, pero surge el problema que algunos de estos pines son utilizados para otros temas como comunicaciones y demás.

  Para solucionar este problema cuenta con 14 entradas analógicas que pueden ser tratadas como digitales, por lo que este problema quedaría resuelto. Esta solución no ha sido adoptada debido a la poca eficiencia que sacaríamos a las entradas/salidas de nuestro *Arduino*. En un futuro podrían ser necesarias más entradas o salidas y estaríamos muy limitados. Además, para encender los LEDs necesitaríamos transistores ya que *Arduino* no proporciona la suficiente tensión para encender los LEDs elegidos.

- Conexión mediante demultiplexores y transistores.

  Utilizando unas pocas salidas digitales conectadas para el control de los LEDs se pueden controlar varias salidas.

  Con demultiplexores de 4 canales se pueden controlar 2^4=16 LEDs.
  Con demultiplexores de 5 canales se pueden controlar 2^5=32 LEDs.

  En caso de encontrar estos últimos demultiplexores sería lo ideal debido a que con uno solo podríamos controlar todos los LEDs.

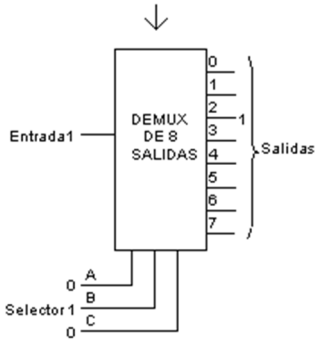


Fig. 4.5.1 Demultiplexor de 3 canales (8 salidas)

De todos modos, se han elegido otros demultiplexores de 3 canales (2^3=8 LEDs por cada demultiplexor) ya que son los utilizados en Azkoyen y a la hora de montarlos en las tarjetas lo ideal sería utilizar unos que ya utilizan en la empresa.

Los demultiplexores elegidos han sido los *74HC4051* que cuentan con 8 canales como se ha indicado anteriormente.

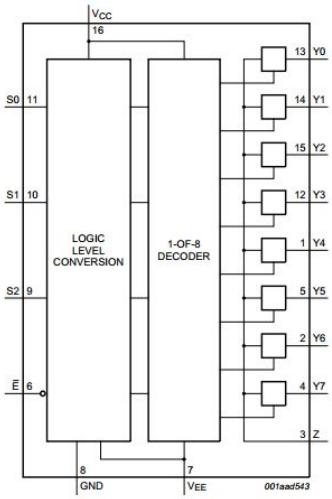


Fig. 4.5.2 Diagrama funcional demultiplexor 74HC4051

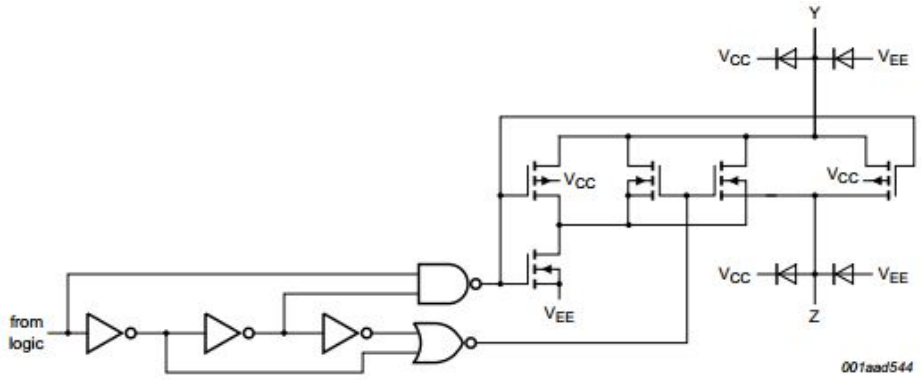


Fig. 4.5.3 Esquemático demultiplexor 74HC4051

Surge un problema con estas conexiones: la tensión de salida del demultiplexor es en 5V y la tira de LEDs a utilizar es de 12V. Además, la corriente que puede suministrar el demultiplexor es muy baja en comparación con la que necesitan los LEDs.

Como solución se propone el uso de transistores. Los transistores elegidos para el diseño han sido, al igual que en el caso de los demultiplexores, componentes que se utilizan en la empresa. En este caso se tratan de los transistores *BC337(NPN)*.

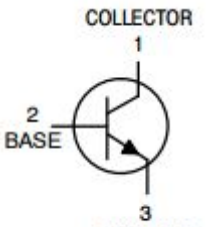


Fig. 4.5.4 Conexiones transistor BC337

Fig. 4.5.5 Transistor BC337

En la tarjeta electrónica principal anteriormente mencionada se encontrarán los demultiplexores para dar las señales de encendido a los LEDS. Estas señales irán a las tarjetas diseñadas que irán insertadas con su LED y sensor y en ellas se encontrarán los 12V de alimentación y los transistores para activar/desactivar los LEDs.

El esquema electrónico diseñado y que se utilizará será el siguiente (simplificado por tema de espacio):

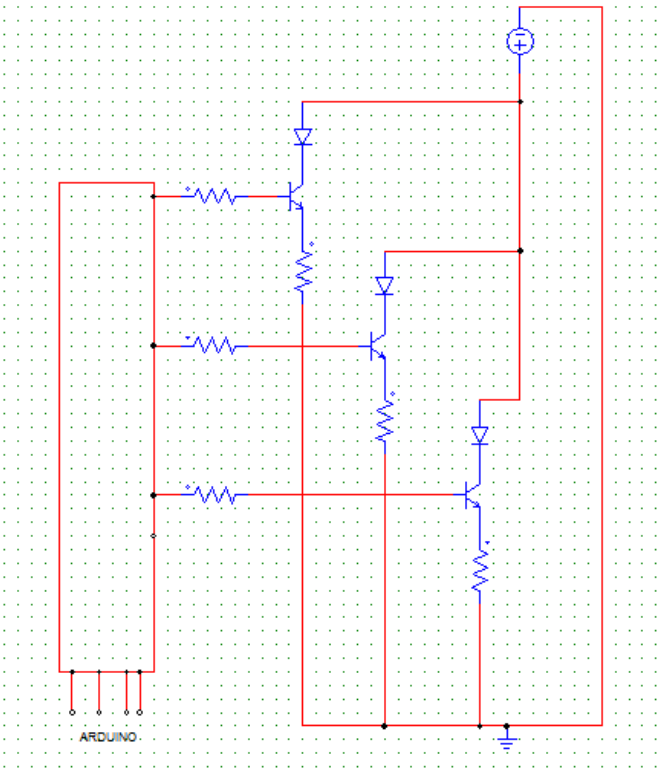


Fig. 4.5.6 Ejemplo de conexión de LEDs mediante demultiplexor y transistores

4.6 Protocolo de comunicación

El protocolo de comunicación elegido para comunicar el ordenador central con los distintos dispositivos (*Arduinos*) ha sido *ethernet*.

Este protocolo de comunicación es muy utilizado a nivel industrial para la configuración de varios dispositivos industriales en zona de producción.

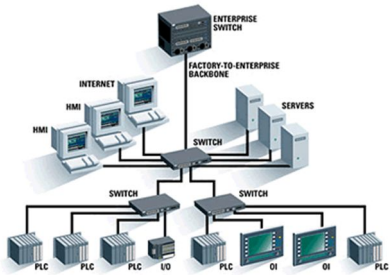


Fig. 4.5.1 Conexión de una red con comunicación Ethernet

Como en la empresa en la que se han realizado las practicas curriculares se encuentra instalado este protocolo se ha decidido realizarlo de esta forma. Además, *Arduino*

cuenta con un *shield* de *ethernet*, así como con una librería para el fácil control mediante este protocolo.

Los datos por ethernet se envían mediante una trama. Esta trama contiene datos como delimitador de inicio, la MAC de destino y origen, la longitud o los datos, por ejemplo.

La trama de ethernet es la siguiente:

Fig. 4.6.2 Campos de trama del protocolo de comunicación Ethernet

De todas formas, como se ha comentado anteriormente, no ha sido necesario entender en profundidad el protocolo de comunicación ya que, teniendo unos conocimientos básicos adoptados en asignaturas universitarias y con la librería de e*thernet shield*, ha sido sencillo conseguir comunicar los dispositivos.

4.7 Lenguajes de programación

Han sido dos los lenguajes de programación utilizados para poder realizar este proyecto final de grado:

Lenguaje *Arduino*:

Arduino utiliza un lenguaje muy parecido a C++ el cual se había utilizado en diferentes asignaturas del grado.

Contiene dos funciones principales: *void setup() y void loop()*. Además de estas, se pueden definir distintas funciones para realizar otras tareas.

Un ejemplo de un programa sería el siguiente, utilizando las dos funciones básicas:

Fig. 4.7.1 Ejemplo programa Arduino encer LED

En la primera función se definen los puertos a utilizar, las variables globales...etc.

La segunda función se trata de un bucle infinito. En esta parte del código es donde se escribe el programa que va a realizar nuestro *arduino*.

Se pueden definir más funciones que estas dos que se acaban de comentar para realizar cualquier otro tipo de trabajo.

Lenguaje *LabView*:

LabView utiliza una programación visual utilizando distintos bloques. Es un tipo de programación distinta a la de código escrito, pero más intuitiva.

Gracias a asignaturas del grado se conocía de antes este tipo de lenguaje, aunque muy brevemente. Aun y todo, gracias al proyecto se ha conseguido mejorar el nivel de programación en *LabView*.

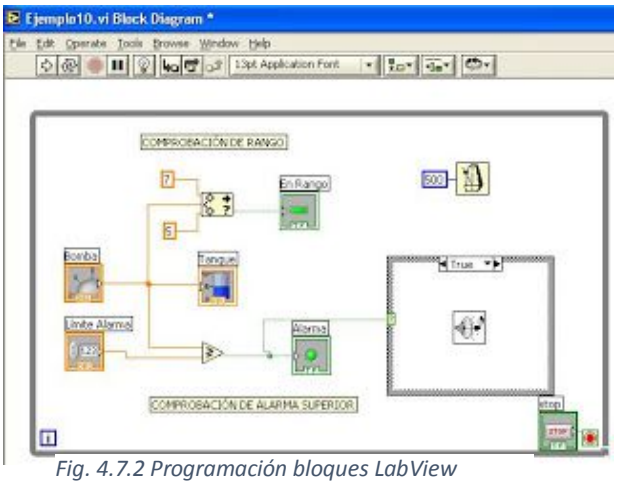


Fig. 4.7.2 Programación bloques LabView

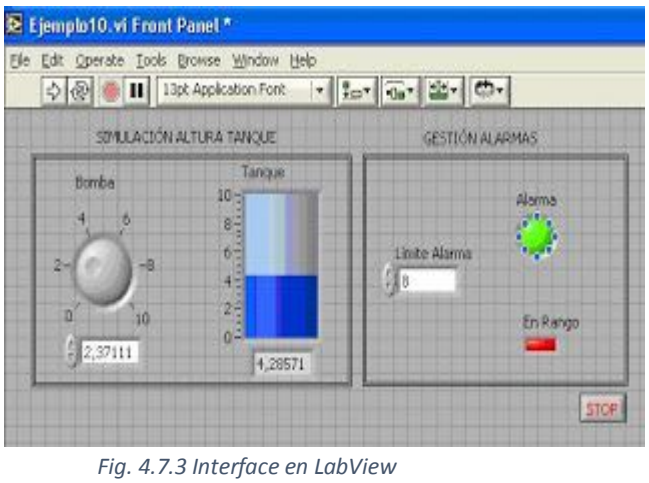


Fig. 4.7.3 Interface en LabView

5. METODOLOGÍA EMPLEADA

Para conseguir cumplir los objetivos marcados desde la empresa, se ha decidido que el sistema a montar tendrá la siguiente forma:

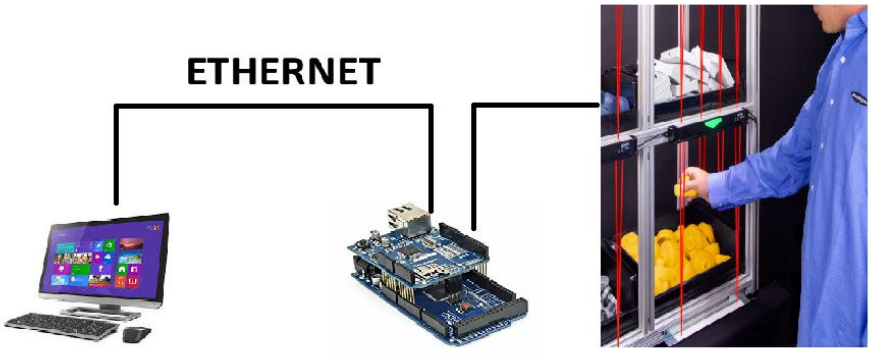


Fig. 5.1 Esquema general del sistema Pick to light

El puesto a montar sería de la siguiente forma:

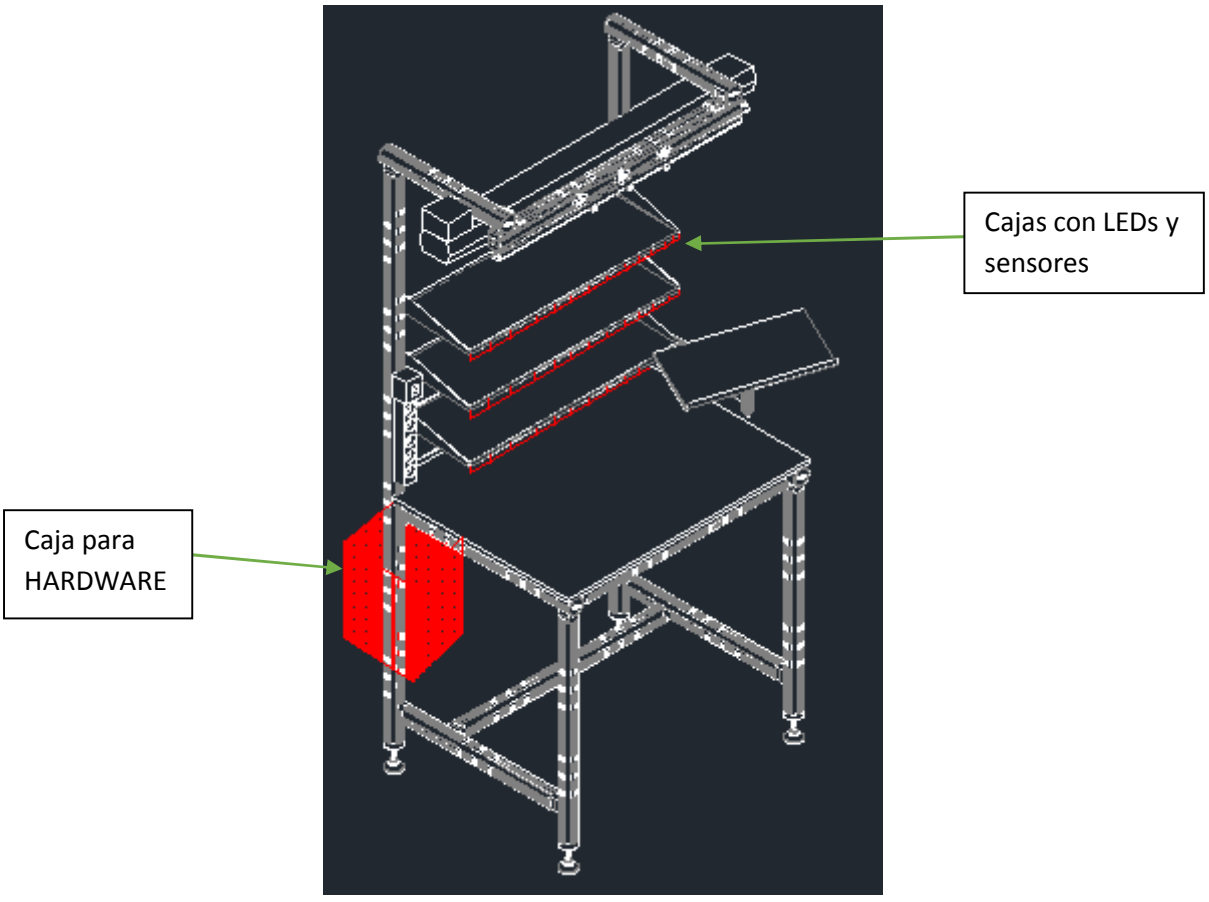


Fig. 5.2 Puesto de montaje

Este proyecto se puede dividir en dos partes muy definidas: la parte *hardware* y el *software.*

Hardware:

Finalmente se ha decidido realizar 3 tarjetas electrónicas para toda la parte del *hardware*: una de forma que funcione como *shield* de *Arduino* y poder sacar los pines deseados de una forma sencilla, otra para controlar los encendidos/apagados de los LEDs y las señales de los sensores, así como manejar todo el cableado a cada caja con LEDs, y finalmente las tarjetas que irán incluidas en cada caja con sus LEDs y sensores. Todo ello será diseñado al detalle para posteriormente mandar a fabricar y montar.

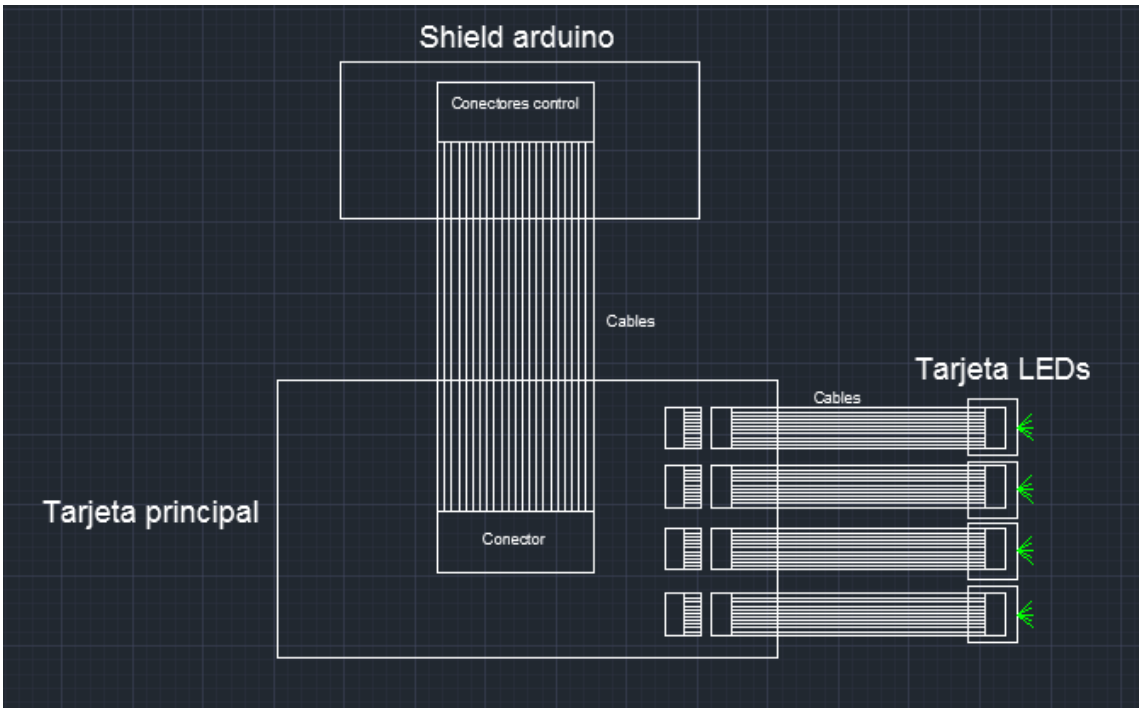


Fig. 5.3 Esquema de las tarjetas electrónicas

Software:

En cuanto a la parte de programación, se realizará con dos lenguajes distintos: *LabView* y *Arduino*.

La primera programación servirá para realizar una *interface* visual para ser manejada por cualquier persona, la cual generará una secuencia que más tarde *Arduino* tendrá que leer e interpretar.

Arduino, en cambio, se encargará de recibir la secuencia generada desde el ordenador (de su interface en *LabView*) y tendrá que leer en partes para poder almacenarla en una tabla y poder seguir la secuencia deseada.

Además, se encargará de ir leyendo los sensores para poder decidir si encender el LED rojo (en caso de que no toque coger algo de esa caja) o verde (en caso de que sea el material que corresponde coger). Resumidamente, *Arduino* será el "cerebro" del sistema, el que tome las decisiones en consecuencia a lo que toque.

Además de esto, contará con el protocolo TCP/IP para mandar la secuencia vía ethernet a nuestro *Arduino*.

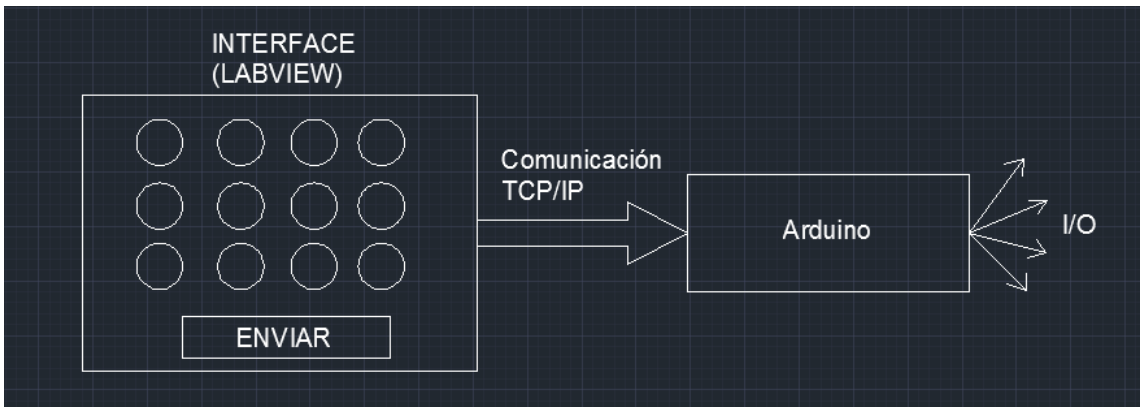


Fig. 5.4 Esquema de la parte software

6. DESARROLLO DEL PROYECTO

A continuación, se explicará el desarrollo del proyecto completo, tanto el software como el hardware, así como la parte mecánica para el acoplamiento de los LEDs, los sensores y las cajas a los puestos de trabajo.

Se dividirá la explicación en 3 partes: hardware, software y montaje. Dentro del apartado de software también hay dos partes: *LabView* y *Arduino*.

Se explicará de forma detallada tanto la programación realizada, así como los circuitos electrónicos diseñados para su posterior implementación en tarjetas electrónicas (también realizadas en el proyecto). Además, la implantación en el puesto de trabajo también se explicará en este apartado.

6.1 Software

Se ha comenzado diseñando la parte software. De esta forma se trató de hacer un aprendizaje autónomo y experimentos para ir consiguiendo los objetivos deseados.

6.1.1 *LabView*

La programación en *Labview* se comenzó por la necesidad de crear una interface visual y sencilla. El programa realizado se fue desarrollando por partes, las cuales se irán explicando en el presente proyecto. Estas partes son 4: comienzo del programa, generar la secuencia a enviar a *Arduino*, transformar la secuencia a un *string* y el envío de la secuencia por comunicación *TCP/IP*.

6.1.1.1 Inicio del programa

Es el comienzo del programa en el cual se trata de la creación de los LEDs que se verán en la interface. A través de la primera parte del programa comienzan apagados los LEDs.

Se han creado los botones que cambian de color (que simbolizan los LEDs del puesto de trabajo).

Para poder usarlos se crean variables locales las cuales hacen referencia al botón que se quiera (esto sirve para poder usar ese botón en otra parte del código).

Fig. 6.1.1.1.1 Botones en LabView para simbolizar los LEDs

Todo el programa va dentro de una secuencia *flat* la cual no pasa de una parte del programa a la siguiente a no ser que se cumpla lo que hay dentro de él.

Fig. 6.1.1.1.2 Secuencia flat de LabView

Como se ha comentado antes, en la primera secuencia del *flat* se inicializan los botones como apagados y se obliga al usuario a poner el LED de mandar la secuencia con nombre ENVIAR a ponerlo de color rojo clicando. En caso de no ser así, LabView muestra en una pantalla pequeña que hay un error y no deja avanzar al usuario para realizar la secuencia.

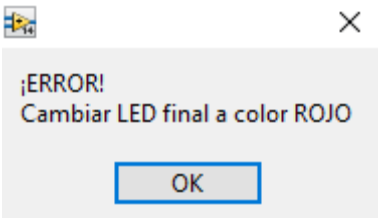


Fig. 6.1.1.1.3 Error que muestra LabView en caso de estar el botón enviar en verde

Fig. 6.1.1.1.4 Botón que envía la secuencia

El error aparece en la pantalla en caso de tener el botón ENVIAR en verde y se dispone de 6 segundos para cambiar el estado ha apagado(rojo) antes de que vuelva a saltar el mismo error.

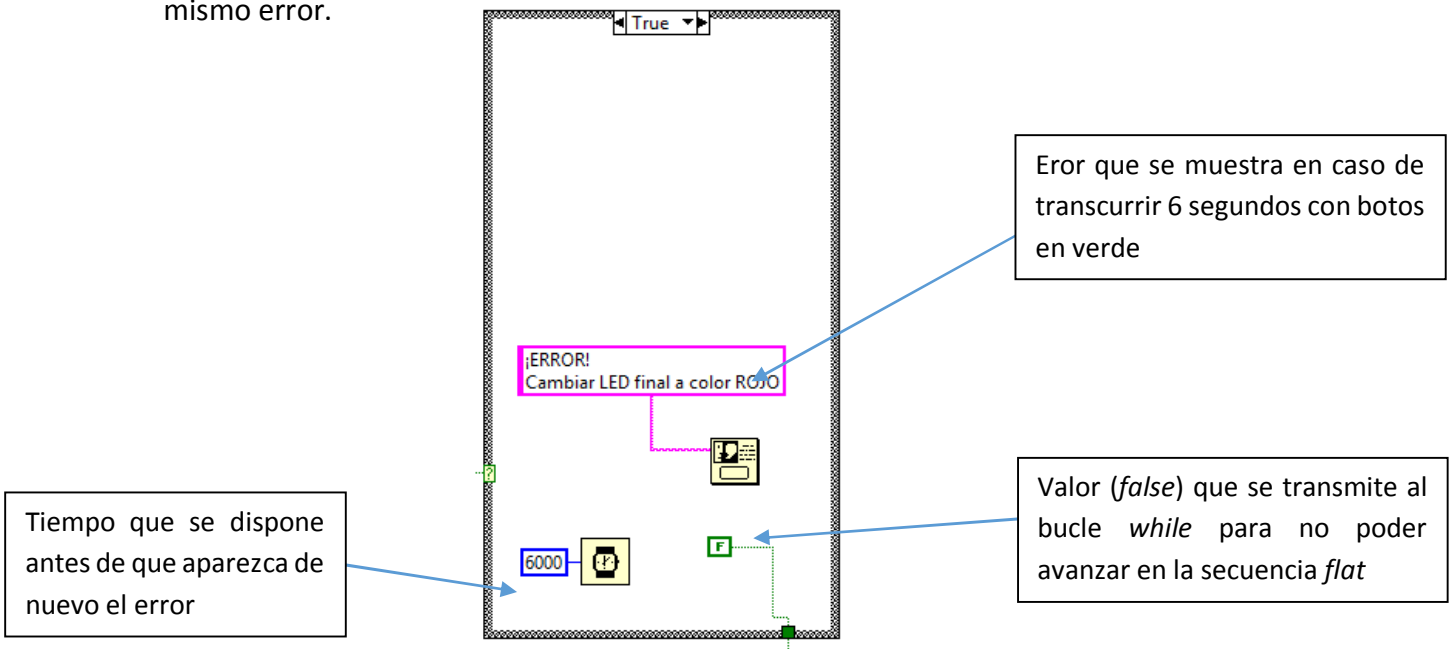


Fig. 6.1.1.1.5 Estructura case de LabView

Las imágenes arriba mostradas corresponden a la parte interface del programa. La parte que se ve a continuación es la parte programada:

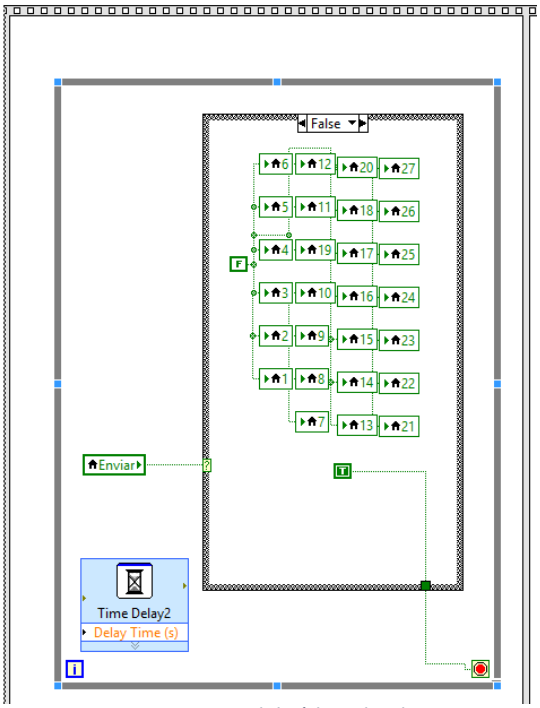


Fig. 6.1.1.1.6 Parte del código donde se apagan los botones.

En esta parte, en caso de que el botón ENVIAR estuviese apagado (en rojo), todos los botones que simbolizan los LEDs estarían apagados y se mandaría un *true* para pasar a la siguiente secuencia del *flat*.

En la parte de la interface esto se visualiza de la siguiente forma (la cual es la que ve el operario).

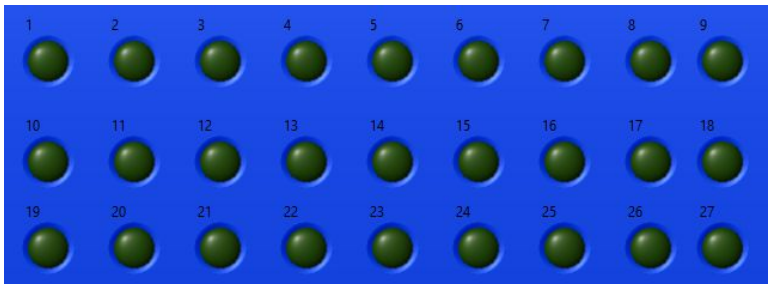


Fig. 6.1.1.1.7 Parte del código donde se apagan los botones.

Fig. 6.1.1.1.8 Parte del código donde se apagan los botones.

Como se puede apreciar en la figura 6.1.1.1.6 todos los LEDs se encuentran apagados. Están numerados desde el 1 hasta el máximo de LEDs que puede haber en el puesto de trabajo que son 27.

El bloque de la primera secuencia *flat* completo sería el siguiente (lo mostrado, pero dentro de un *while).*

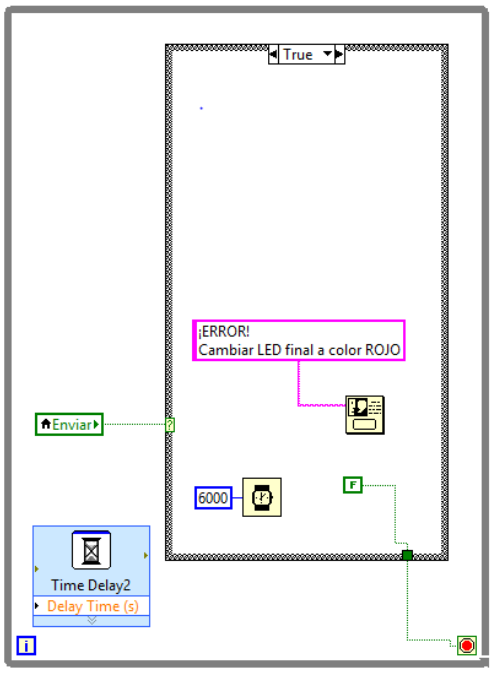


Fig. 6.1.1.1.9 Primera secuencia flat. ENVIAR pulsado (verde)

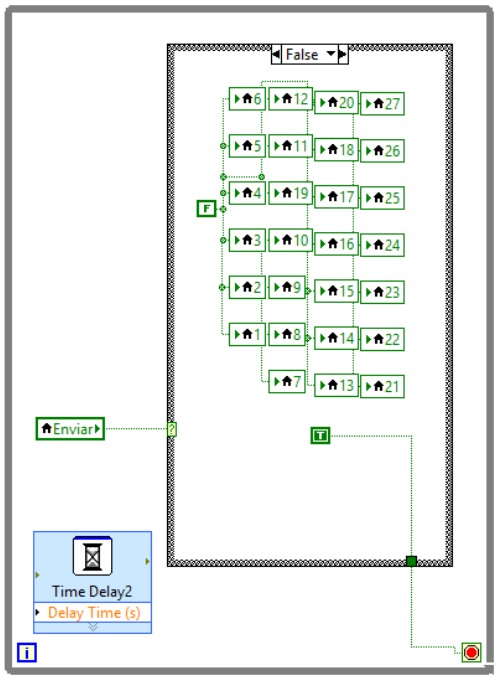


Fig. 6.1.1.1.10 Primera secuencia flat. ENVIAR no pulsado(rojo)

Esta primera parte del programa se ha realizado porque en caso de tener el botón pulsado (en verde) antes de empezar a generar la secuencia, no se enviaba correctamente. Para prevenir el posible error humano como puede ser olvidarse el botón pulsado, se introdujo la parte que indica que hay un error y que hacer para solucionarlo.

Este fue un requerimiento por parte del tutor de la empresa para poder minimizar los errores humanos y no tener que repetir el trabajo perdiendo el tiempo.

6.1.1.2 Generar secuencia a enviar

En este apartado, segunda secuencia del *flat*, se genera la secuencia que se quiere que sigan los LEDs para guiar al operario en la construcción del producto. Para ello el operario tiene que ir seleccionando uno a uno y en orden los LEDs que quiere que se le iluminen posteriormente en el puesto.

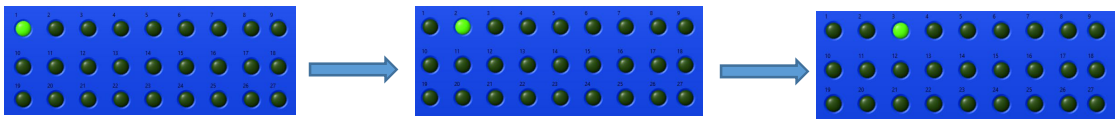


Fig. 6.1.1.2.1 Ejemplo de generar secuencia 1,2,3

Conforme se pulsan los botones en la interface, el programa va generando un *array* donde almacena el número correspondiente (si se pulsa el 1 primero, el primer dato de la tabla será un 1, después el botón 2 se guardará en la posición 2 el número 2...y así sucesivamente).

Dentro de la secuencia de *flat* se encuentra un *while* el cual termina al pulsar el botón ENVIAR (se pone en verde) lo que quiere decir que la secuencia está finalizada y puede pasar a la siguiente parte del *flat*.

A través de la siguiente imagen se mostrará cómo se ha realizado esta parte del programa con LabView.

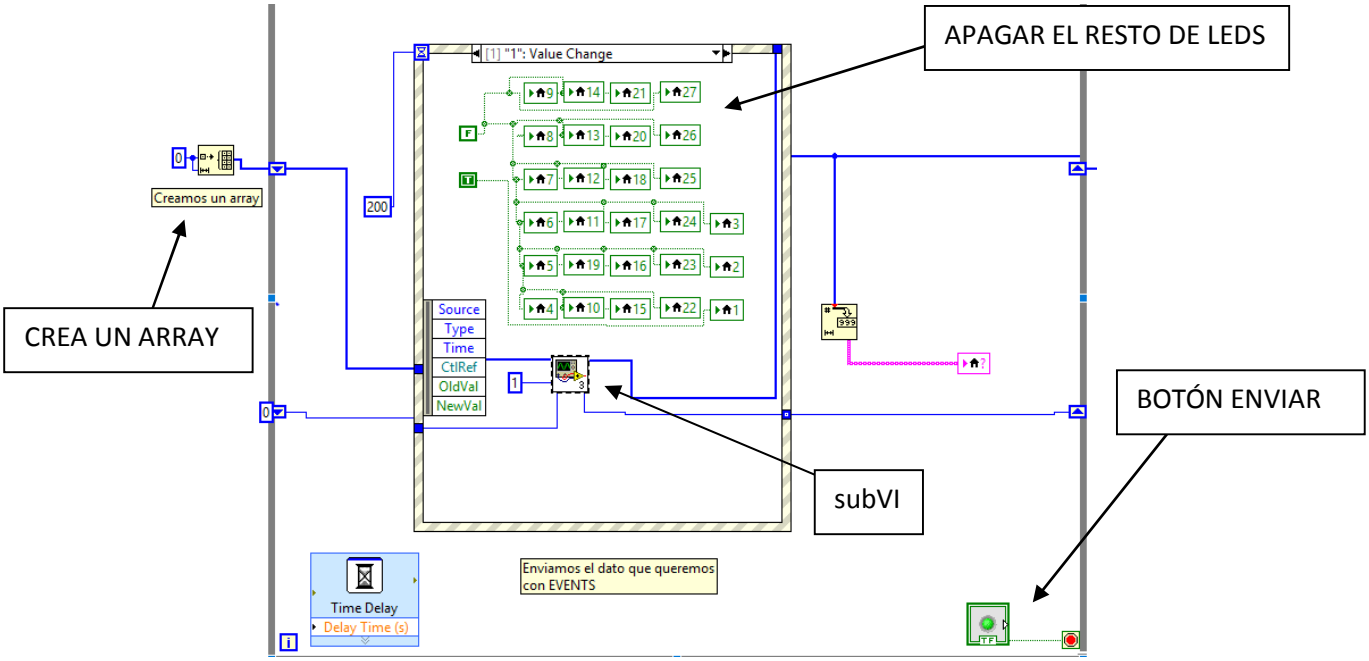


Fig. 6.1.1.2.2 Cómo se genera la secuencia al pulsar los botones

Para la programación de esta parte se ha utilizado la estructura *events* (eventos). Gracias a este tipo de estructura se pueden crear eventos para botones, pulsadores, movimientos de ratón del que maneje el programa…los cuales se pueden conectar a que el programa haga algo que se haya programado.

En este caso se ha utilizado esta estructura para poder generar la secuencia al pulsar los botones. Utilizando este tipo de estructura simplifica inmensamente el programa, ya que en un solo recuadro se pueden introducir todas las funciones que queremos que haga. En caso contrario habría que realizar un programa (a través de un *while* infinito seguramente) para poder saber cuándo se ha pulsado un botón y cuál ha sido y, a partir de ahí, ejecutar lo deseado.

Dentro de la estructura *events* se encuentra un *subVI* el cual contiene dentro de sí un programa que, al tener que usarlo en varios sitios, se ha creado para comodidad del programador. Un *subVI* se comporta como una función, a la cual se la pasan unos parámetros y devuelve otros. En vez de ocupar un gran tamaño dentro del programa, se empaqueta en una caja como la que se muestra a continuación:

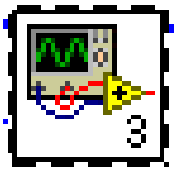


Fig. 6.1.1.2.3 Forma de subVI en LabView

En este caso, dentro del subVI creado se encuentra el *array* que se va generando sumándole el número de la posición del LED (el 1 para la posición 1, el 2 para la 2…).

El programa del subVI es el siguiente:

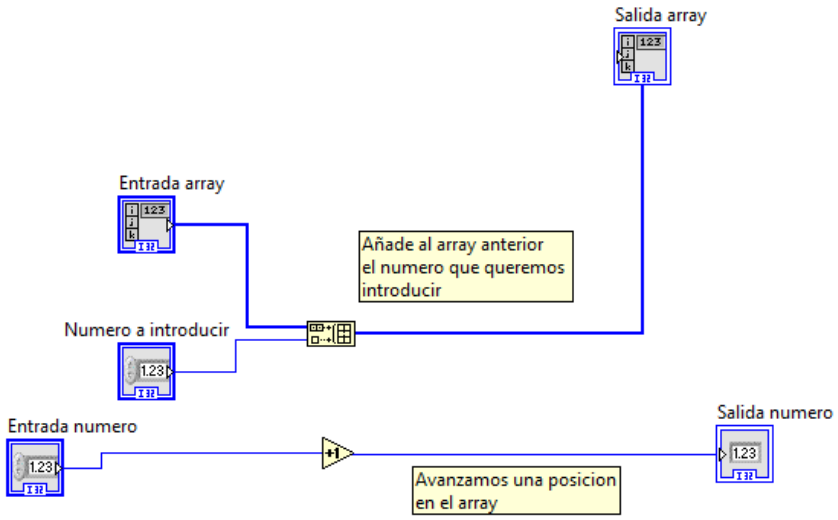


Fig. 6.1.1.2.4 Programa subVI para generar la secuencia en un array

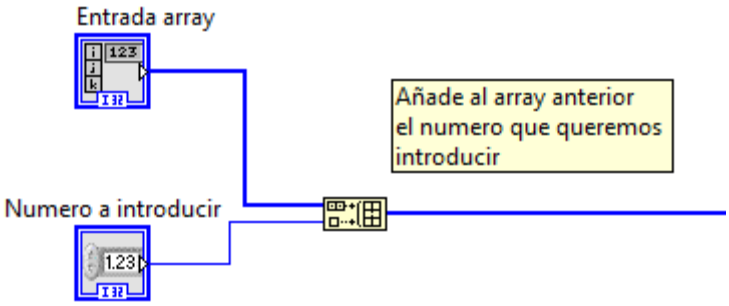


Fig. 6.1.1.2.5 Programa para sumar a un array creado un elemento nuevo

En esta parte del *subVI* viene el *array* y se le suma el número a introducir (1,2,3…) dependiendo la posición. Se utiliza un elemento para sumar al array al anterior.

En caso de que el botón pulsado sea el primero, se introducirá el número 0, si es el segundo se introducirá el, etc… Ese número se sumara a la cola del array que ya está creado.

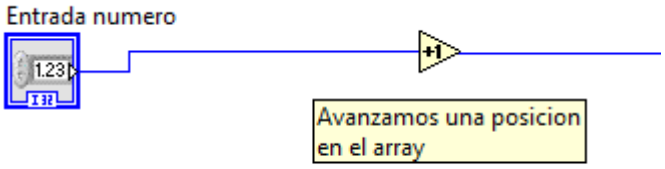


Fig. 6.1.1.2.6 Parte del programa para escribir en la siguiente posición del array y no solapar los datos

Comienza en 0 esta entrada y se le suma de 1 en 1 para poder avanzar por el *array* y que no se solapen los datos unos encima de otros. Gracias a esto primero se escribe en la dirección 1, luego la 2 y así sucesivamente.

Por lo tanto, en esta secuencia del *flag* lo que se trata es de generar la secuencia deseada de una forma sencilla. Todas las opciones de la estructura *events* son iguales ya que la única diferencia entre pulsar un botón u otro es el número que se introduce en el *array*. Con ver una de las opciones (por ejemplo, pulsar el botón 1) basta para conocer como están compuestos el resto de botones.

Lo más importante que se realiza en este apartado es crear el *array* donde va incluida la secuencia. El resto de programación es un asunto de estética que en este tipo de proyectos es un aspecto muy importante ya que tiene que tratarse de algo visual y manejable para cualquier usuario.

Para que sea más visual, se ha hecho que al pulsar un botón se encienda en verde, pero al pulsar el siguiente el anterior se apague y se encienda el que se ha pulsado.

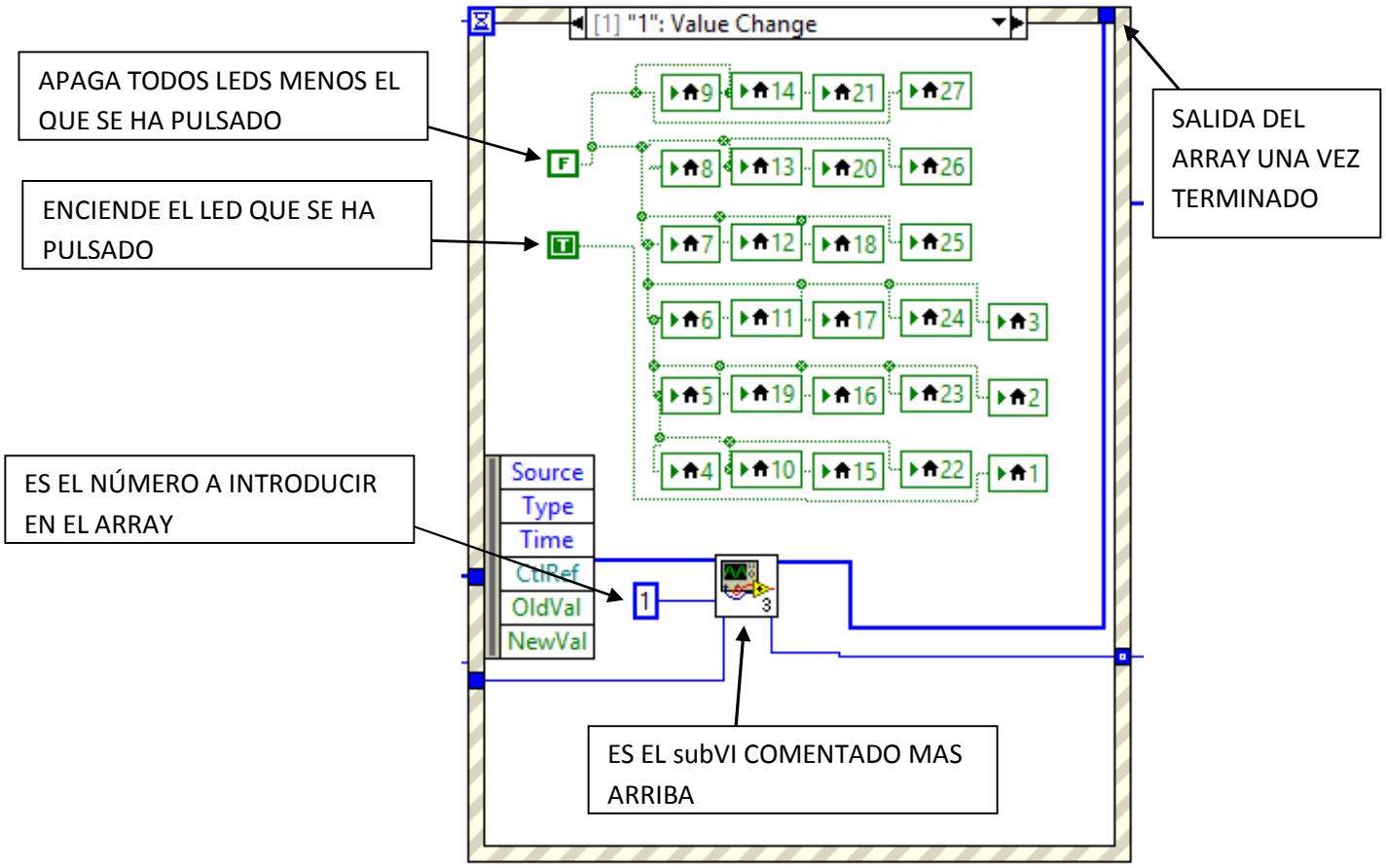


Fig. 6.1.1.2.7 Estructura events para generar la secuencia

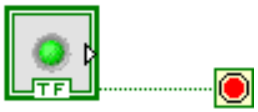


Fig. 6.1.1.2.8 Botón ENVIAR para finalizar el while y mandar la secuencia

Fig. 6.1.1.2.9 Botón ENVIAR en interface sin pulsar

Fig. 6.1.1.2.10 Botón ENVIAR en interface sin pulsar

Junto el programa se encuentra un bloque:

Esta parte hace que se pueda visualizar en el momento que pulsamos el botón el número pulsado.

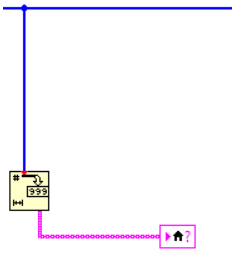


Fig. 6.1.1.2.11 Manda dato para visualizarlo en el momento que se pulsa

6.1.1.3 Convertir secuencia a string

Al realizar la comunicación con *TCP/IP* con *LabView* los datos se mandan en forma de *string*. Esto quiere decir que cada número que se manda es un byte que más adelante *Arduino* tendrá que interpretar uno a uno cada byte y juntarlos.

En este apartado se le da solución a un problema que más adelante surge en la programación de *Arduino*. A la hora de leer los datos, hay que diferenciar de alguna forma cuando el número es de un solo dígito o de dos. La separación entre número y número se realiza mediante ";" para saber cuándo se termina un número. De todas formas, el problema es que *Arduino* lee *string* a *string* (byte a byte) la secuencia y, por lo tanto, no sabe si ese número es un 1 que vale por 1 o por 10. Para dar solución a este problema y simplificar notoriamente la programación en *Arduino,* con *LabView* se ha hecho que los números sean mandados siempre de dos en dos. En caso de que el número a mandar sea mayor a 9, el programa manda el número como es (10, 11, 12, 13…). En caso de que el número sea 9 o más pequeño que este, se le introduce un cero delante del número (09, 08, 07, 06…). Gracias a este método, al leer los *string* que le llegan a *arduino,* este multiplica el primer número por 10 y se lo suma al segundo número. Para indicar que la secuencia ha terminado, se mandan dos "-" que al ser leídos por *arduino* acaba la lectura de datos desde *LabView*.

Si llega el 09→ 10*0 + 9 = 9 si llega 28→ 2*10 + 8 = 28

Este proceso se realiza siempre con todos los números. Así, se evita el tener que memorizar los números anteriores para poder decidir si es de un dígito o de dos. Como inconveniente se podría mencionar que se están mandando más datos de los necesarios, pero en este caso no es algo crítico ya que las secuencias no son muy largas (debido al reparto de tiempos industrial en producción) y la velocidad en la transmisión de los datos no es algo que importe.

Para ello se tiene la tercera parte de la secuencia *flat.* En ella se consigue convertir los números en *strings* y además en caso de ser necesario se le añade un cero para lo comentado anteriormente.

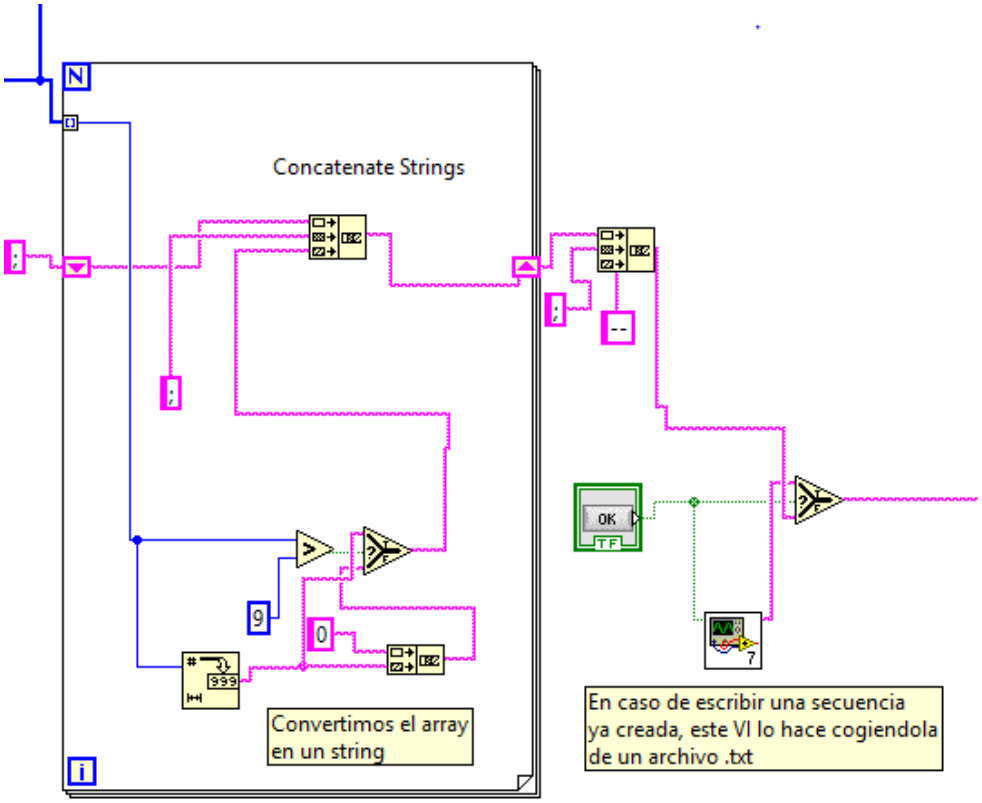


Fig. 6.1.1.3.1 Parte completa de la tercera secuencia flat

Se divide en 3 partes este *flat*: conversión a *string* añadiéndole dígito en caso de ser necesario, concatenar el *string* entero (lo que ya hay con lo que viene) y la salida con sus datos de finalización de secuencia. Todo el programa está, como se puede ver en la imagen anterior, dentro de un *while* el cual va sacando los datos conforme van llegando.

1. Conversión a *string*

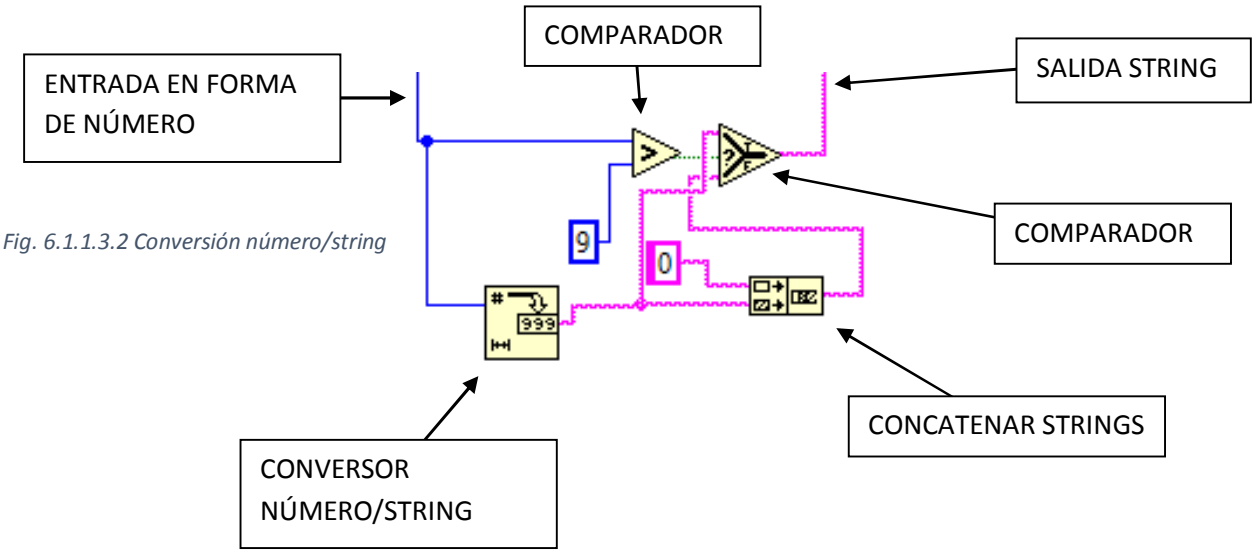


Fig. 6.1.1.3.2 Conversión número/string

La secuencia entra en forma de número y a través del comparador (>) si el número es mayor que 9, se le pasará mediante el conversor número/*string* la parte de la secuencia correspondiente a la salida *string*.

En caso de que sea igual o menor a 9, a la salida *string* se le pasará el número (en formato *string*) pero concatenando un 0 delante.

2. Concatenar *strings*

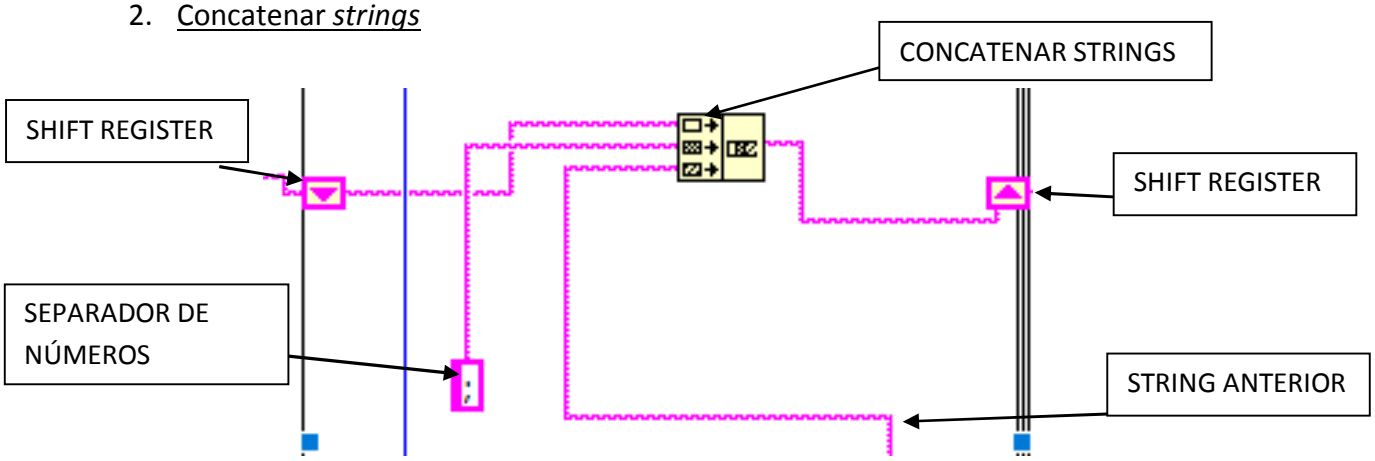


Fig. 6.1.1.3.3 Concatena secuencia de strings

Gracias a los *shift register* se consiguen concatenar el número que toca a los anteriores. Estos registros lo que hacen es en un ciclo del *while* mantienen el valor anterior. Por ello se conectan a un bloque de concatenar, para juntar el anterior valor con el separador ";" y con el nuevo valor de la secuencia.

Anterior valor + ";" + valor nuevo

La salida del bloque de concatenar se introduce de nuevo al *shift register* para poder juntarle el siguiente valor de la secuencia.

3. Salida de los datos

Se saca la secuencia completa en forma de *string* concatenado. Se le añade, como se ha comentado antes, un separador ";" y un símbolo de final de secuencia "-".

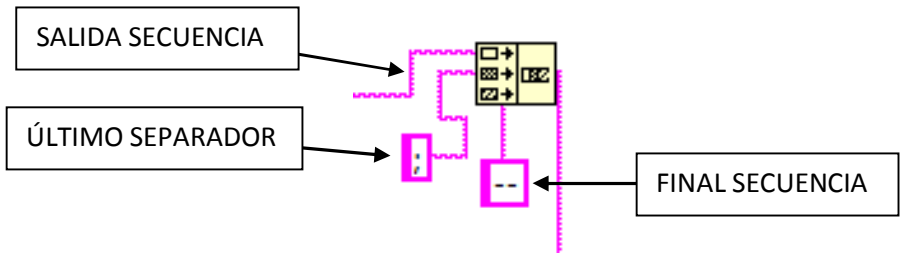


Fig. 6.1.1.3.4 Final de secuencia

Como se puede ver, la secuencia completa llega y se le pone un último separador. Finalmente se pone como final de secuencia otro símbolo para que *arduino* sepa cuando acaba la secuencia entera. Se ponen dos "-" ya que se observó que el último dato de todos no se enviaba correctamente. Así que, al enviar dos "-" el último lo ignora por lo que queda un único separador de final de secuencia.

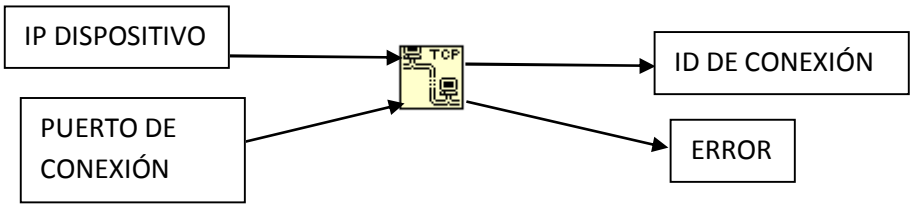
6.1.1.4 Enviar secuencia mediante TCP/IP

Lo único que falta para finalizar el programa de *LabView* es el envío de los datos. Se eligió el protocolo de comunicación TCP/IP ya que es un protocolo muy conocido a nivel industrial (en la empresa donde se han realizado las prácticas y el proyecto se utiliza) y en la asignatura *Comunicaciones Industriales* de la universidad se había realizado alguna práctica con *LabView* y este protocolo.

Existe una gran ventaja para realizar este tipo de comunicación con *LabView* ya que dispone de bloques con los que se puede realizar de una forma muy sencilla y enviar datos.

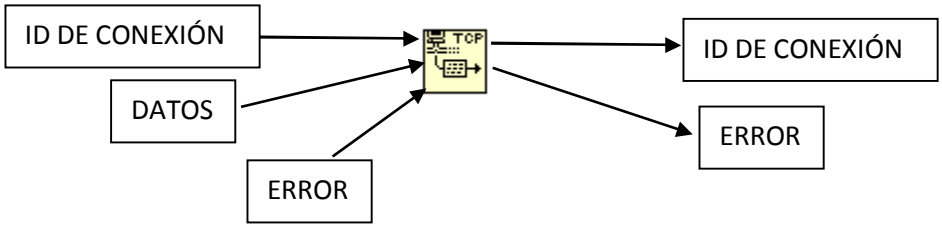
Inicio protocolo

Este bloque abre la comunicación. Tiene varios datos que hay que introducirle, pero con estos 2 es suficiente: IP del dispositivo al que se van a mandar los datos y el puerto por el que se le van a mandar. Como salida tiene el ID de la conexión y el error (en caso de que haya, sino OK).



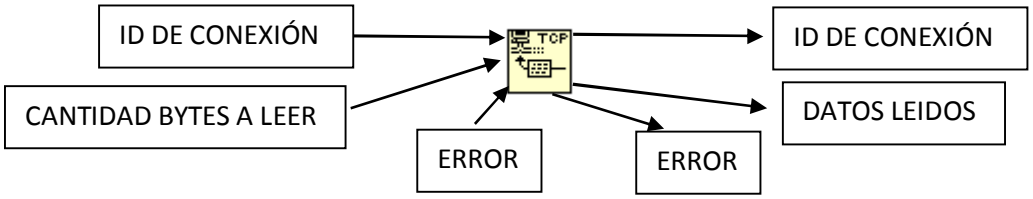
Envío por TCP

Es el bloque que envía la secuencia que se desea. Algo que hay que saber es que los datos se mandan en forma de *string,* por eso se había transformado la secuencia generada de número a *string*. Se introduce a este bloque el ID de conexión, los datos a enviar y el error.



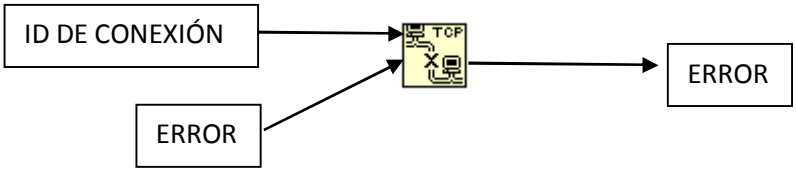
Lectura por TCP

Similar al bloque anterior, sirve para leer los datos que se mandan. En nuestro caso se ha utilizado para la lectura de tiempos de ciclo del sistema *Pick to Light.*



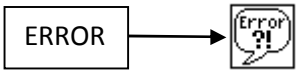
Cerrar comunicación TCP

Una vez que se ha realizado toda la comunicación (enviado y recibido) se cierra la comunicación. En nuestro caso, los bloques de lectura y escritura se encuentran dentro de un bucle *while* y al pulsar un botón este bucle termina y pasa a cerrar la comunicación.



Mostrar error

Por último, disponemos de un bloque el cual indica en caso de que haya algún error de comunicación.



La secuencia *flat* completa sería así:

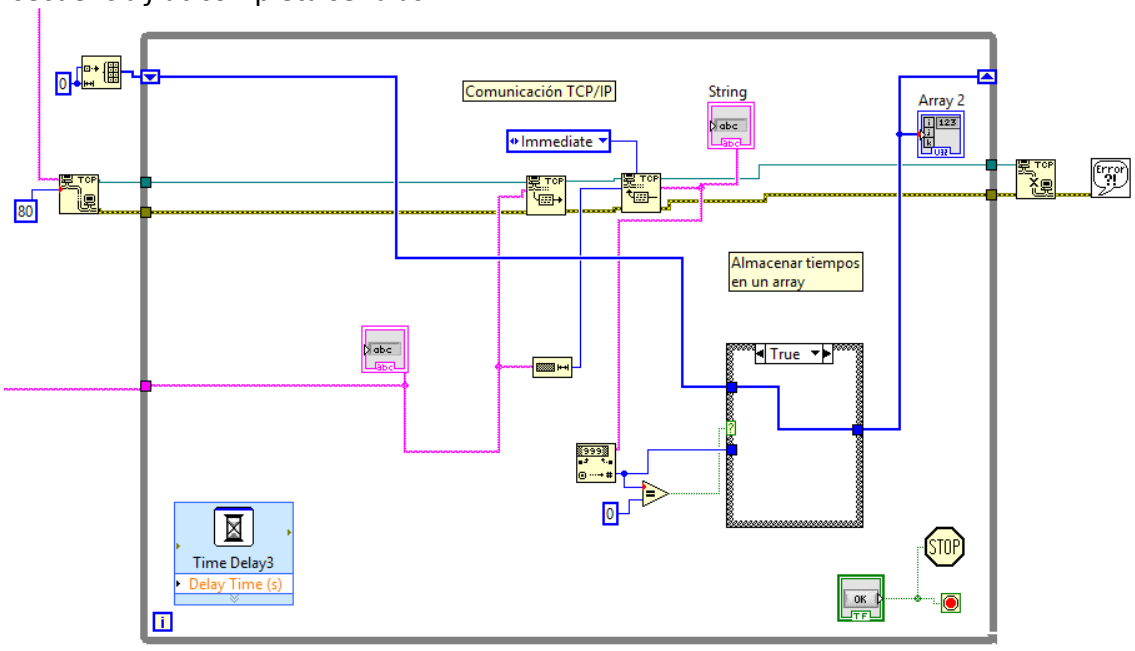


Fig. 6.1.1.4.1 Flat completo envío datos TCP/IP

Se pueden diferenciar dos partes claras en este *flat:* la parte que envía los datos (la secuencia generada) y la parte que lee los datos recibidos desde *arduino* (los tiempos medidos).

- Envío de la secuencia generada

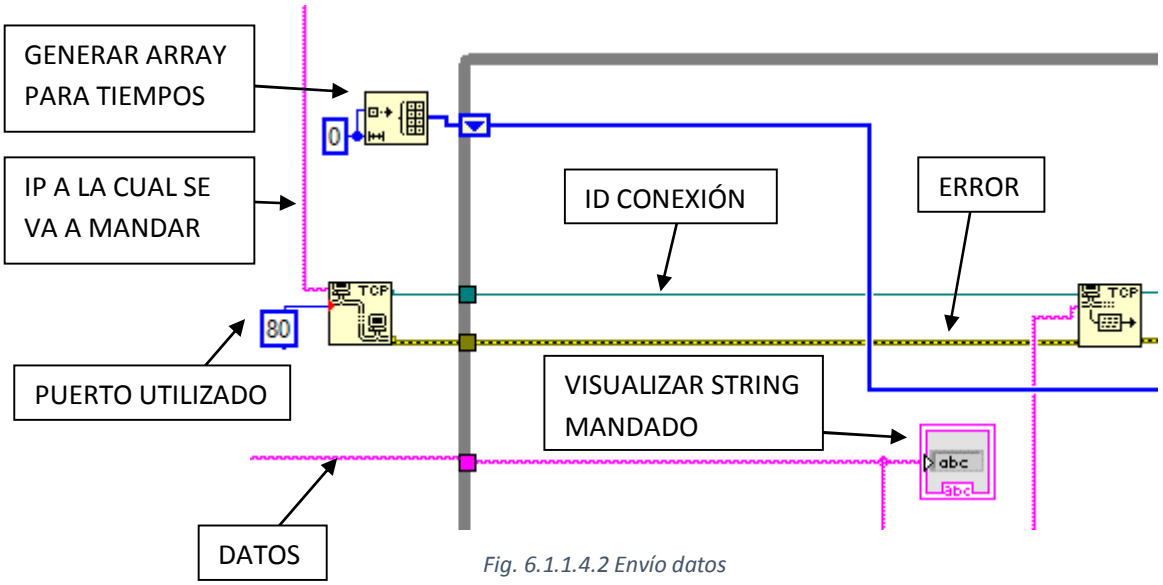


Fig. 6.1.1.4.2 Envío datos

Se introduce la IP del dispositivo al cual se va a enviar la secuencia. Para generar esa IP se ha creado un *subVI*:

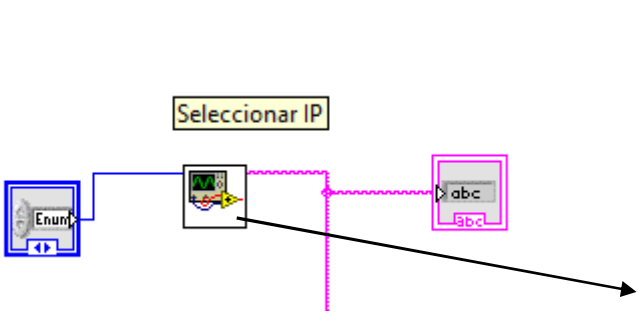


Fig. 6.1.1.4.3 Selección de IP

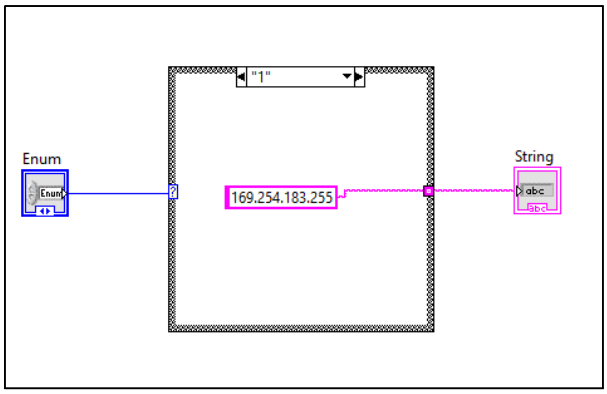


Fig. 6.1.1.4.4 subVI selección IP

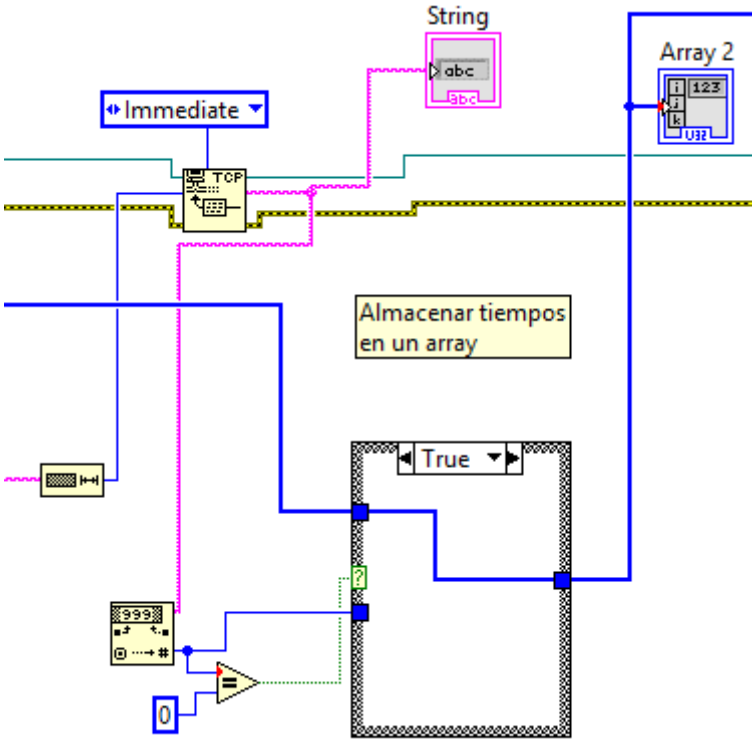
Simplemente se trata de una estructura *case* la cual en caso de qué número de puesto se seleccione entrega una IP diferente.

Como se especifica en la foto, el puerto utilizado ha sido el 80 (simplemente porque se vio que funcionaba).

Por la línea rosa (que significa que es un *string*) se envían los datos que vienen del anterior *flat* ya en formato *string* y se introducen al bloque anteriormente comentado para enviar los datos.

También se ha puesto para poder visualizar la secuencia al enviarse (aunque esto se ha hecho principalmente para saber a la hora de la programación si se estaban mandando correctamente los datos o no).

- Recepción de datos



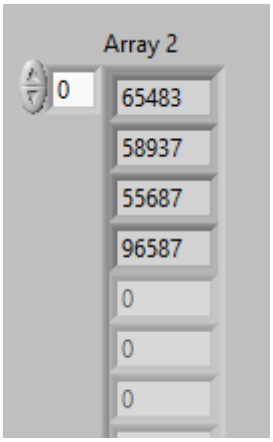


Fig. 6.1.1.4.5 Recepción datos (tiempos)

Este apartado se ha realizado como mejora del sistema *pick to light*. Este sistema consiste en generar una secuencia de encendido de LEDs para guiar a los operarios en el montaje y optimizar el proceso, además de hacer que todos operarios realicen los mismos movimientos.

Aprovechando que los LEDs se encienden conforme el operario coge y monta el producto, se pueden medir los tiempos que le cuesta montar cada parte del producto. De esta manera se obtienen unos datos muy valiosos a nivel industrial y de producción. Hasta el momento este proceso se realizaba mediante grabación con cámara y posteriormente un análisis de los tiempos visualizando el video. De esta forma se obtienen datos sin tener que grabar nada ni observar los videos.

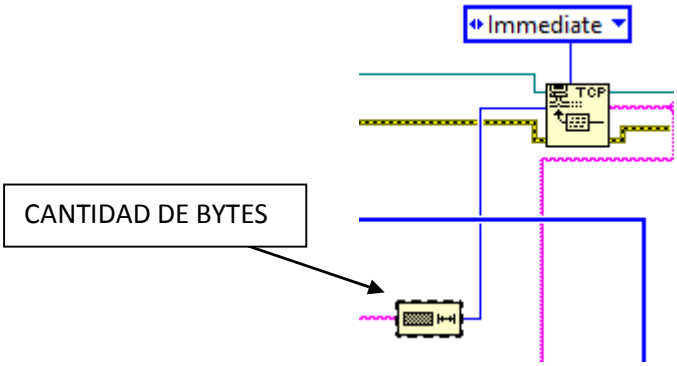
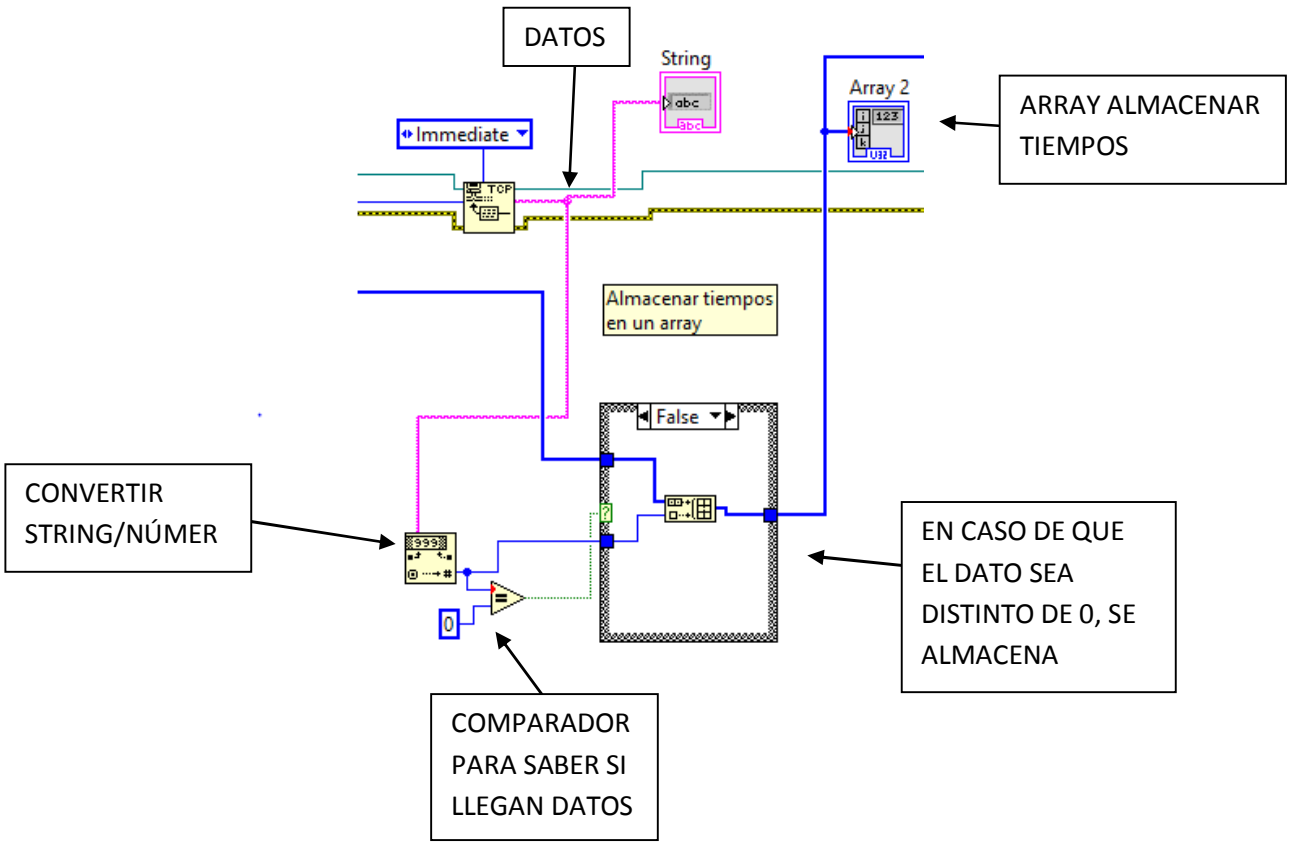


Fig. 6.1.1.4.6 Cantidad de datos a leer

Se manda la cantidad de bytes que se van a leer. Con esto se le dice al bloque receptor cuantos bytes tiene que esperar recibir.



Esta parte de la programación lee recibe los datos desde *arduino* y compara el número que llega con el 0. En caso de llagar 0, se ignora y la tabla de tiempos se mantiene igual. Si recibe algún dato distinto a 0, quiere decir que es una medida correcta y entonces lo almacena en el siguiente valor de la tabla.

La comparación con 0 se hace ya que se vio que *arduino* mandaba ceros y para poder ignorarlos y no tener la tabla llena de ceros.

6.1.1.5 Mejoras para comodidad del usuario

Aunque con todo lo explicado anteriormente era suficiente para el objetivo que se había propuesto, se pensaron algún tipo de mejoras para facilitar la manejabilidad y simplificar procesos de programación del usuario.

Guardar secuencia

El objetivo de esta parte es almacenar de alguna manera la secuencia generada para no tener que volver a repetirla pulsando uno por uno todos los botones. Esto se hace ya que dos puestos distintos pueden llevar la misma secuencia de LEDs y, por tanto, habría que teclear la misma secuencia entera dos veces.

Para dar solución a este problema se ha añadido un botón *Guardar* para almacenar la secuencia en un archivo. La programación que hace este trabajo es la siguiente:

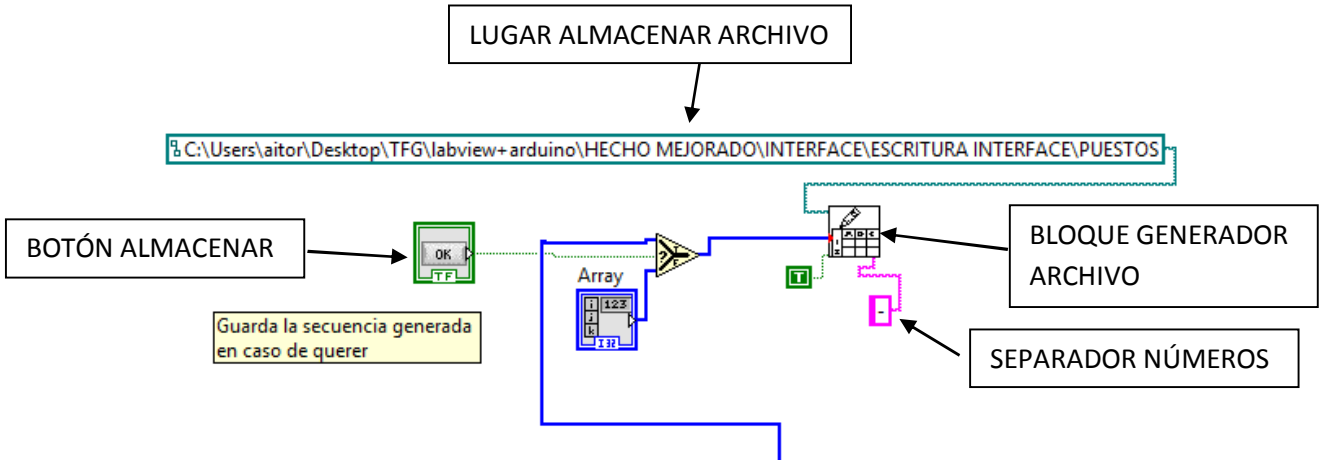


Fig. 6.1.1.5.1 Guardar secuencia generada

La forma de programación es la siguiente: se selecciona el botón *Guardar* (en verde) antes de enviar la secuencia y una vez que se pulsa el botón *Enviar* se genera un archivo donde se haya indicado con la secuencia tecleada y separados los números con el separador que se haya decidido.

Escribir secuencia

Este apartado va en conjunto con el anterior, ya que la idea es poder escribir una secuencia anteriormente guardada para no tener que tecleara de nuevo.

En este caso es algo más complicada la programación ya que hay que separar la secuencia con sus separadores ";" y con carácter de secuencia final "--".

La programación se ha realizado de la siguiente forma:

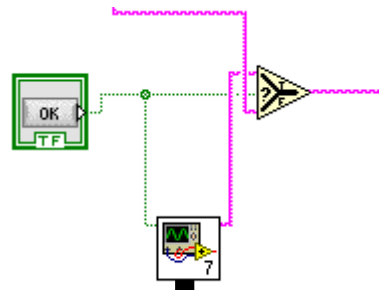


Fig. 6.1.1.5.2 Escribir secuencia guardada

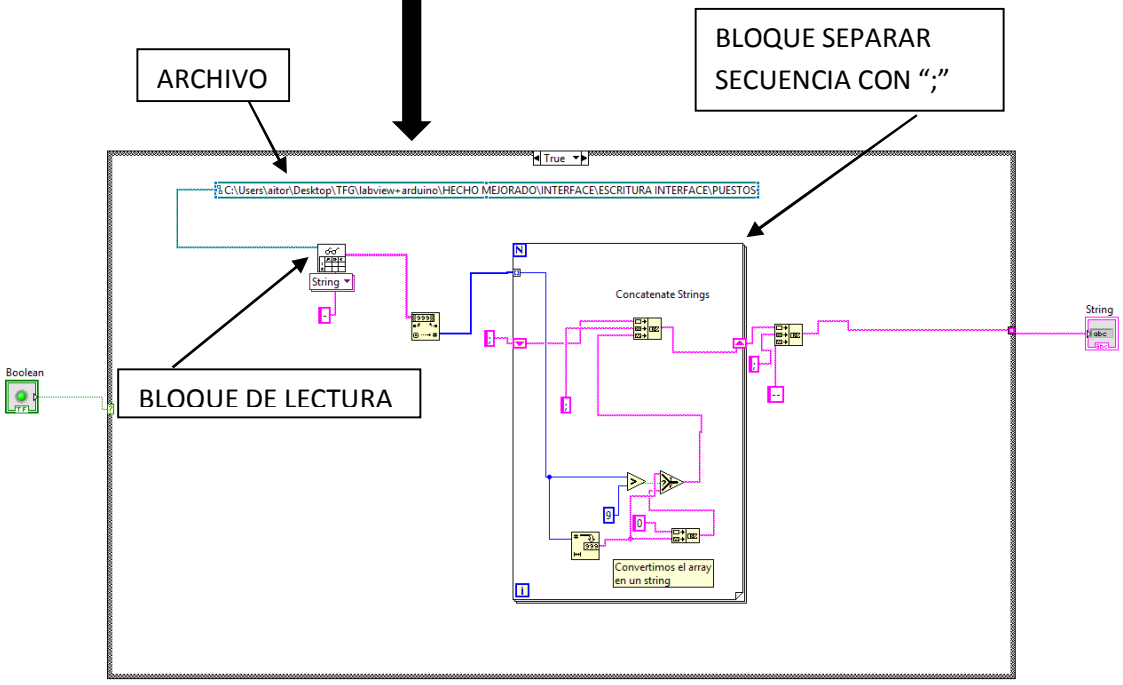


Fig. 6.1.1.5.3 Conversión archivo/secuencia

En la *Fig. 6.1.1.5.2* se puede ver el botón *Escribir* acompañado de un subVI, el cual es el que aparece en la *Fig 6.1.1.5.3.* Dentro de este programa se encuentra la lectura del archivo anteriormente guardado para transformarlo en el formato que lee *arduino* (el cual hemos programado). En caso de que el botón *Escribir* este pulsado (en verde) se lee la secuencia guardada en el archivo de texto y se transforma en la secuencia deseada.

El archivo guardado está almacenado de la siguiente forma:

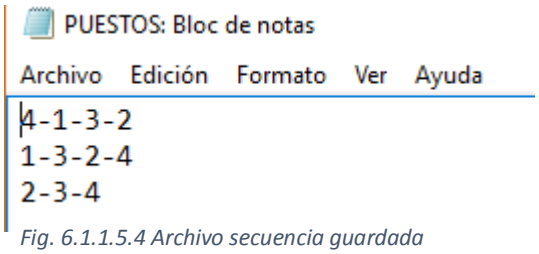


Fig. 6.1.1.5.4 Archivo secuencia guardada

Como se puede apreciar en la imagen, en este caso están almacenadas tres secuencias distintas. El único requisito que hay a la hora de escribir una secuencia anteriormente es que la programación realizada lee la primera línea del archivo, por lo que tendremos que copiar-pegar la secuencia que queramos en la primera línea.

La *interface* final ha quedado de la siguiente manera:

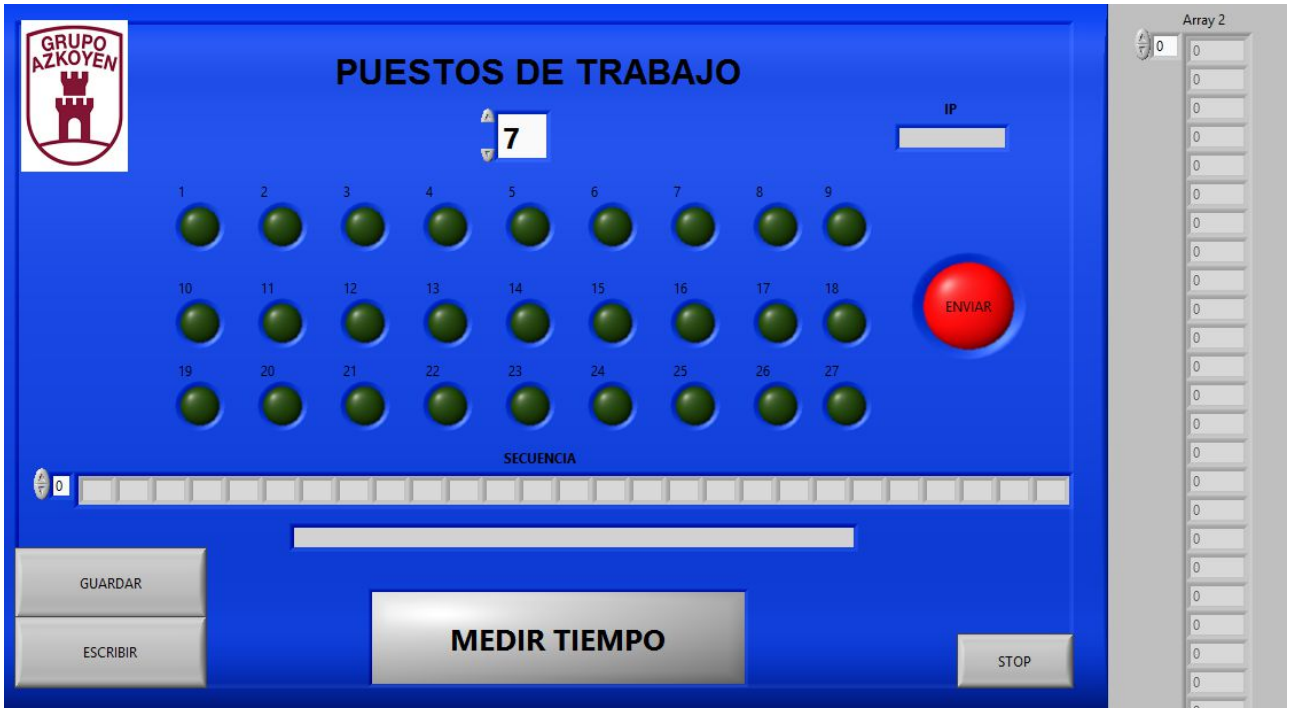


Fig. 6.1.1.5.5 Interface final

6.1.2 *Arduino*

Así como en *LabView* se ha programado para los 27 LEDs que puede haber como máximo en un puesto, *arduino* de momento se ha programado para un prototipo de 8 LEDs (esto se ha hecho para validar el diseño y ver si funciona ya que lo pedía le empresa).

Azkoyen pidió realizar un prototipo de 8 LEDs para poder validar el diseño y poder mandar a fabricar las tarjetas electrónicas. La programación que se explicara a continuación corresponde a este prototipo. De todas formas, el programa final es el mismo ya que en vez de tener 8 veces repetidas la parte de encender o apagar los LEDs, estará 27 veces.

El programa realizado en *arduino* se compone de distintas partes: librerías utilizadas, definición de variables y I/O, comunicación *ethernet*, separación del *string* recibido desde *LabView*, escritura en memoria EEPROM y, finalmente, el programa principal que controla la lectura de los sensores y la activación o no de los LEDs.

6.1.2.1 Librerias utilizadas

Arduino cuenta con mucha gente interesada en este mundo, la cual desarrolla sus propios programas y los publica para cualquier usuario que quiera utilizarlo. A estos se les llama *makers*. Algunos de ellos, a la hora de programar cosas más complejas como

pueden ser tema de comunicaciones, motores, etc…crean lo que se llaman librerías. A través de estas, cualquier persona con menos experiencia en esos temas puede manejarlas sin tener un gran conocimiento.

Por ejemplo, para arrancar un motor paso a paso, habría que excitar las bobinas de una forma determinada para avanzar *x* pasos. Con la librería *Adafruit* se puede controlar un motor paso a paso con instrucciones tan sencillas como *single* o *doublé.*

En este caso se han utilizado tres librerías: *SPI, ETHERNET* y *EEPROM*.

- Librería *SPI*

  Esta librería maneja la comunicación a través del bus *SPI*. Su arquitectura es del tipo maestro/esclavo. El maestro puede iniciar comunicación con uno o varios esclavos y enviar o recibir datos de ellos.

Fig. 6.1.2.1 Comunicación SPI

  La parte difícil de esta comunicación (la programación) está realizada por algún *maker* comentado anteriormente, lo que simplifica su utilidad muchísimo.

  Se puede descargar desde varios sitios de internet y, a continuación, hay que incluirla en el programa.

- Librería *ETHERNET*

  Una de las librerías más importantes utilizadas en este proyecto ha sido la librería *ethernet.* Toda la comunicación entre el ordenador y *arduino* se ha realizado mediante este protocolo de comunicaciones.

  Para empezar, hay que definir la MAC del *shield* ethernet que se tiene. En este caso no venía con ninguna, por lo que se le puso la MAC estándar que se suele

utilizar: *{0xDE, 0xAD, 0xBE, 0xEF , 0xFE, 0xED}.* Tras realizar varias pruebas se vio que funcionaba correctamente.

Para el uso de esta librería y al tratarse de un protocolo de comunicación TCP/IP, es necesario usar una IP. Al estar en prácticas en una empresa, se hizo un ping al ordenador personal para averiguar la IP que tenía.

Abriendo la ventana *cmd* y escribiendo el comando *ipconfig* se averiguó la IP del ordenador personal.

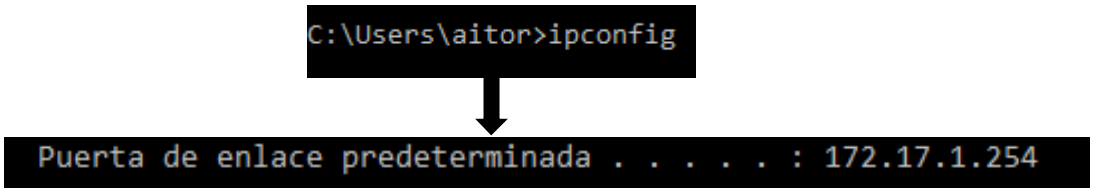


Fig. 6.1.2.2 Método descubrir IP ordenador

Se eligió la IP 169.254.183.255 y para asegurarse de que estaba libre, se realizó otro ping a esta IP para comprobarlo.

```
C:\Users\aitor>ping 169.254.183.255

Haciendo ping a 169.254.183.255 con 32 bytes de datos:
Respuesta desde 172.17.1.26: Host de destino inaccesible.
Respuesta desde 172.17.1.26: Host de destino inaccesible.
Respuesta desde 172.17.1.26: Host de destino inaccesible.
Respuesta desde 172.17.1.26: Host de destino inaccesible.

Estadísticas de ping para 169.254.183.255:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
```

Fig. 6.1.2.3 Ping a la IP libre y su respuesta

En caso de que la IP estuviese ocupada, aparecería este otro mensaje:

```
C:\Users\aitor>ping 172.17.1.254

Haciendo ping a 172.17.1.254 con 32 bytes de datos:
Respuesta desde 172.17.1.254: bytes=32 tiempo=74ms TTL=255
Respuesta desde 172.17.1.254: bytes=32 tiempo=1ms TTL=255
Respuesta desde 172.17.1.254: bytes=32 tiempo=14ms TTL=255
Respuesta desde 172.17.1.254: bytes=32 tiempo=1ms TTL=255

Estadísticas de ping para 172.17.1.254:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 1ms, Máximo = 74ms, Media = 22ms
```

Fig. 6.1.2.4 Ping a la IP ocupada y su respuesta

A continuación, se ha creado el servidor con *EthernetServer()* (crea un servidor que escucha por las conexiones entrantes del puerto definido).

En el *setup* del programa se utiliza la instrucción *Ethernet.begin(mac,ip)* para inicializar la librería y las configuraciones de red.

Ya en el comienzo del bucle *loop* se crea un cliente y se conecta a una IP y un puerto. Esto se realiza mediante *EthernetClient client=server.available().*

Otro elemento de esta librería es *client.connected()* el cual nos dice que hay un cliente conectado o no.

Con *client.available()* nos dice que el cliente está disponible y *client.read()* sirve para leer los datos que mandan.

Finalmente se cierra la comunicación con *client.stop().*

Además de estos elementos existen mucho más, pero para nuestro caso no han sido necesarios, por lo que únicamente se han explicado los utilizados en la programación.

- Librería EEPROM

Esta librería ha sido necesaria para almacenar la secuencia en la memoria y que *arduino* no se "olvidase" de la última secuencia programada una vez desconectada la alimentación.

Utiliza algunos comandos para realizar esto. A la hora de escribir se utiliza el comando *EEPROM.write(dirección, información).* Como puede verse, se escribe dentro del paréntesis la dirección en la cual se va almacenar la información y después la información.

Para leer los datos desde la EEPROM se utiliza el comando *EEPROM.read(dirección).* Directamente leemos el valor que esté almacenado en la dirección que le indicamos. Le damos a una variable el valor de esta dirección. Por ejemplo, variable=*EEPROM.read(dirección).*

La idea de utilizar esto en la programación es ver si ha habido comunicación alguna con el ordenador vía *ethernet*. En caso de que haya habido comunicación, se graba esa secuencia en la memoria EEPROM. En caso de que no haya habido comunicación, lo que esté almacenado en la memoria se pasa a una tabla, la cual administra los encendidos/apagados de los LEDs.

6.1.2.2 Definición de variables y de I/O

En programación es indispensable definir variables para poder manejar distintos datos o activar distintas entradas o salidas.

En este caso hay varias variables definidas debido a la complejidad del programa. Han sido necesarias variables para almacenar los datos leídos desde *LabView,* transformar esos datos de *string* a números enteros, almacenar los datos de la memoria, leer las entradas de los sensores, actuar sobre los LEDs rojos y verdes, etc…

Todas ellas tienen que ser definidas antes de ser utilizadas ya que el programa no las identifica y las señala como errores.

Para el control de los LEDs y los sensores se han definido los siguientes pines como entrada o salida:

- Pines 5, 6 y 7 para el encendido de los LEDs verdes (el pin 4 como *enable*).
- Pines 38, 40, 46 para el encendido de los LEDs rojos (el pin 3 como *enable*).
- Del pin 22 al pin 30 para la lectura de la señal de los sensores.

Estos han sido los pines utilizados para el prototipo. Para el diseño final se han utilizado otros muchos:

- Del pin 22 al pin 49 para la lectura de los sensores.
- Del pin 9 al 13 para el control de los LEDs verdes.
- Del pin 4 al 8 para el control de los LEDs rojos.

Para comentar cada una de las variables definidas a lo largo del programa y no tener que ir una por una aquí, se ha comentado el programa en profundidad en los anexos.

6.1.2.3 Comunicación ethernet

Una de las partes más importantes en este proyecto ha sido el tema comunicaciones. Como antes se ha comentado, *Arduino* cuenta con una librería para manejar este protocolo de comunicación, la cual simplifica la programación.

Se ha creado una variable en la cual se irá acumulando la secuencia que va llegando, concatenando los bytes que llegan. Esta variable se llamará *bufferString.* Lo primero se creará un cliente que se conectará a la IP y puerto definidos anteriormente.

Se esperará 10 segundos para esperar si hay comunicación alguna y en caso de que la haya se comenzarán a leer los datos y almacenarlos en la variable anteriormente mencionada. Además, en una variable con nombre *numDatos* se almacenarán la cantidad de datos recibidos para posteriormente disponer de esta información y utilizarla.

Una vez que no hay más datos para leer, el cliente se desconecta y el programa continua.

```
void loop() {
  delay(10000);                                  //esperar 10sg a ver si hay comunicacion o no
Serial.println("fin delay");

EthernetClient client=server.available();       //crea un cliente que se conecta a una IP y puerto
if(client) {                                    //indica si el cliente esta preparado

  bufferString="";                              //limpiamos el buffer para leer la secuencia
  while(client.connected()){                    //mientras el ciente esté conectado

    if(client.available()>0){                   //si el numero de bytes a leer es mayor a 0 (hay algo para leer)

      numDatos=client.available();              //cantidad de datos que hay para leer
      delay(20);                                //esperamos 20 milisegundos

      while(client.available()>0){              //mientras haya algo para leer

        bufferString += (char) client.read();   //leemos y se lo sumamos a lo leido anteriormente
      }



  }


}
  client.stop();                                //desconecta el cliente del servidor
```

Fig. 6.1.2.5 Programa Arduino para comunicación ethernet

6.1.2.4 Separar secuencia

Para realizar este proceso primero se ha transformado el *string* en *char* para posteriormente poder transformarlo en número entero de una forma más fácil.

Para transformar de *string* a *char* existe una función que la transforma.

string.toCharArray(Char,cantidadDatos)

Para poder ir uno por uno pasando por cada dato se ha utilizado un puntero. Gracias a este elemento se puede ir uno a uno por cada uno de los elementos.

Una vez que se tiene la secuencia en formato *char*, se almacena en una tabla utilizando un bucle *for* y el puntero anteriormente mencionado.

```
char *ptr=stringToChar;                          //puntero para recorrer la variable y poder separar los caracteres

for (int i=0; i<numDatos;i++){

   tabla[i]=(*ptr++);                            //separa cada numero en cada parte del array

}
```

Fig. 6.1.2.6 Programa para almacenar en una tabla la secuencia(char)

Una vez que se tiene en una tabla guardados todos los elementos de la secuencia, se pasará cada número *char* a un número entero. Para esto no existe ninguna función la cual pueda ser de ayuda, por lo que se hace mediante un bucle *for* nuevamente.

Los *char* que se tienen almacenados están en código ASCII. Este código está basado en el alfabeto latino y utiliza 8 bits (1 byte) con el cual se puede representar símbolos, letras y números.

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Fig. 6.1.2.7 Tabla del código ASCII

Como se puede observar, el número en ASCII “48” es el número entero “0”. Se puede concluir que si el número que se dispone se le resta “48” se obtiene el número deseado en entero.

Char-48=Número entero

O

Char-‘0’=Número entero

Cualquiera de estas dos formas vale ya que son iguales.

```
for (int i=0;i<numDatos;i++){

    tablaInt[i]=tabla[i]-'0';                    //convertimos los numeros char a enteros

}
```

Fig. 6.1.2.8 Transformar char en número entero

Una vez que ya se tienen los datos de la forma deseada, se procede a juntar los números. Esto se hace ya que de momento se tienen todos los números separados, por ejemplo, el número 15 está en dos partes separado (el 1 y el 5) o el número 7 está en otras dos partes separado (el 0 y el 7).

Ahora corresponde juntar los números y almacenarlos en una tabla todos (07,15,27, etc…). Para ello se hace utilizando un método con los siguientes pasos:

1. Bucle *for* que empieza en el comienzo de la secuencia y termina cuando la cantidad de bytes almacenados acaba.

2. Los números que están en la posición par (0,2,4,6, etc…) se multiplican por 10 siempre. Los números que están en la posición impar (1,3,5,7, etc…) se dejan sin multiplicarlos por nada.

3. Se suman el de la posición par con el de la impar (el del 0 con el del 1, el del 2 con el del 3, etc…) y se almacenan en una posición cada una de las sumas.

4. Cuando el símbolo es “;” es que un número ya ha terminado, por lo que empieza el siguiente número. A esto se le llamara un separador.

5. Por último, cuando el símbolo sea un “-“ querrá decir que la secuencia ha terminado, por lo que el bucle *for* acabará.

```
for (int i=0;i<numDatos;i++){                    //for para juntar los numeros (si son 11,12 etc hay que juntar el 1 con el 2 por ejemplo)

  if (tabla[i]==';'){

    NULL;                                        //si hay ; no hacemos nada

  }

  else if (tabla[i]=='-'){

    tablaSuma[j]=0;                             //se almacena un 0 para saber que se acaba la secuencia
    parar=j;                                    //parar=j para saber la cantidad de datos que hay en la tabla
    j++;

  }

  else if (contador==0){

    suma=tablaInt[i]*10;                        //si es la primera parte del numero multiplicamos por 10
    contador=1;                                 //el numero sera= parte1*10 + parte2


  }

  else if (contador==1) {

    suma=suma+tablaInt[i];                      //si es la segunda parte sumamos el numero y la guardamos en tablaSuma
    tablaSuma[j]=suma;
    j++;
    contador=0;
    numSeparados++;                             //variable para saber cuantos numero hay


  }


}
```

Fig. 6.1.2.9 Programación separación de la secuencia

6.1.2.5 Grabar secuencia en EEPROM

Uno de los problemas que tiene *Arduino* es que una vez desconectada la fuente de alimentación la secuencia recibida se borra, por lo que no existe ninguna secuencia. Para ello se ha decidido grabarla en la EEPROM de *Arduino* para que, una vez desconectada la alimentación y de nuevo conectada, se pueda leer la secuencia que se había generado.

Dentro de grabar la secuencia en la EEPROM puede haber dos opciones diferentes: una en la que haya habido comunicación vía TCP/IP con el ordenador y, por lo tanto, se haya recibido una secuencia nueva, y otra en la que no haya habido ningún tipo de comunicación, por lo que no hay ninguna secuencia nueva.

En la primera opción, es necesario guardar esa secuencia en la EEPROM para más adelante poder utilizarla. En la segunda, en cambio, no hay que grabar nada en la memoria EEPROM, sino que hay que leer lo que hay escrito en ella para poder pasarla a una tabla y utilizarla.

A la hora de grabar la secuencia, como se ha comentado en el apartado de librerías (más concretamente en la librería EEPROM), se utilizan los comandos ***EEPROM.write(direccion,cantidad de datos)*** y ***EEPROM.read(dirección a leer).***

Si ha habido comunicación con el ordenador, entre en un bucle *for* para escribir la secuencia en la memoria EEPROM y va avanzando por la memoria para almacenar todos los datos.

```
if (noVolatil==1){                                  //si ha habido comunicacion con el ordenador
  noVolatil=0;                                      //cambiamos vble. para no tener que hacer esto otra vez

  EEPROM.write(direccion,numSeparados);             //se guarda en la direccion la cantidad de numeros que hay
  for (j=0;j<(numSeparados);j++){
    EEPROM.write(direccion+1+j,tablaSuma[j]);      //se almacenan los numeros a partir de la anterior direccion
  }
}
```

Fig. 6.1.2.10 Grabado de la secuencia en la memoria EEPROM

En caso de no haber comunicación alguna entre ordenador/*Arduino*, se tiene que leer lo que se grabó por última vez en la memoria EEPROM y pasarla a una tabla, la cual se utilizará para encender o apagar los LEDs. Esto se hace con un bucle *for* el cual hace que se vaya leyendo uno a uno todos los datos almacenados.

```
if (disponible==0){                          //si no ha habido comunicación con el ordenador hay que leer desde la EEPROM la anterior secuencia guardada

 for (int i=0;i<numSeparadosVolatil;i++){
   val1=EEPROM.read(direccion+i+1);           //se lee lo que hay en la EEPROM
   tablaSuma[i]=val1;                         //y se pone en a tablaSuma
   Serial.println(tablaSuma[i]);
 }
 disponible=1;                               // se cambia la vble. para que no vuelva a entrar aqui
 parar=numSeparadosVolatil;                  //almacena la cantidad de numeros que tiene la secuencia
}
```

Fig. 6.1.2.11 Leer la secuencia almacenada en EEPROM e introducirla en una tabla

6.1.2.6 Programa principal

Todo lo explicado anteriormente se trata de la comunicación y el procesado de los datos recibidos. El programa que controla las tarjetas electrónicas (*hardware*) es el programa principal.

Esta parte del programa está dividida en diversas partes: si corresponde al primer número de la secuencia o el sensor ha detectado presencia, encendido de LED verde, encendido de LED rojo y reinicio de secuencia.

Constantemente se está leyendo el estado de los sensores para saber si se ha detectado o no la presencia del operario.

En este apartado no se adjuntará el programa, sino que se adjuntará un diagrama de bloques para entender cómo funciona el programa principal. En los anexos ya está adjunto el programa completo con sus comentarios correspondientes.

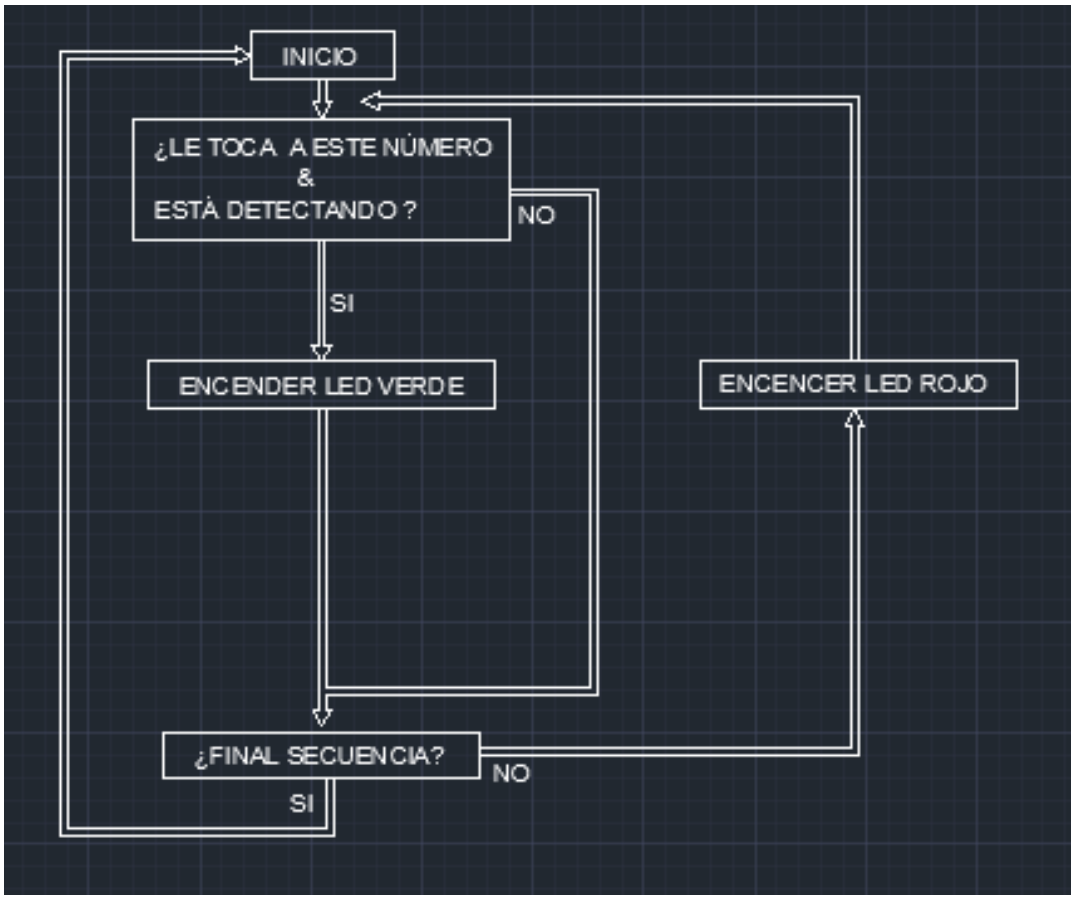


Fig. 6.1.2.12 Diagrama de bloques del programa principal

6.2 Hardware

El segundo apartado del diseño del proyecto final de carrera es el apartado *hardware*. Esta parte es la que controla los encendidos y apagados de los LEDs, así como el acondicionamiento de los sensores. *Arduino* manda y recibe las señales a las tarjetas electrónicas para que estas hagan su cometido.

Se dividen 3 tarjetas electrónicas distintas:

6.2.1 Tarjeta *shield*

Es la tarjeta que se incluye encima del *Arduino* que controlará el puesto de trabajo. Esta tarjeta ha sido diseñada ya que extraer los pines de *Arduino* con cables era algo poco robusto, por lo que se decidió diseñar una tarjeta en forma de *shield* para poder extraer los pines de una forma segura y cómoda.

Lo primero fue decidir qué pines iban a ser utilizados para la escritura/lectura de los sensores y los LEDs.

Los pines elegidos fueron los siguientes:

- Del pin 22 al pin 49 para la lectura de los sensores.
- Del pin 9 al 13 para el control de los LEDs verdes.
- Del pin 4 al 8 para el control de los LEDs rojos.

Esta tarjeta se ha diseñado con el programa *fritzing* ya que cuenta con prototipos de tarjetas de *Arduino* y esto simplifica considerablemente el diseño.

La tarjeta se ha diseñado a dos capas para simplificar el diseño y porque el precio de la tarjeta no variaba mucho respecto a una capa.

Las dimensiones de la tarjeta son 110.3x77.9 mm. La tarjeta ha sido diseñada con estas dimensiones para poder enrutar las pistas de una forma simple y para que el *shield* ocupe todo el *Arduino*.

El diseño de la tarjeta ha quedado de la siguiente manera:

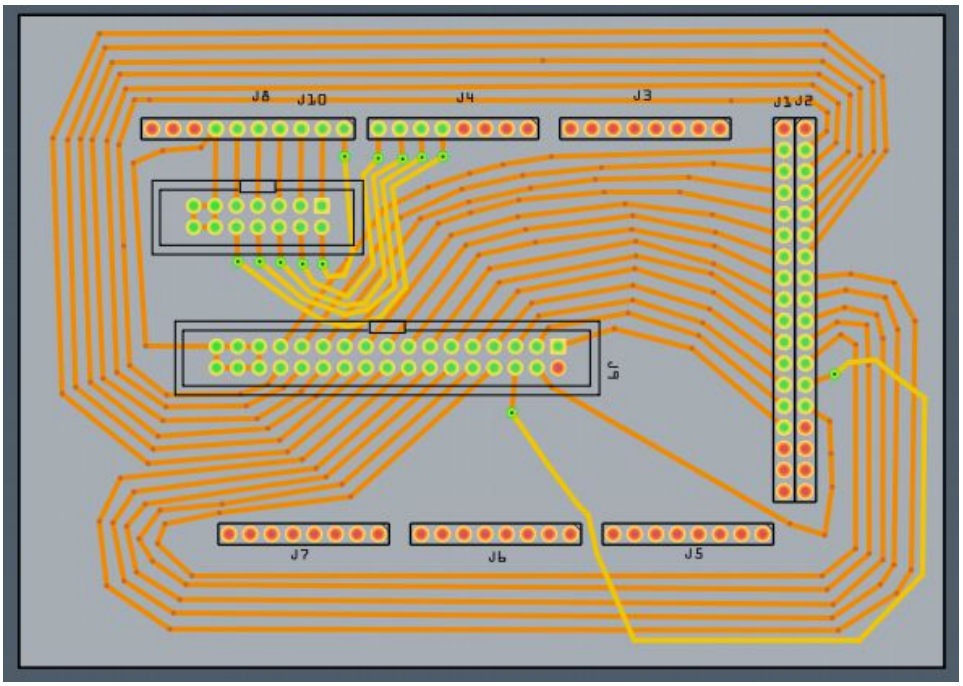


Fig. 6.2.1.1 Diseño tarjeta shield en fritzing

Al extraer los archivos *Gerber* (los cuales son los necesarios para mandar a fabricar la tarjeta electrónica) se visualiza mediante un simulador *online* para ver cómo quedaría la tarjeta.

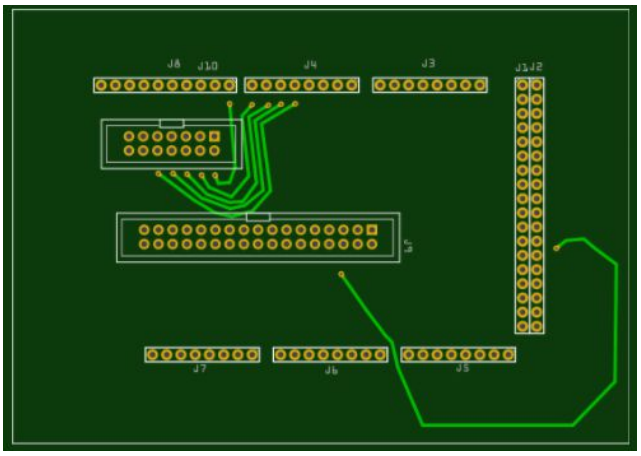


Fig. 6.2.1.2 Vista top tarjeta shield

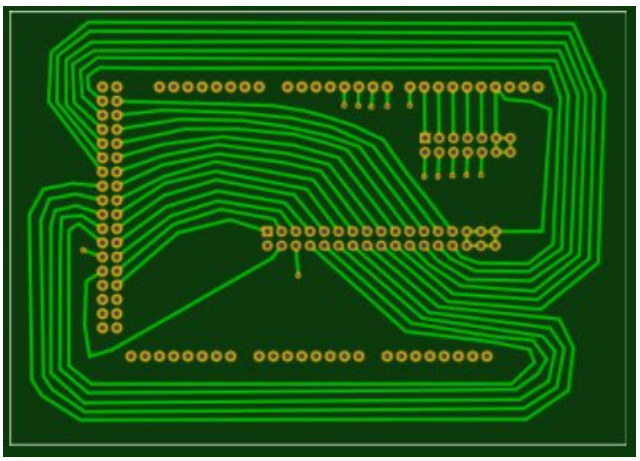


Fig. 6.2.1.3 Vista bottom tarjeta shield

La tarjeta también ha sido montada a mano para el prototipo, aunque con menos conectores y para menos LEDs debido a la complejidad de soldar tantos cables. El prototipo ha sido de 8 LEDs y 8 sensores.

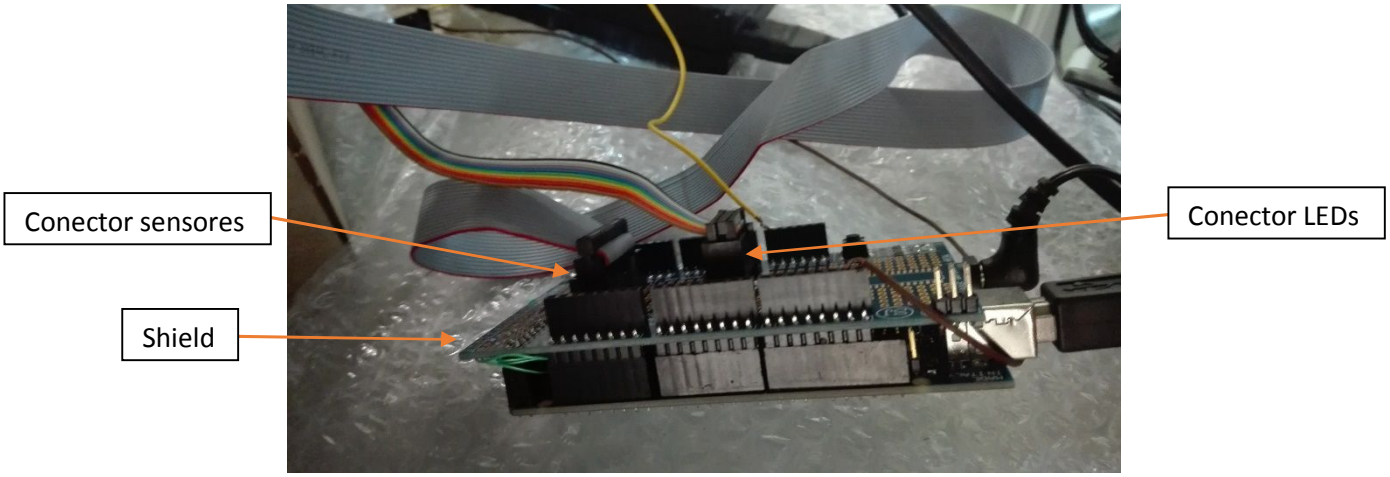


Fig. 6.2.1.4 Tarjeta shield prototipo

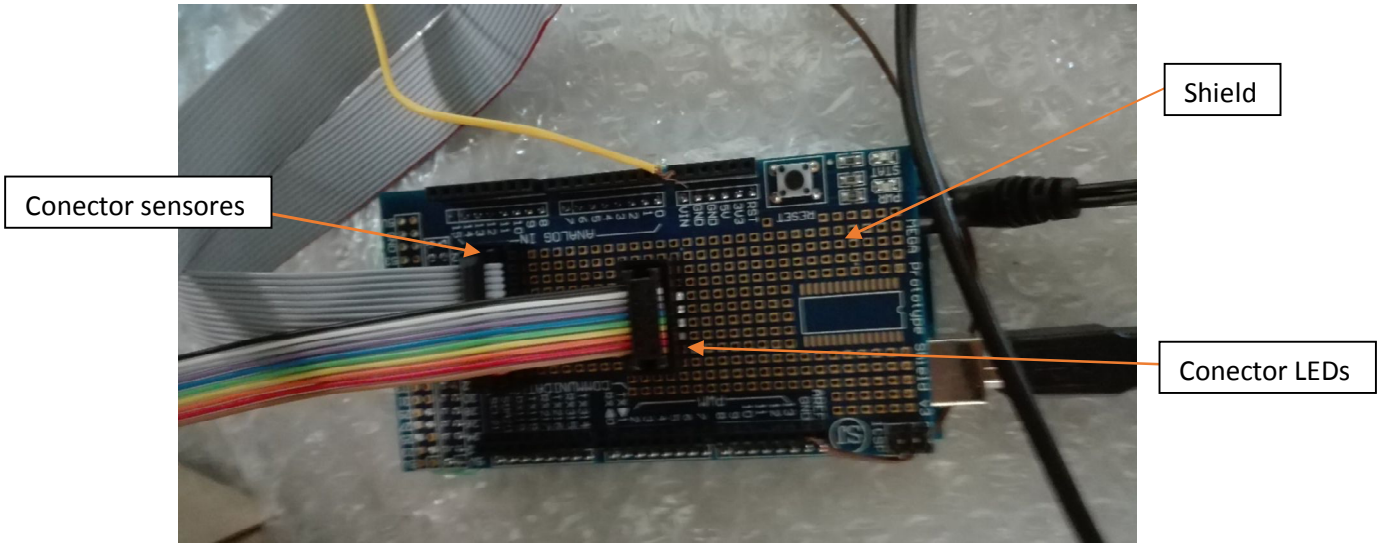


Fig. 6.2.1.5 Tarjeta shield prototipo

Tras verificar el comportamiento del prototipo y ver que las señales son correctas, se procederá a mandarla a fabricar.

6.2.2 Tarjeta principal

Esta tarjeta es la que se encarga de controlar los encendidos/apagados de los LEDs, de acondicionar las señales de los sensores, donde se introducirá las tensiones de alimentación, etc…

En ella se incluyen varios demultiplexores, resistencias, conectores, comparadores, etc…

Se adjuntarán fotos sobre el diseño eléctrico del prototipo ya que el diseño final está en proceso de ser acabado y revisado junto al responsable de la empresa.

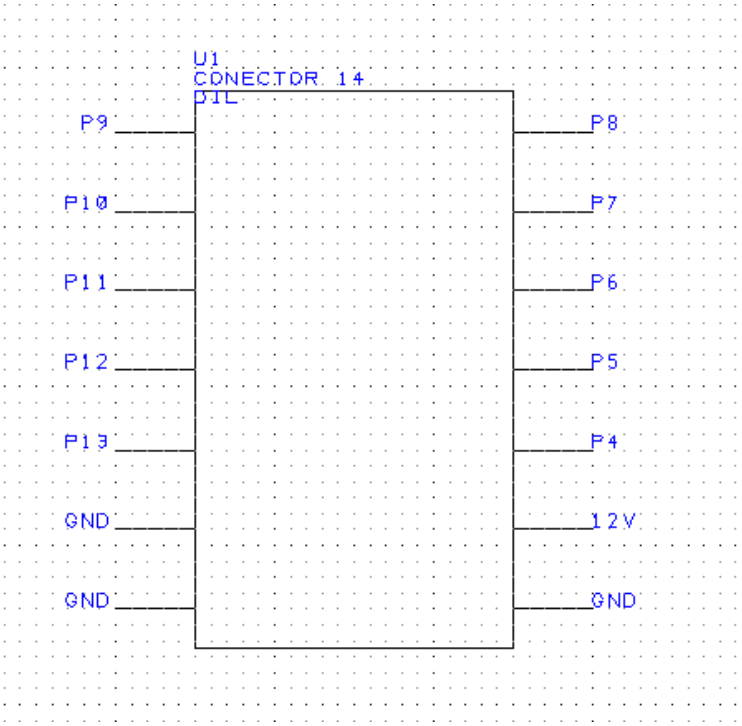


Fig. 6.2.2.1 Conector 10 vías control arduino

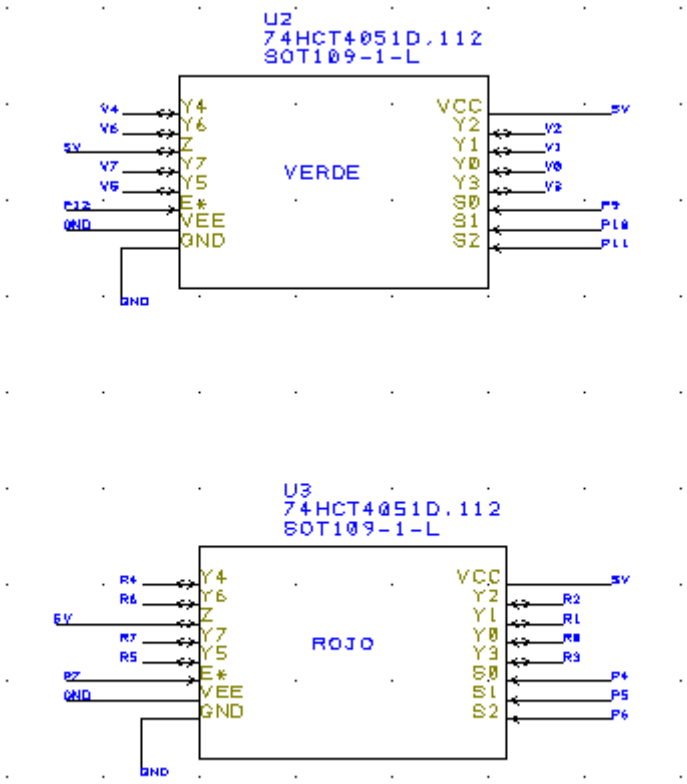


Fig. 6.2.2.2 Multiplexores para control LED rojo y verde

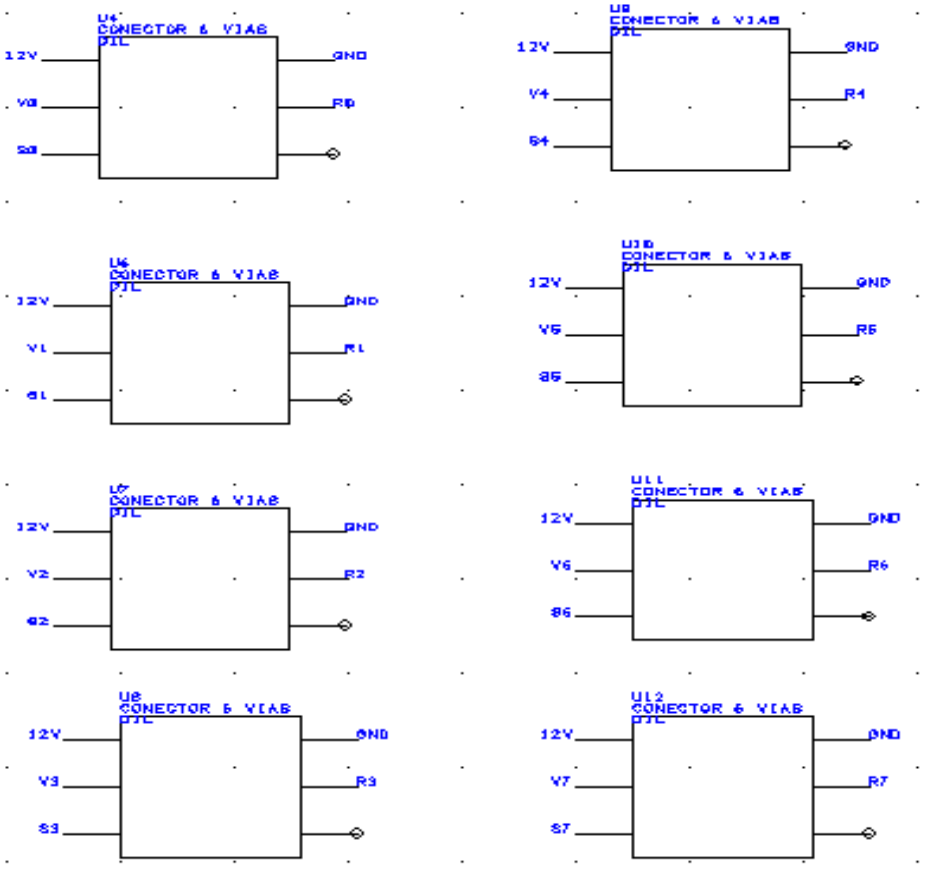


Fig. 6.2.2.3 Conectores a cada tarjeta de cada caja

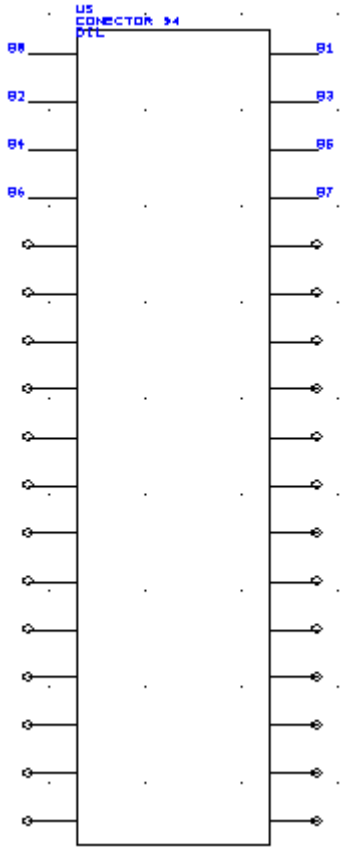


Fig. 6.2.2.4 Conector de señales de sensores

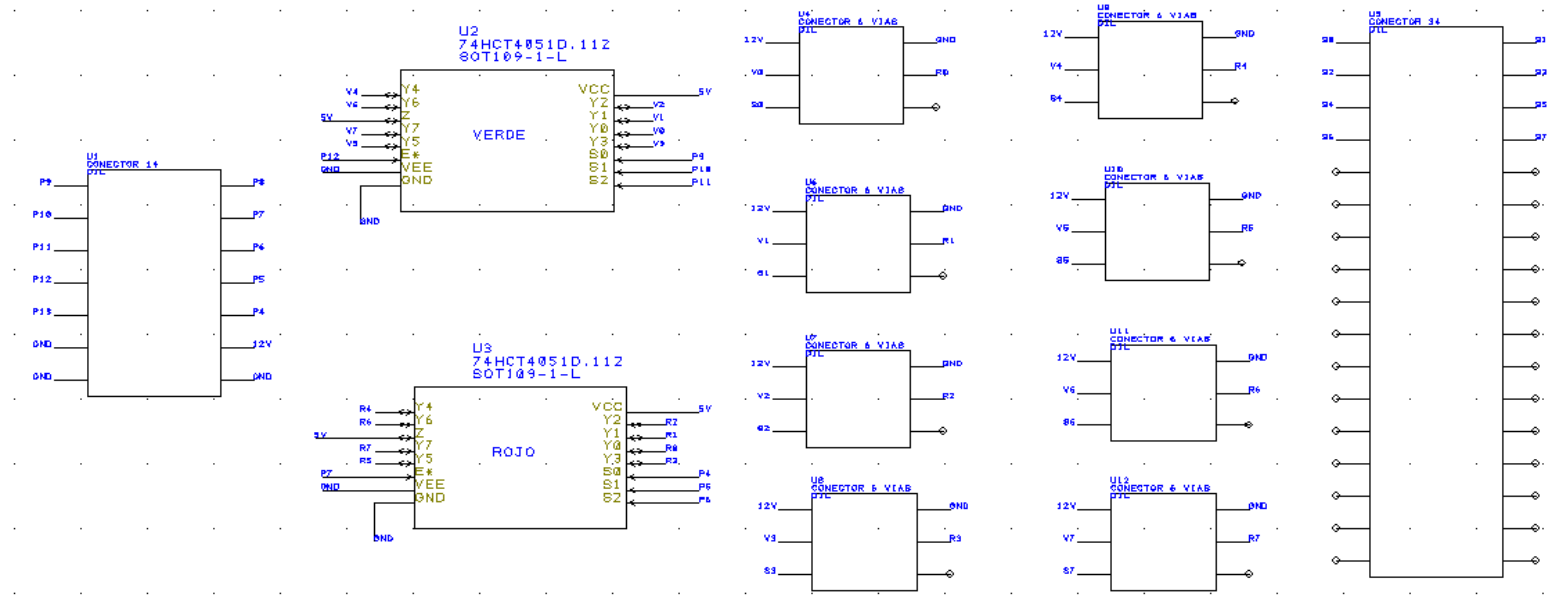


Fig. 6.2.2.5 Circuito completo de la tarjeta principal del prototipo

La tarjeta del prototipo se ha soldado toda a mano y el resultado ha sido el siguiente:

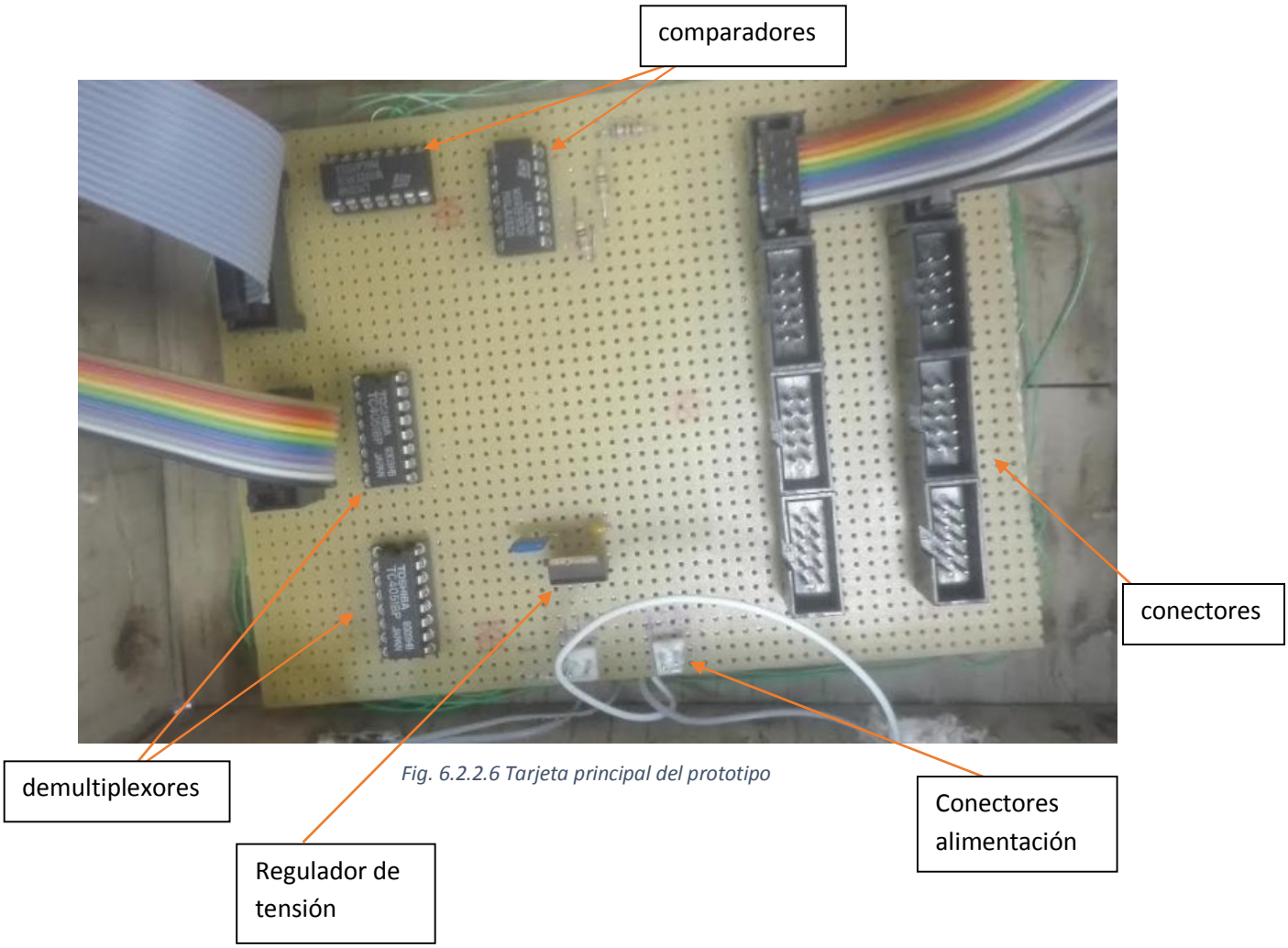


Fig. 6.2.2.6 Tarjeta principal del prototipo

6.2.3 Tarjeta LEDs/sensores

Para encender y apagar los LEDs, así como regular la tensión para estos y para el sensor, se ha diseñado una tarjeta la cual va dentro de la caja donde se encuentran la tarjeta de LEDs y el sensor.

El diseño del prototipo se ha hecho un poco diferente al diseño final, aunque muy similar al del diseño final, por temas de empresa.

El diseño del prototipo se ha realizado utilizando dos transistores, 6 resistencias y 2 conectores (uno para el sensor y otro para las señales).

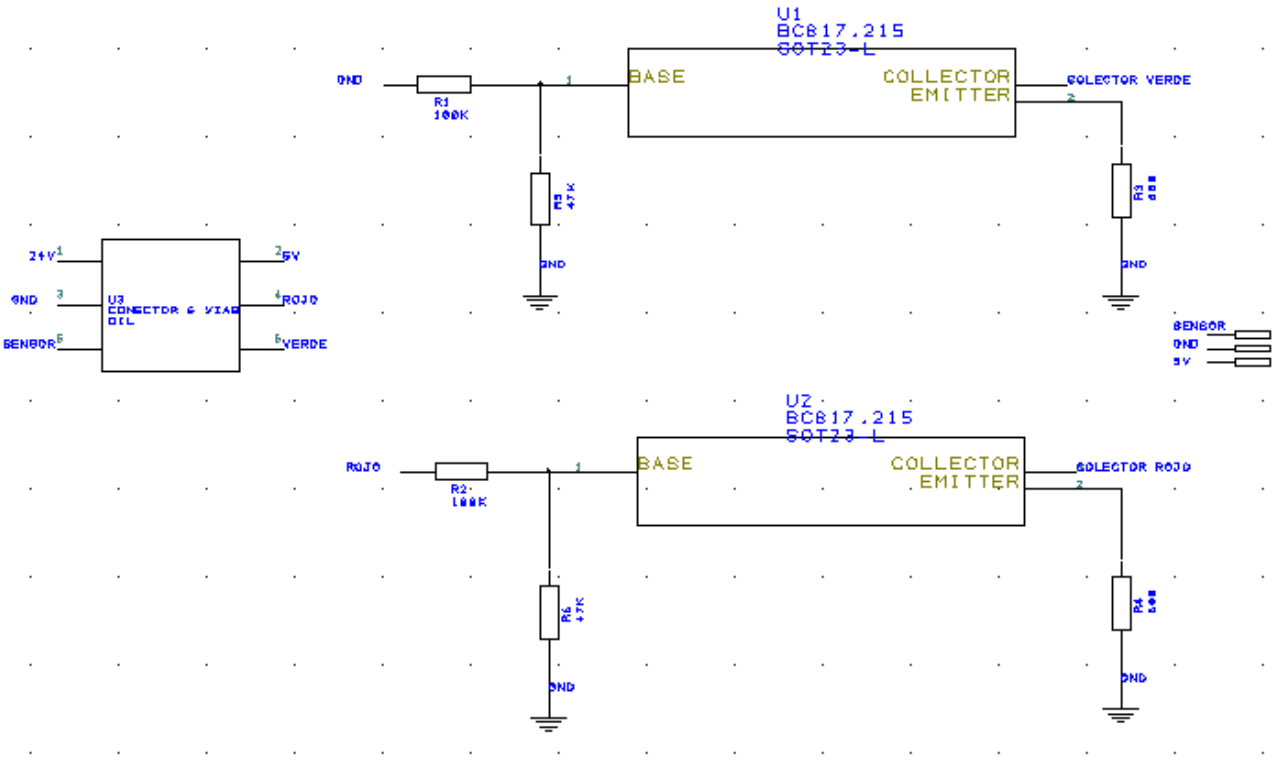


Fig. 6.2.3.1 Esquemático tarjeta caja

El prototipo ha sido montado en placas de prueba y soldado a mano. El resultado que ha quedado de este ha sido el siguiente.

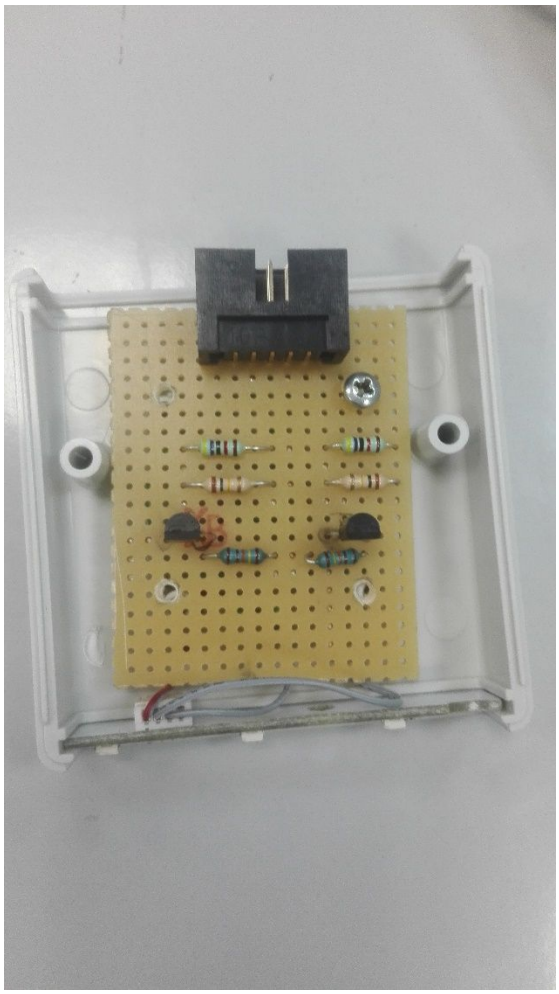


Fig. 6.2.3.2 Vista superior tarjeta prototipo

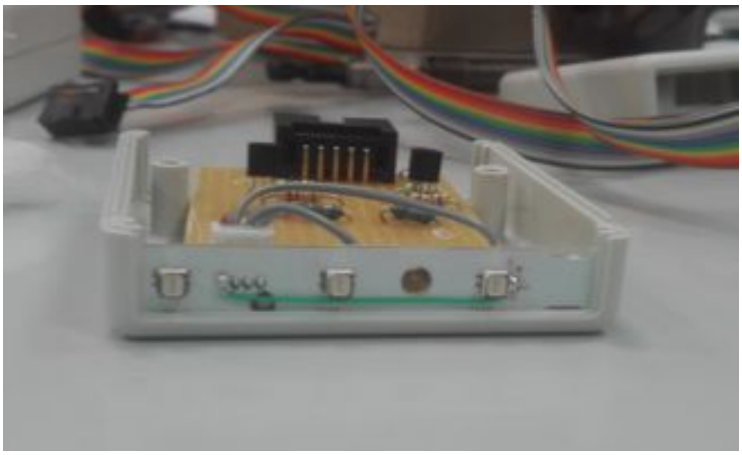


Fig. 6.2.3.3 Tarjeta prototipo en montaje

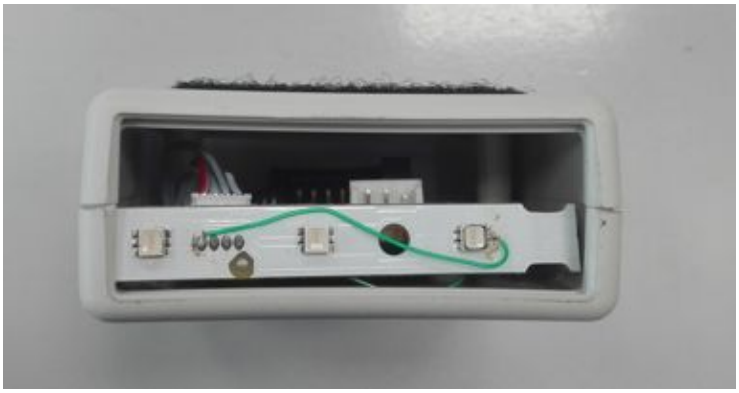


Fig. 6.2.3.4 Tarjeta prototipo finalizada

Se montaron 8 de estas tarjetas para poder validar el diseño y, tras comprobar su correcto funcionamiento, se diseñó la PCB para mandar a fabricar.

El montaje de las 8 cajas quedó de la siguiente manera, con el sensor incluido (aunque para el diseño final el sensor irá dentro de la caja).

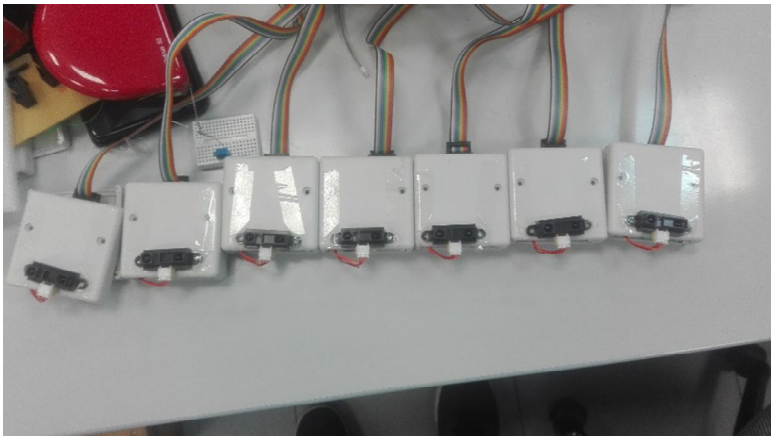


Fig. 6.2.3.5 Cajas montadas para el prototipo

El diseño de la tarjeta final se ha ido diseñando una vez comprobado todo el funcionamiento del prototipo.

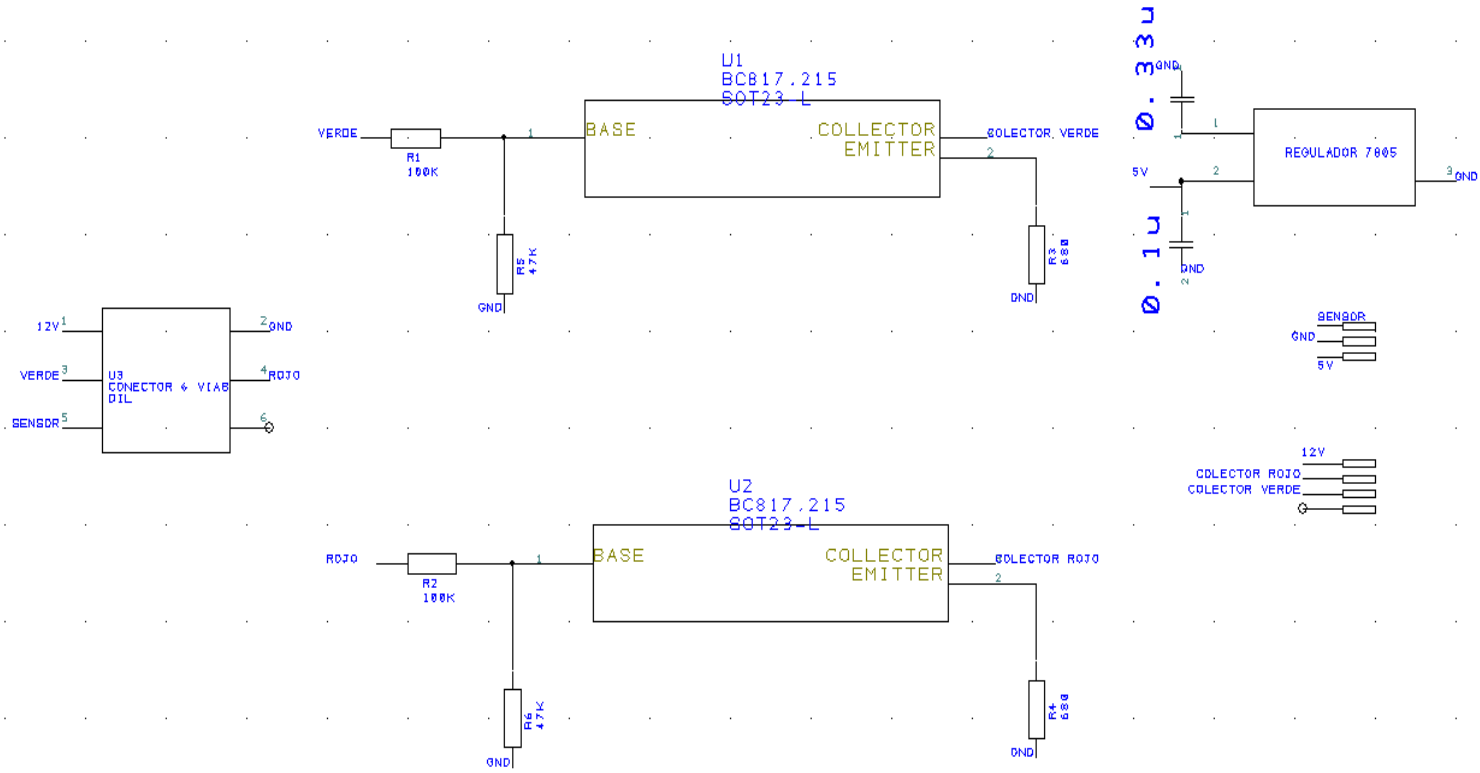


Fig. 6.2.3.6 Esquemático tarjeta final

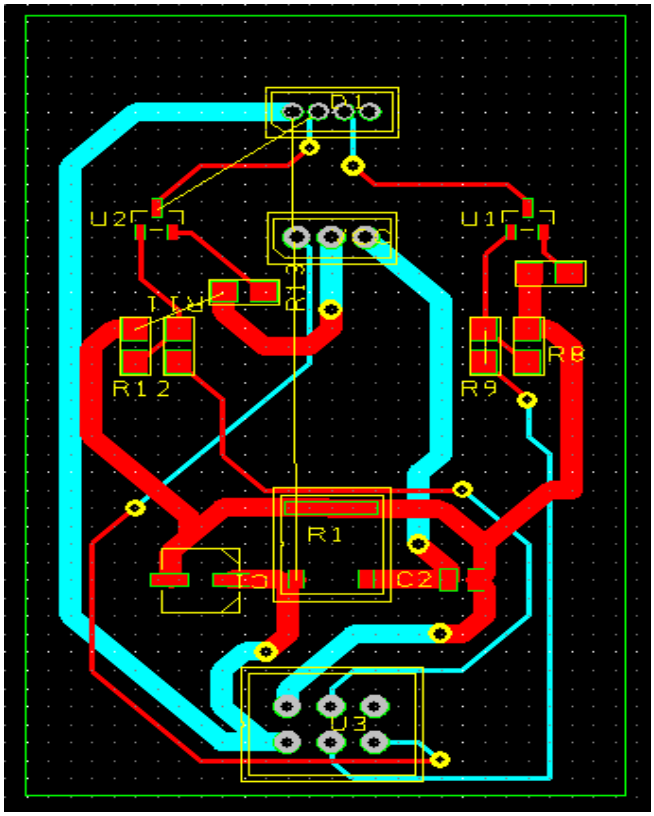


Fig. 6.2.3.7 Tarjeta final

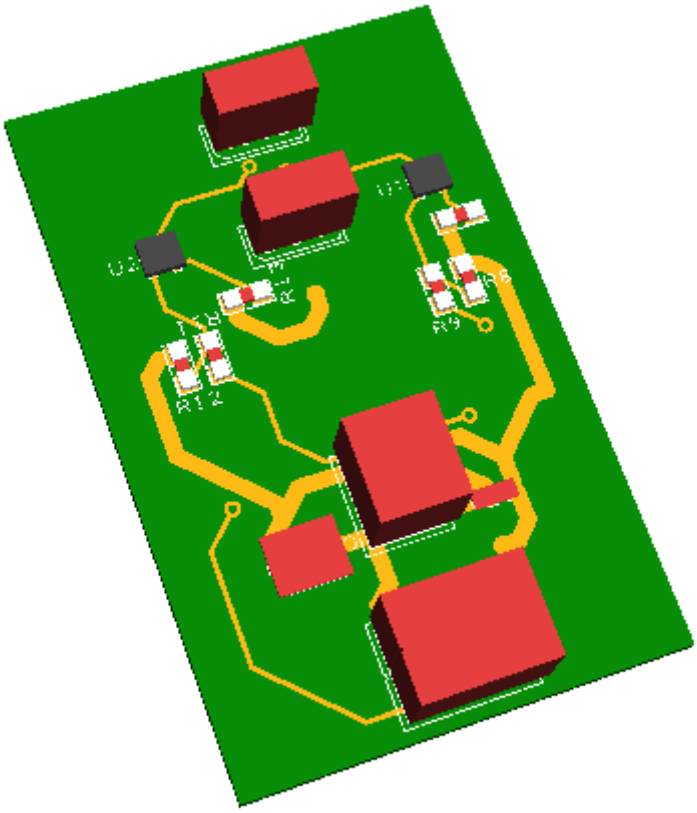


Fig. 6.2.3.8 Tarjeta final 3D(top)

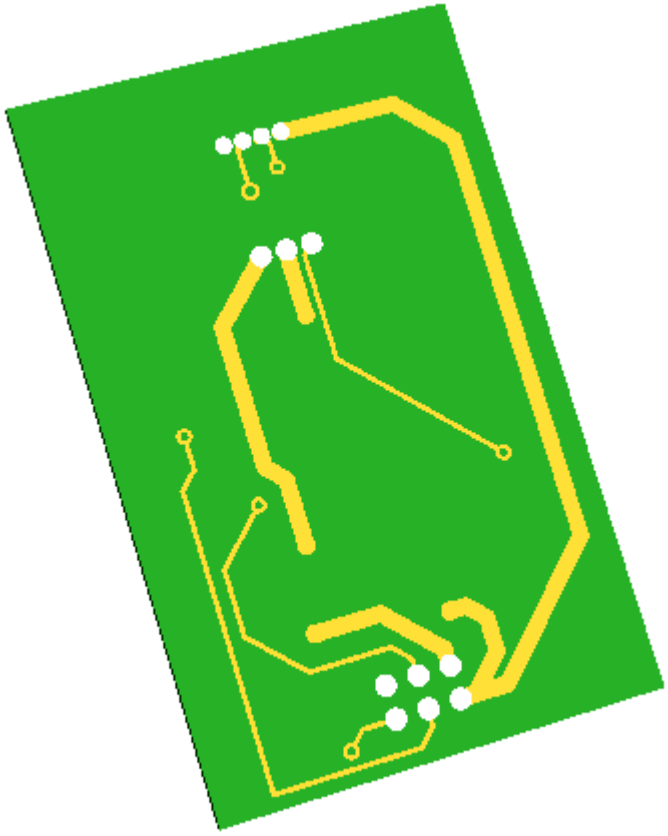


Fig. 6.2.3.9 Tarjeta final 3D(bottom)

6.3 Montaje

Para realizar el montaje se tuvo que pensar como sujetarlo al puesto de trabajo de una forma sencilla.

En cuanto a las tarjetas en la caja no ha habido ningún problema ya que la caja cuenta con unos agujeros para sujetar tarjetas electrónicas, por lo que la PCB ha sido diseñada para esa caja en concreto.

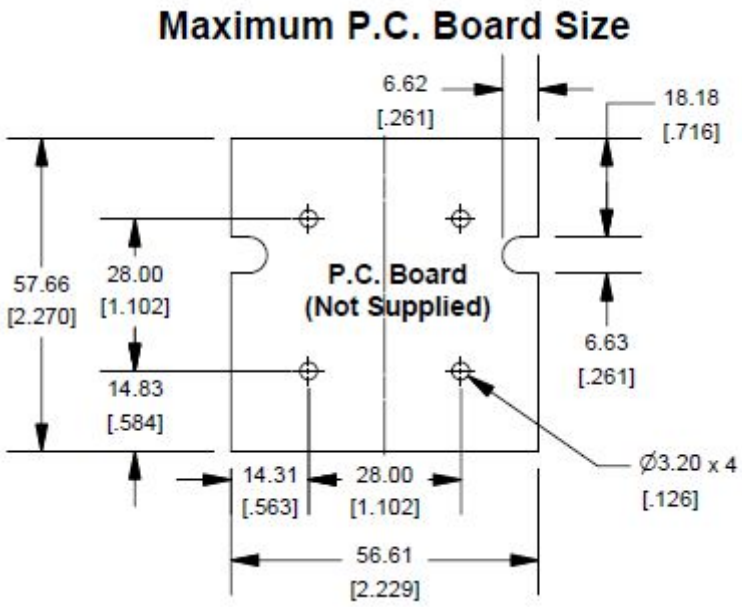


Fig. 6.3.1 Dimensiones máximas PCB caja

En cuanto a fijar las cajas en el puesto de trabajo, se han barajado varias opciones:

1. Perfiles

La primera opción de todas fue la de incluir un perfil bajo la bandeja donde se colocan las cajas con materiales. Atornillando unas chapas en forma de *omega* del tamaño de las cajas, se fijaban las cajas a los perfiles y estos a la bandeja mediante pegamento.

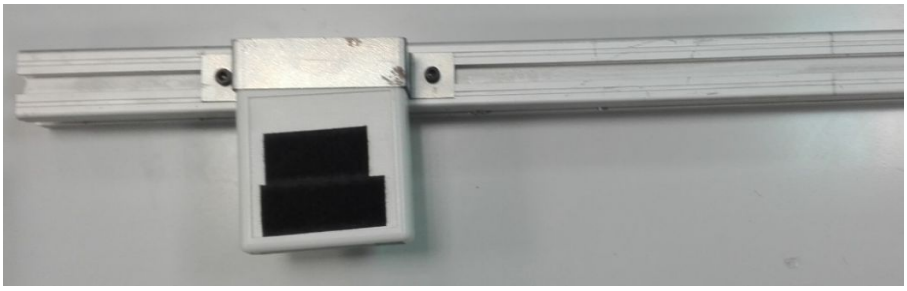


Fig. 6.3.2 Perfil con omega y caja fijada

Aun siendo una forma cómoda para fijar las cajas, surgía un problema muy importante para el proyecto: al poner los perfiles, las bandejas donde se colocan las cajas para montar quedaban demasiado altas ya que había que subirlas para asegurar que los sensores no detectaban la mesa. Esto supuso las quejas de las trabajadoras porque se encontraban demasiado altas las cajas con material. En conclusión, se descartó esta opción.

2. Imanes

La siguiente opción que se ha barajado ha sido la de usar imanes para fijar las cajas a la bandeja. Usando unos pequeños imanes en la parte superior de la tarjeta se pegarían a la bandeja y de esta forma se sujetarían.

Evidentemente, al hablar de temas de electricidad junto con magnetismo surgen varias preocupaciones por el efecto que tienen los campos magnéticos sobre las señales eléctricas. Además, en este caso se tratan de señales muy pequeñas, por lo que la influencia puede ser mayor.

Esta opción también ha sido descartada por este motivo, por la incertidumbre que producía el comportamiento de las señales cuando se encuentran bajo efectos de un campo magnético.

3. Velcro

Finalmente surgió la opción del velcro, algo que no había sido planteado en ningún momento y fue una muy buena idea en cuanto a comodidad y que no afecta en nada a las señales.

Como con todo, existen muchos tipos de velcro. El velcro elegido para fijar las cajas a las bandejas ha sido el *Dual Lock 3M,* el cual se ha comprobado su gran fuerza para soportar esfuerzos.

Fig. 6.3.3 Velcro Dual Lock 3M

Tras ponerles una pequeña parte a las cajas y una tira a la bandeja, las cajas han quedado fijadas de la siguiente manera:

Fig. 6.3.4 Cajas sujetas con velcro

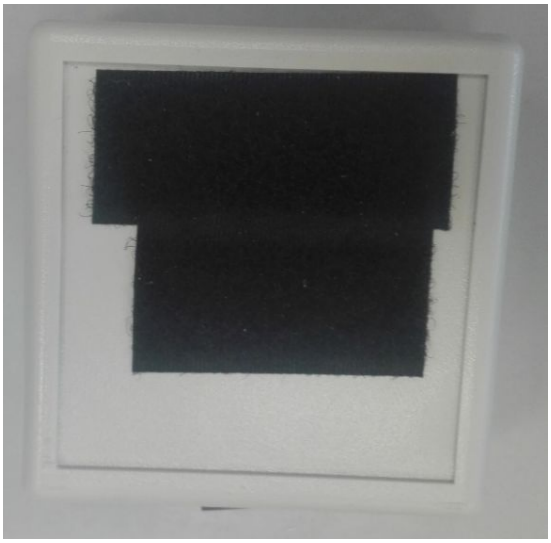


Fig. 6.3.5 Velcro en la parte superior de la caja

De esta forma se consigue con el objetivo marcado en este apartado, fijar las cajas de una forma muy sencilla y cómoda como que no influya en nada en el comportamiento del circuito electrónico.

7. PRESUPUESTO

En todos los proyectos, el presupuesto es algo a tener muy en cuenta a la hora de desarrollarlos.

En este caso se ha intentado reducir al máximo los costes de este proyecto utilizando materiales que ya se utilizan en la empresa, como podrían ser los elementos electrónicos, para poder montar las tarjetas en el apartado de SMD de Azkoyen y no tener que mandarlo a fabricar fuera de la empresa.

Además, se ha tratado de utilizar todos los materiales comprados a los proveedores con los que trabajan en la empresa.

PRECIO HARDWARE Y SOFTWARE:

Componentes	Precio/unidad	Precio 32 puestos
Arduino Mega2560	43,51 €	1.392,32 €
Shield ethernet arduino	19,98 €	639,36 €
Licencia LabView base	399,00 €	12.768,00 €
Regulador LM7805	0,44 €	14,08 €
Demultiplexor 4051	1,27 €	40,64 €
Conector molex 6 vías	1,85 €	59,20 €
Conector molex 14 vías	1,85 €	59,20 €
Conector molex 34 vías	1,85 €	59,20 €
Transistor BC337	0,14 €	4,48 €
Resistencia 100K	0,06 €	1,92 €
Resistencia 47K	0,06 €	1,92 €
Resistencia 680	0,06 €	1,92 €
Resistencia 100K	0,06 €	1,92 €
Caja sensores	3,21 €	102,72 €
Sensor sharp GP2Y0A21YK0F	9,14 €	292,48 €
Tarjeta LEDs	2,32 €	74,24 €
Cables sensores	1,23 €	39,36 €
Cableado puesto de trabajo	8,50 €	272,00 €
Fabricación tarjeta shield	5,00 €	160,00 €
Fabricación tarjeta principal	10,00 €	320,00 €
Fabricación tarjeta caja	5,00 €	160,00 €
TOTAL	**514,53 €**	**16.464,96 €**

PRECIO TRABAJADORES:

Cantidad trabajadores	Precio/hora	Cantidad horas	Total
1	30,00 €	650	**19.500,00 €**

PRECIO TOTAL:

HARDWARE+SOFTWARE+TRABAJADORES
35.964,96 €

8. CONCLUSIONES

Con este trabajo de final de carrera se ha conseguido realizar un proyecto en el cual se han podido aplicar diversos conocimientos de las ramas eléctrica, electrónica, telecomunicación e informática. A continuación, se presentan los hitos que se han conseguido a lo largo del desarrollo del proyecto:

- Implementar un sistema *pick to light* para toda la zona de producción de la empresa Azkoyen Medios de pago para guiar a los operarios en los movimientos a realizar.

- Desarrollo de una *interface* mediante el programa *LabView* para facilitar la manejabilidad al operario al cambiar la secuencia.

- Programación de la comunicación entre el ordenador y *Arduino* a través del protocolo de comunicación *ethernet* para el envío de la secuencia.

- Programación de *Arduino* para ser capaz de leer la secuencia, interpretarla y actuar en función de ello y la lectura de los sensores sobre los LEDs.

- Medir tiempos que le cuesta al operario para realizar las tareas y enviarlos al ordenador, pudiendo visualizarlos en tiempo real y ser almacenados en EXCEL.

- Guardar la secuencia generada y poder ser utilizada en cualquier momento (para no tener que reescribir la misma secuencia una y otra vez).

- Diseño de las tarjetas electrónicas necesarias para el desarrollo del proyecto: tarjeta *shield*, tarjeta principal y tarjeta de cada caja.

- Diseño de cables para los sensores, ya que no se contaba con este tipo de cableados con estos conectores en la empresa.

También se han adquirido diversos conocimientos de la ingeniería, así como temas de comunicaciones industriales, programación y diseño *hardware*.

Como se puede observar, se ha conseguido desarrollar un proyecto muy completo ya que se ha diseñado tanto el apartado *software* como el *hardware*.

9. LÍNEAS FUTURAS DE AMPLIACIÓN Y MEJORA DE LA APLICACIÓN

Debido a la brevedad del periodo de prácticas y, por consiguiente, para realizar el trabajo final de grado, no se han podido realizar todas las tareas deseadas.

Hasta el momento, se ha realizado la implementación del sistema *pick to light* en un puesto de trabajo prototipo, el cual era el objetivo por parte de la empresa. Además, se ha dejado diseñado todo el sistema para los 32 puestos de trabajo para la empresa, así como los pasos a seguir para la implantación.

Para la mejora del proyecto se ha pensado en alguna otra alternativa, sobre todo para minimizar el cableado de los puestos de trabajo, ya que actualmente cuentan con varios cables.

El proyecto realizado hasta ahora cuenta con que hay que llevar todas las señales a cada caja de cada dispositivo, su lectura del sensor, encendido LEDs y alimentación. Se podría haber utilizado sensores con un protocolo de comunicación (como por ejemplo I^2C) y que cada dispositivo hubiese tenido su dirección y poder actuar como *slave* y *master.* Para esto se habían encontrado sensores como el *TMD2672* el cual tiene un precio muy asequible y podría cumplir con esta función que se deseaba. Poniendo un microprocesador junto a ellos y programando la comunicación, se podría haber puesto los sensores en cascada y así ahorrando mucho cableado.

Este método ha sido documentado como futura mejora para la empresa, pero para la actualidad la empresa decidió no realizarlo por tema de tiempo (ya que era demasiado tarde para cambiar el diseño).

En cuanto a la interface, algo muy interesante sería poder modificar la secuencia en el instante que se equivoca al pulsar algún botón, ya que actualmente en caso de equivocación habría que comenzar desde el principio a teclear toda secuencia.

10. ANEXOS

1. Datasheet transistores BC337 (NPN)
2. Datasheet demultiplexores 4051
3. Arduino Mega 2560
4. Arduino shield ethernet
5. Regulador LM7805
6. Caja sensores
7. Sensor Sharp GP2Y0A41SK0F
8. Tarjeta LEDs Azkoyen
9. Cables sensores
10. Tarjeta shield
11. Tarjeta principal
12. Tarjeta caja LEDs y sensores
13. Programa LabView
14. Programa Arduino

10.1 Datasheet transistores BC337 (NPN)

BC337, BC337-25, BC337-40

Amplifier Transistors

NPN Silicon

Features

- These are Pb–Free Devices

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Collector – Emitter Voltage	V_{CEO}	45	Vdc
Collector – Base Voltage	V_{CBO}	50	Vdc
Emitter – Base Voltage	V_{EBO}	5.0	Vdc
Collector Current – Continuous	I_C	800	mAdc
Total Device Dissipation @ T_A = 25°C Derate above 25°C	P_D	625 5.0	mW mW/°C
Total Device Dissipation @ T_C = 25°C Derate above 25°C	P_D	1.5 12	W mW/°C
Operating and Storage Junction Temperature Range	T_J, T_{stg}	−55 to +150	°C

THERMAL CHARACTERISTICS

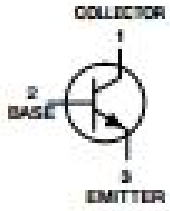
Characteristic	Symbol	Max	Unit
Thermal Resistance, Junction–to–Ambient	$R_{\theta JA}$	200	°C/W
Thermal Resistance, Junction–to–Case	$R_{\theta JC}$	83.3	°C/W

Stresses exceeding those listed in the Maximum Ratings table may damage the device. If any of these limits are exceeded, device functionality should not be assumed, damage may occur and reliability may be affected.

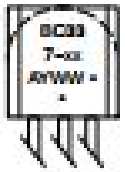
*For additional information on our Pb–Free strategy and soldering details, please download the ON Semiconductor Soldering and Mounting Techniques Reference Manual, SOLDERRM/D.

ON Semiconductor®

http://onsemi.com



MARKING DIAGRAM



BC337–xx = Device Code (Refer to page 4)
A = Assembly Location
Y = Year
WW = Work Week
▪ = Pb–Free Package
(Note: Microdot may be in either location)

ORDERING INFORMATION

See detailed ordering and shipping information in the package dimensions section on page 4 of this data sheet.

ethernet

BC337, BC337–25, BC337–40

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
OFF CHARACTERISTICS						
Collector – Emitter Breakdown Voltage (I_C = 10 mA, I_B = 0)		$V_{(BR)CEO}$	45	–	–	Vdc
Collector – Emitter Breakdown Voltage (I_C = 100 μA, I_E = 0)		$V_{(BR)CES}$	50	–	–	Vdc
Emitter – Base Breakdown Voltage (I_E = 10 μA, I_C = 0)		$V_{(BR)EBO}$	5.0	–	–	Vdc
Collector Cutoff Current (V_{CB} = 30 V, I_E = 0)		I_{CBO}	–	–	100	nAdc
Collector Cutoff Current (V_{CE} = 45 V, V_{BE} = 0)		I_{CES}	–	–	100	nAdc
Emitter Cutoff Current (V_{EB} = 4.0 V, I_C = 0)		I_{EBO}	–	–	100	nAdc
ON CHARACTERISTICS						
DC Current Gain		h_{FE}				–
(I_C = 100 mA, V_{CE} = 1.0 V)	BC337		100	–	630	
	BC337–25		160	–	400	
	BC337–40		250	–	630	
(I_C = 300 mA, V_{CE} = 1.0 V)			60	–	–	
Base–Emitter On Voltage (I_C = 300 mA, V_{CE} = 1.0 V)		$V_{BE(on)}$	–	–	1.2	Vdc
Collector – Emitter Saturation Voltage (I_C = 500 mA, I_B = 50 mA)		$V_{CE(sat)}$	–	–	0.7	Vdc
SMALL–SIGNAL CHARACTERISTICS						
Output Capacitance (V_{CB} = 10 V, I_E = 0, f = 1.0 MHz)		C_{ob}	–	15	–	pF
Current – Gain – Bandwidth Product (I_C = 10 mA, V_{CE} = 5.0 V, f = 100 MHz)		f_T	–	210	–	MHz

Product parametric performance is indicated in the Electrical Characteristics for the listed test conditions, unless otherwise noted. Product performance may not be indicated by the Electrical Characteristics if operated under different conditions.

10.2 Datasheet demultiplexores 4051

74HC4051; 74HCT4051

8-channel analog multiplexer/demultiplexer

Rev. 8 — 5 February 2016 Product data sheet

1. General description

The 74HC4051; 74HCT4051 is a single-pole octal-throw analog switch (SP8T) suitable for use in analog or digital 8:1 multiplexer/demultiplexer applications. The switch features three digital select inputs (S0, S1 and S2), eight independent inputs/outputs (Yn), a common input/output (Z) and a digital enable input ($\overline{E}$). When $\overline{E}$ is HIGH, the switches are turned off. Inputs include clamp diodes. This enables the use of current limiting resistors to interface inputs to voltages in excess of V_{CC}.

2. Features and benefits

- Wide analog input voltage range from –5 V to +5 V
- Complies with JEDEC standard no. 7A
- Low ON resistance:
  - 80 Ω (typical) at $V_{CC} – V_{EE}$ = 4.5 V
  - 70 Ω (typical) at $V_{CC} – V_{EE}$ = 6.0 V
  - 60 Ω (typical) at $V_{CC} – V_{EE}$ = 9.0 V
- Logic level translation: to enable 5 V logic to communicate with ±5 V analog signals
- Typical 'break before make' built-in
- ESD protection:
  - HBM JESD22-A114F exceeds 2000 V
  - MM JESD22-A115-A exceeds 200 V
  - CDM JESD22-C101E exceeds 1000 V
- Multiple package options
- Specified from –40 °C to +85 °C and –40 °C to +125 °C

4. Ordering information

Table 1. Ordering information

Type number	Package			
	Temperature range	Name	Description	Version
74HC4051D	−40 °C to +125 °C	SO16	plastic small outline package; 16 leads; body width 3.9 mm	SOT109-1
74HCT4051D				
74HC4051DB	−40 °C to +125 °C	SSOP16	plastic shrink small outline package; 16 leads; body width 5.3 mm	SOT338-1
74HCT4051DB				
74HC4051PW	−40 °C to +125 °C	TSSOP16	plastic thin shrink small outline package; 16 leads; body width 4.4 mm	SOT403-1
74HCT4051PW				
74HC4051BQ	−40 °C to +125 °C	DHVQFN16	plastic dual in-line compatible thermal enhanced very thin quad flat package; no leads; 16 terminals; body 2.5 × 3.5 × 0.85 mm	SOT763-1
74HCT4051BQ				

5. Functional diagram

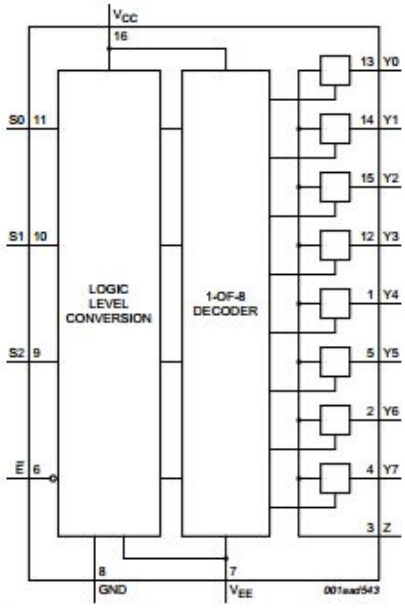


Fig 1. Functional diagram

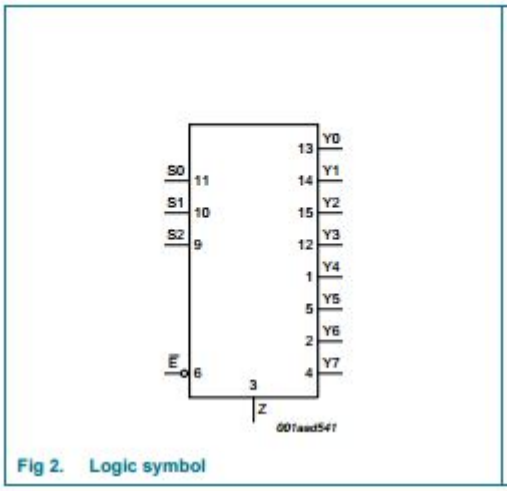


Fig 2. Logic symbol

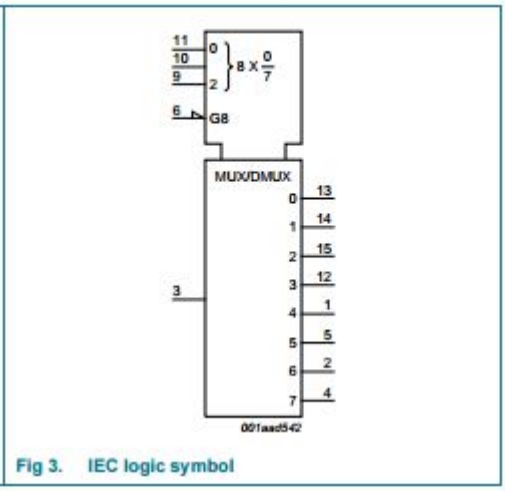


Fig 3. IEC logic symbol

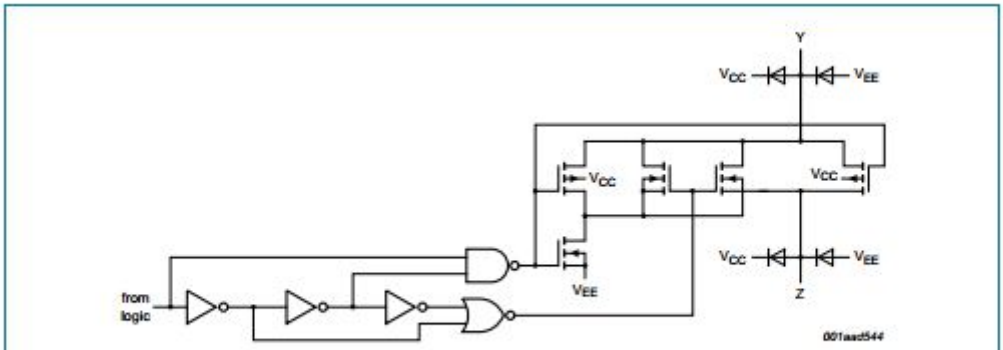


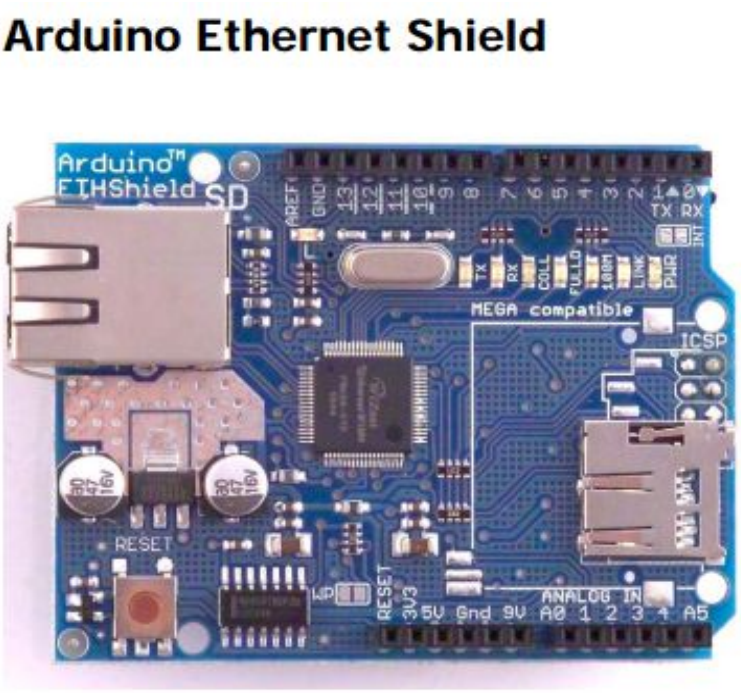
Fig 4. Schematic diagram (one switch)

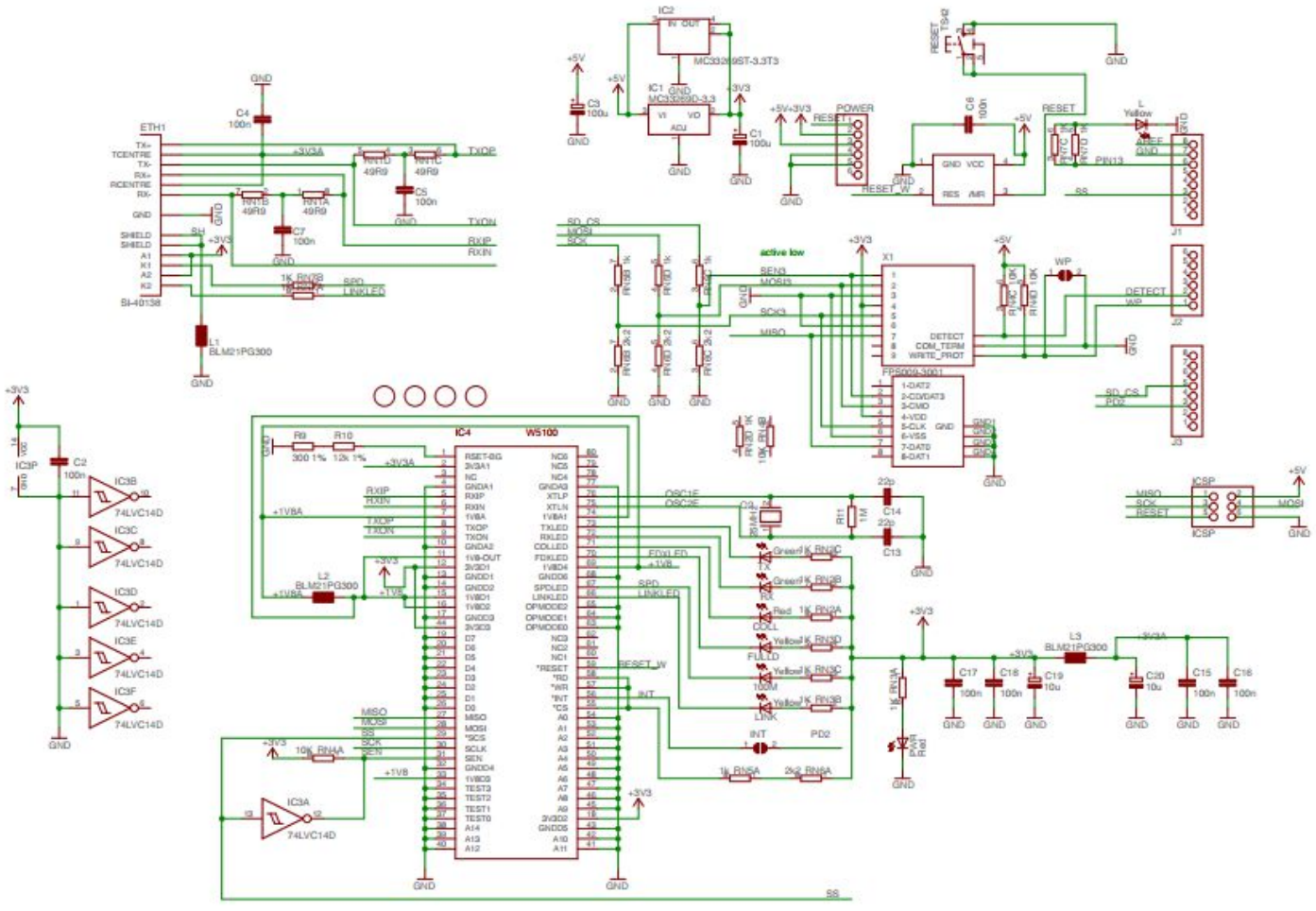
10.3 Arduino Mega 2560

Technical specs

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

10.4 Arduino shield ethernet





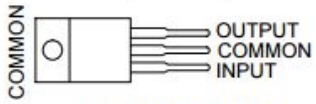
10.5 Regulador LM7805

μA7800 SERIES
POSITIVE-VOLTAGE REGULATORS

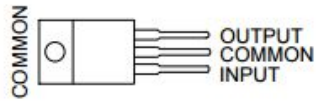
SLVS056J – MAY 1976 – REVISED MAY 2003

- 3-Terminal Regulators
- Output Current up to 1.5 A
- Internal Thermal-Overload Protection
- High Power-Dissipation Capability
- Internal Short-Circuit Current Limiting
- Output Transistor Safe-Area Compensation

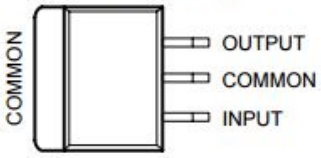
KC (TO-220) PACKAGE
(TOP VIEW)



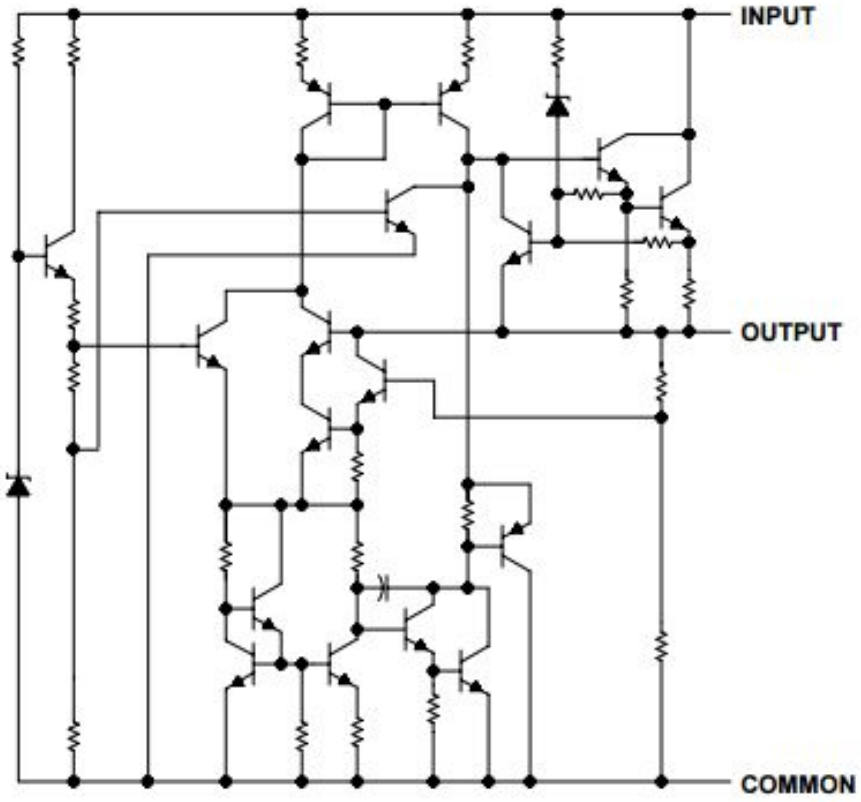
KCS (TO-220) PACKAGE
(TOP VIEW)



KTE PACKAGE
(TOP VIEW)



schematic



absolute maximum ratings over virtual junction temperature range (unless otherwise noted)†

Input voltage, V_I: µA7824C . . . . . 40 V
All others . . . . . 35 V
Operating virtual junction temperature, T_J . . . . . 150°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds . . . . . 260°C
Storage temperature range, T_{stg} . . . . . –65°C to 150°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

recommended operating conditions

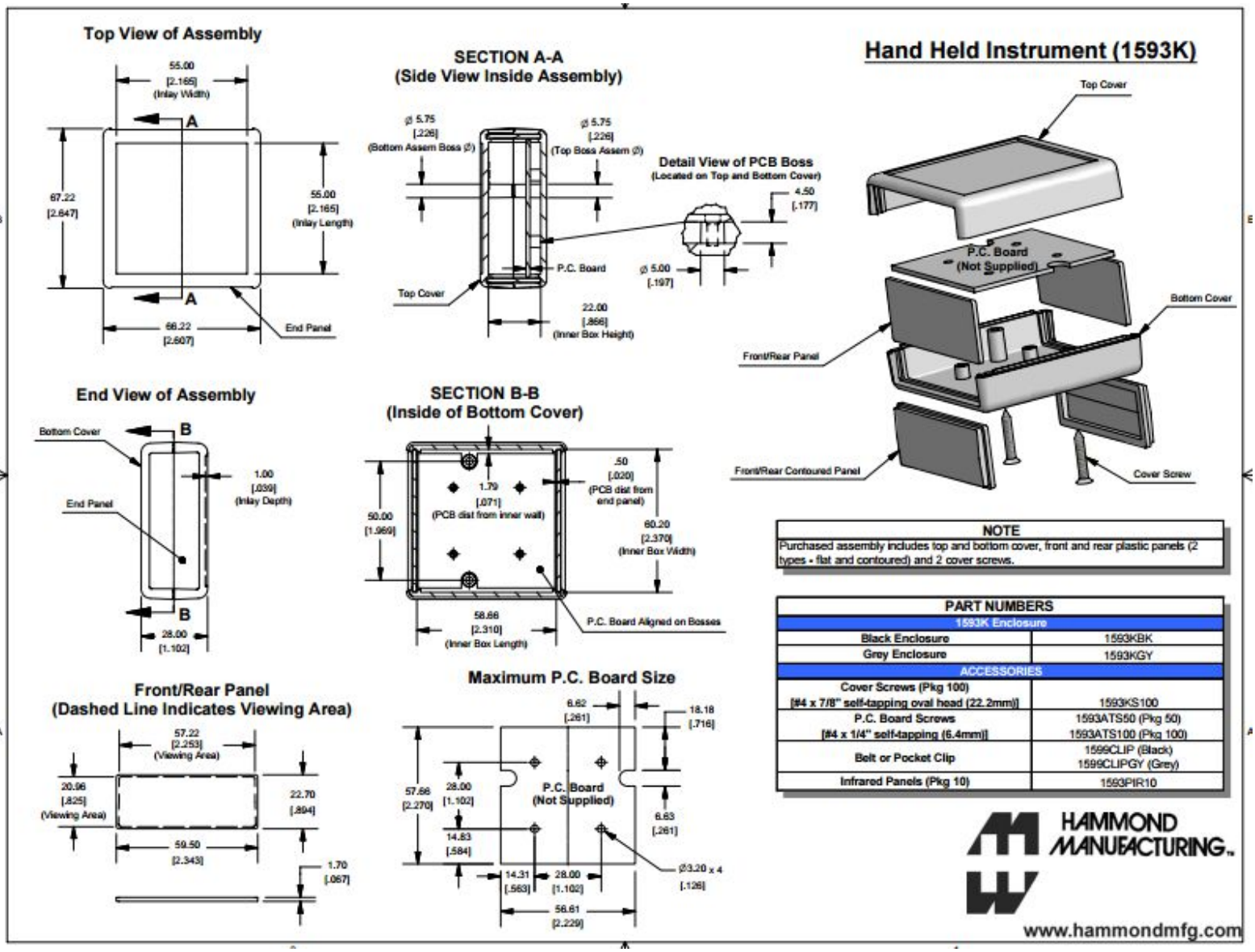
			MIN	MAX	UNIT
V_I	Input voltage	µA7805C	7	25	V
		µA7808C	10.5	25	
		µA7810C	12.5	28	
		µA7812C	14.5	30	
		µA7815C	17.5	30	
		µA7824C	27	38	
I_O	Output current			1.5	A
T_J	Operating virtual junction temperature	µA7800C series	0	125	°C

electrical characteristics at specified virtual junction temperature, V_I = 10 V, I_O = 500 mA (unless otherwise noted)

PARAMETER	TEST CONDITIONS	T_J†	µA7805C			UNIT
			MIN	TYP	MAX	
Output voltage	I_O = 5 mA to 1 A, V_I = 7 V to 20 V, $P_D \le 15$ W	25°C	4.8	5	5.2	V
		0°C to 125°C	4.75		5.25	
Input voltage regulation	V_I = 7 V to 25 V	25°C		3	100	mV
	V_I = 8 V to 12 V			1	50	
Ripple rejection	V_I = 8 V to 18 V, f = 120 Hz	0°C to 125°C	62	78		dB
Output voltage regulation	I_O = 5 mA to 1.5 A	25°C		15	100	mV
	I_O = 250 mA to 750 mA			5	50	
Output resistance	f = 1 kHz	0°C to 125°C		0.017		Ω
Temperature coefficient of output voltage	I_O = 5 mA	0°C to 125°C		–1.1		mV/°C
Output noise voltage	f = 10 Hz to 100 kHz	25°C		40		µV
Dropout voltage	I_O = 1 A	25°C		2		V
Bias current		25°C		4.2	8	mA
Bias current change	V_I = 7 V to 25 V	0°C to 125°C			1.3	mA
	I_O = 5 mA to 1 A				0.5	
Short-circuit output current		25°C		750		mA
Peak output current		25°C		2.2		A

† Pulse-testing techniques maintain the junction temperature as close to the ambient temperature as possible. Thermal effects must be taken into account separately. All characteristics are measured with a 0.33-µF capacitor across the input and a 0.1-µF capacitor across the output.

10.6 Caja sensores



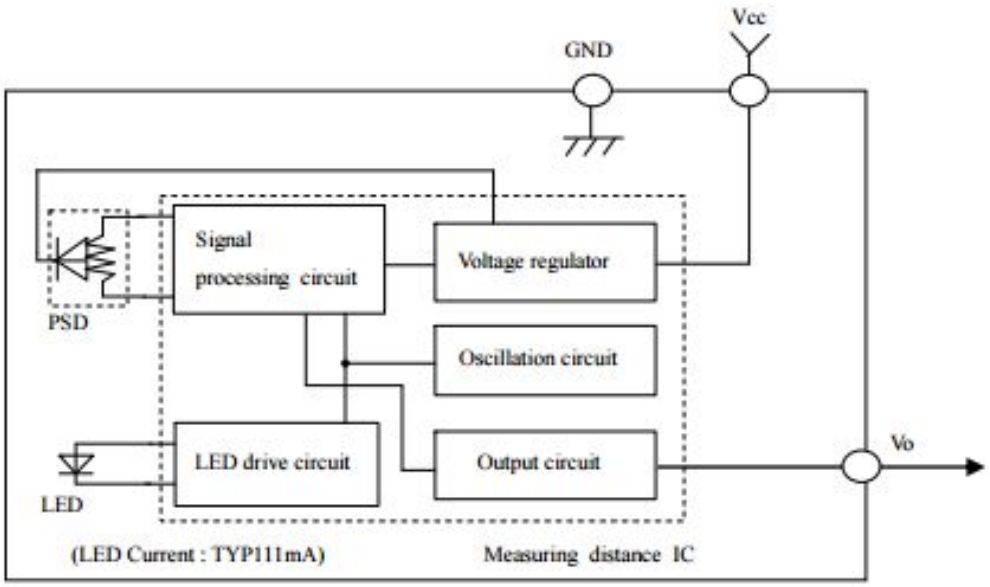
10.7 Sensor Sharp GP2Y0A41SK0F

SHARP GP2Y0A41SK0F

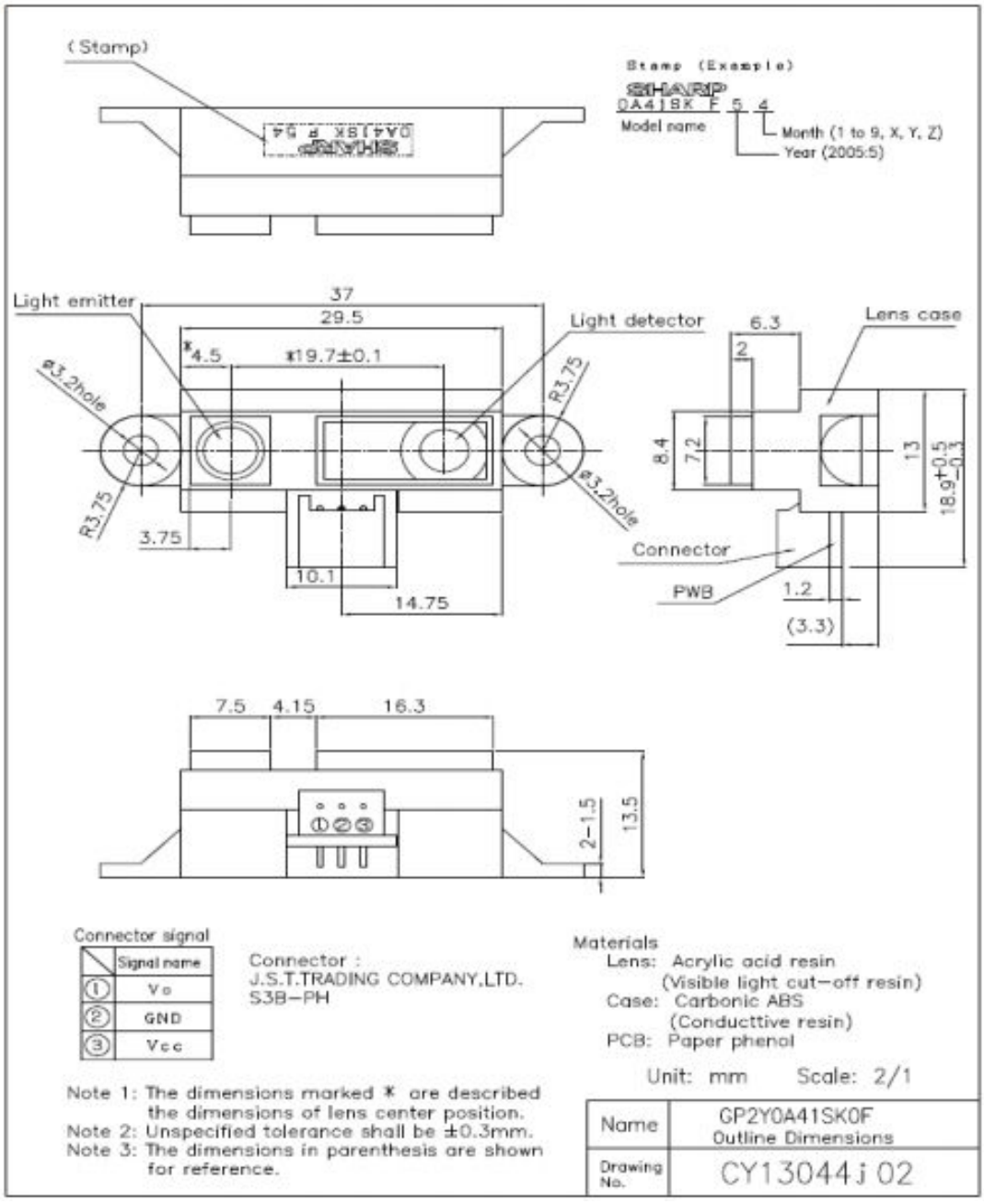
GP2Y0A41SK0F

Distance Measuring Sensor Unit
Measuring distance : 4 to 30 cm
Analog output type

■Schematic



■Outline



Connector signal

	Signal name
①	Vo
②	GND
③	Vcc

Connector :
J.S.T.TRADING COMPANY,LTD.
S3B-PH

Materials
Lens: Acrylic acid resin
(Visible light cut-off resin)
Case: Carbonic ABS
(Conducttive resin)
PCB: Paper phenol

Unit: mm Scale: 2/1

Note 1: The dimensions marked * are described the dimensions of lens center position.
Note 2: Unspecified tolerance shall be ±0.3mm.
Note 3: The dimensions in parenthesis are shown for reference.

Name	GP2Y0A41SK0F Outline Dimensions
Drawing No.	CY13044j 02

■Absolute maximum ratings

(Ta=25°C, Vcc=5V)

Parameter	Symbol	Ratings	Unit	Remark
Supply voltage	Vcc	-0.3 to +7	V	-
Output terminal voltage	Vo	-0.3 to Vcc+0.3	V	-
Operating temperature	Topr	-10 to +60	°C	-
Storage temperature	Tstg	-40 to +70	°C	-

■Operating supply voltage

Symbol	Rating	Unit	Remark
Vcc	4.5 to 5.5	V	-

■Electro-optical Characteristics

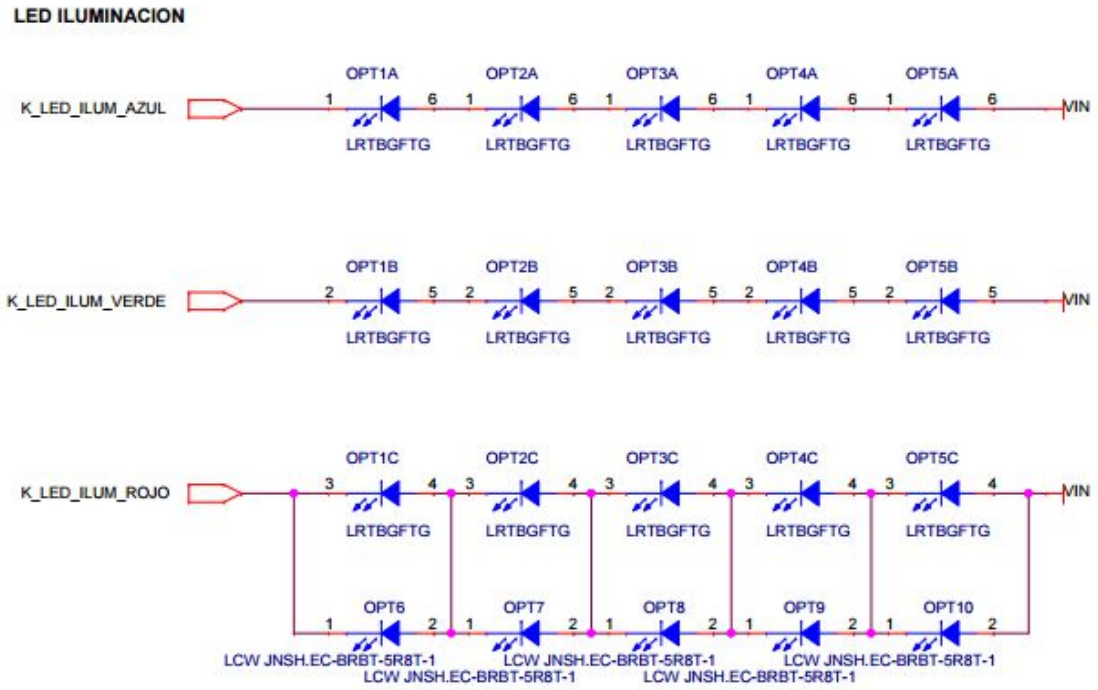
Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Measuring distance range	ΔL	(Note 1)	4	-	30	Cm
Output terminal voltage	Vo	L=30cm (Note 1)	0.25	0.4	0.55	V
Output voltage difference	ΔVo	Output change at L change (30cm → 4cm) (Note 1)	1.95	2.25	2.55	V
Average supply current	Icc	L=30cm (Note 1)	-	12	22	mA

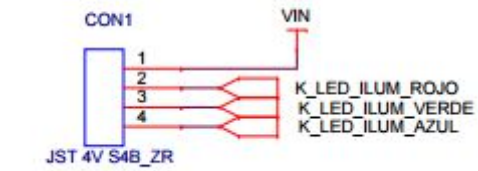
※L : Distance to reflective object

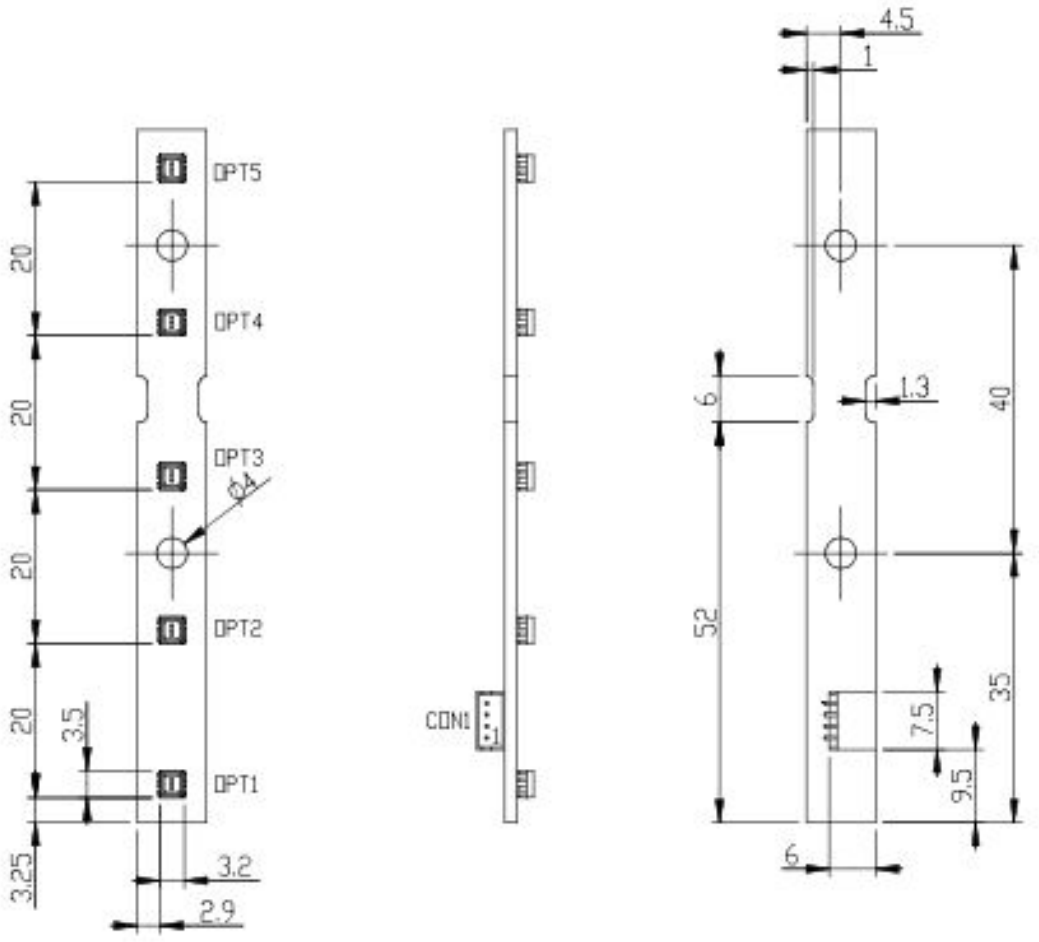
(Note 1) Using reflective object : White paper

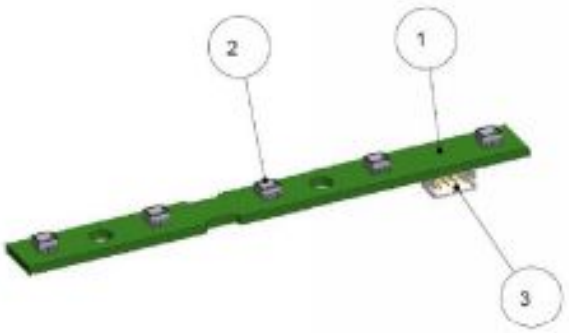
(Made by Kodak Co., Ltd. gray cards R-27 · white face, reflective ratio ; 90%)

10.8 Tarjeta LEDs Azkoyen



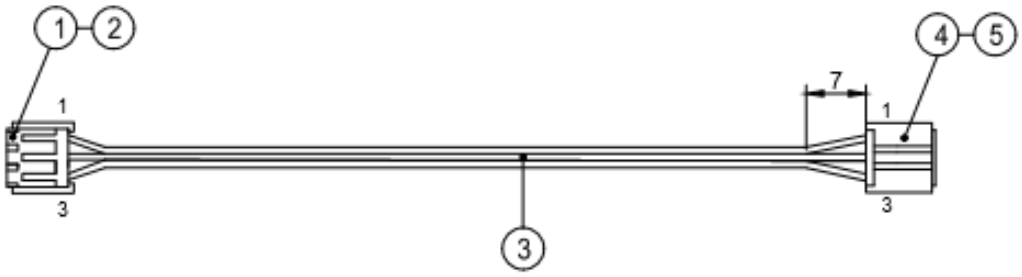



OPT5
OPT4
OPT3
OPT2
OPT1
ϕ4
20
20
20
20
3.5
3.25
3.2
2.9
CON1
4.5
1
1.3
6
40
52
35
7.5
9.5
6

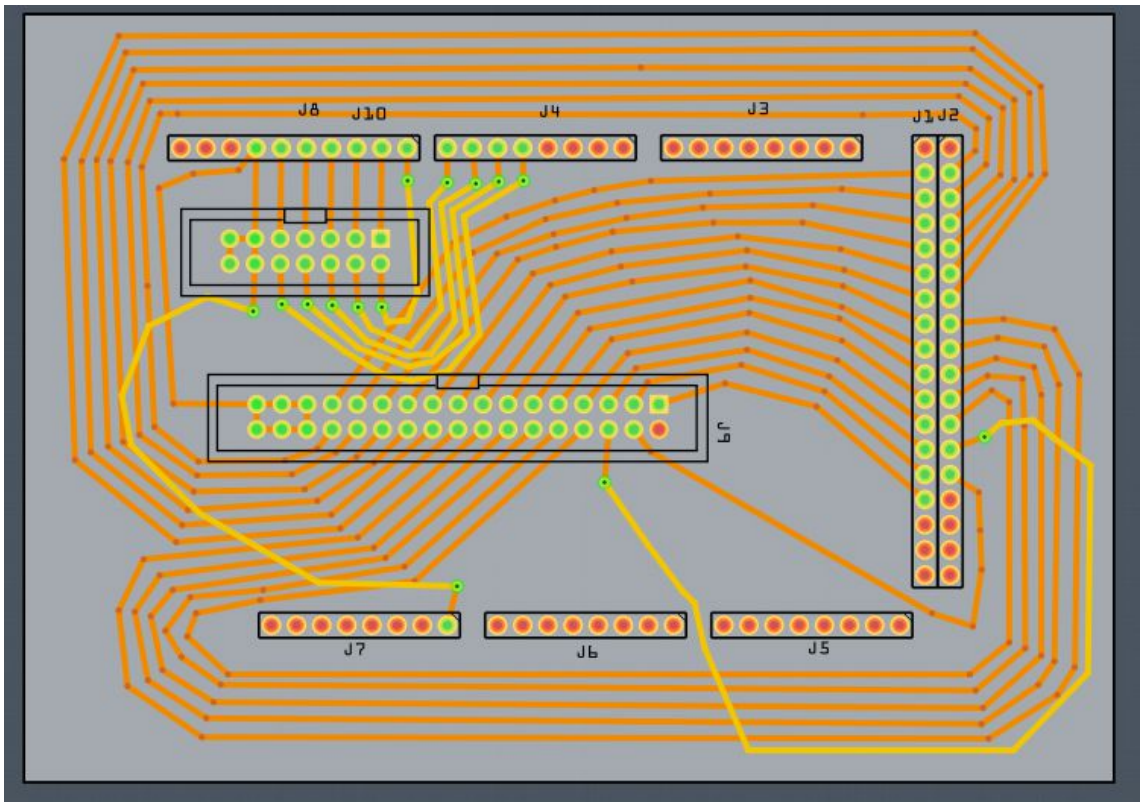

2
1
3

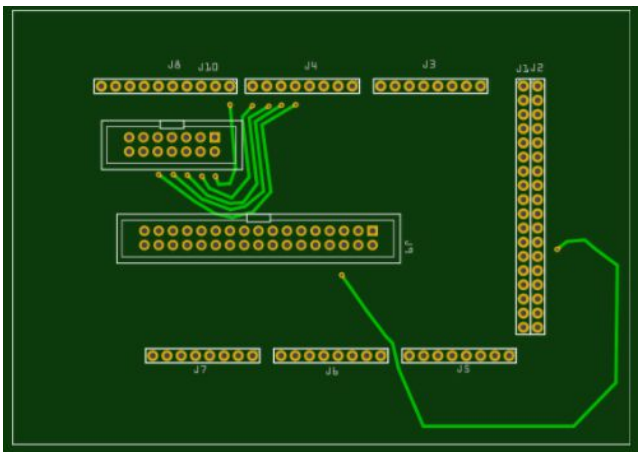
10.9 Cables sensores

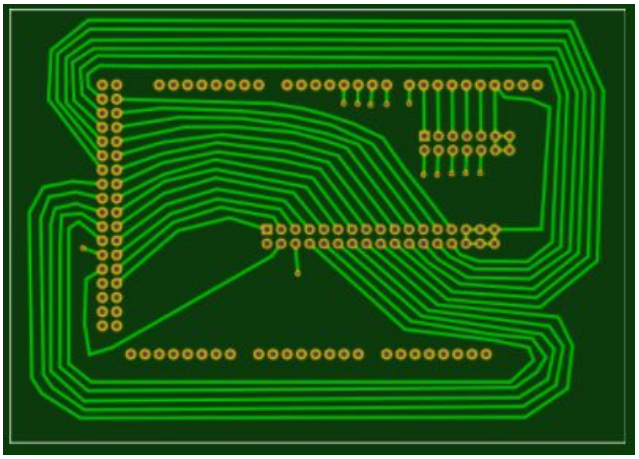
MARCA	NUMERO	DENOMINACION	Nº VIAS	NORMA TENSION	SECC/DIAM	COLOR	CANTIDAD/LONGITUD
	43232610-0	MAZO CONECTORES SENSOR PICK TO LIGHT					
1	03810840-0	CONECTOR JST 3 VIAS PHR-3					1
2	03903700-0	TERMINAL JST SPH-002T-P0.5S					3
3	23100130-0	CABLE GRIS	3		0.1	GRIS	60
4	03802660-0	CTOR. JST 3V EHR-3 HEMBRA					1
5	03900530-0	TERMINAL JST SEH 001T					3



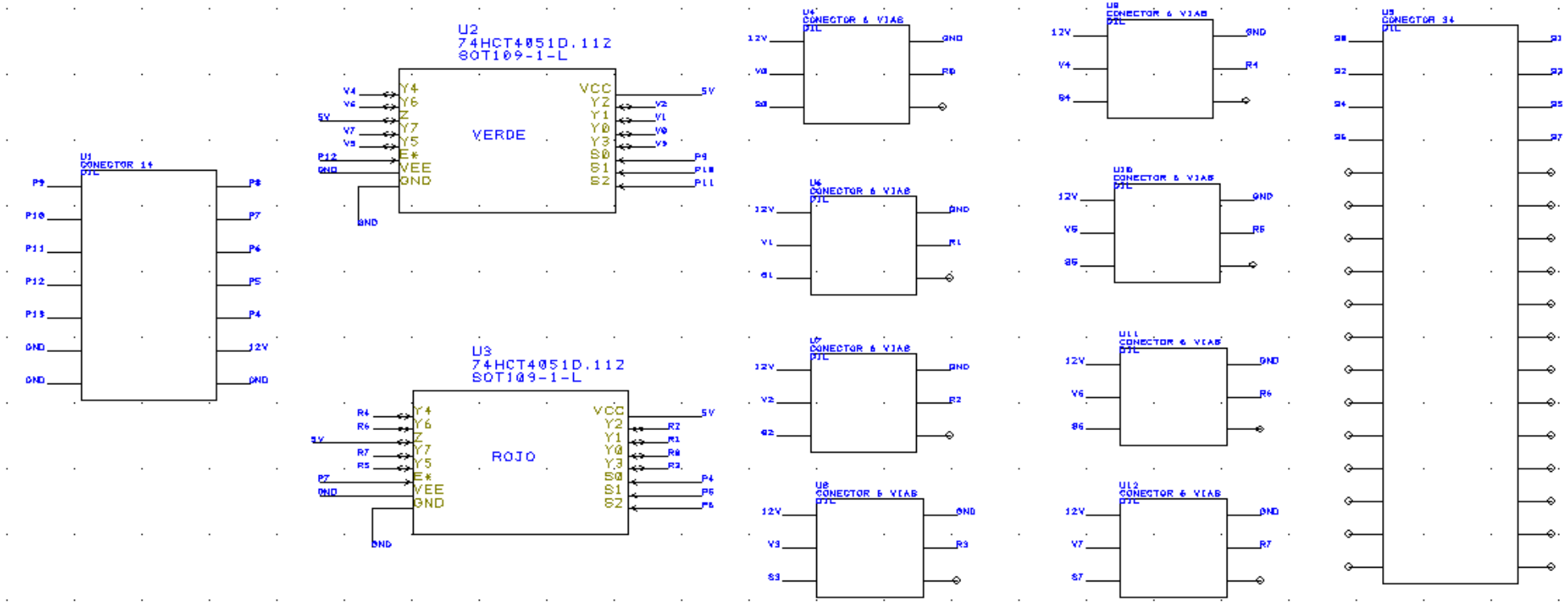
10.10 Tarjeta shield



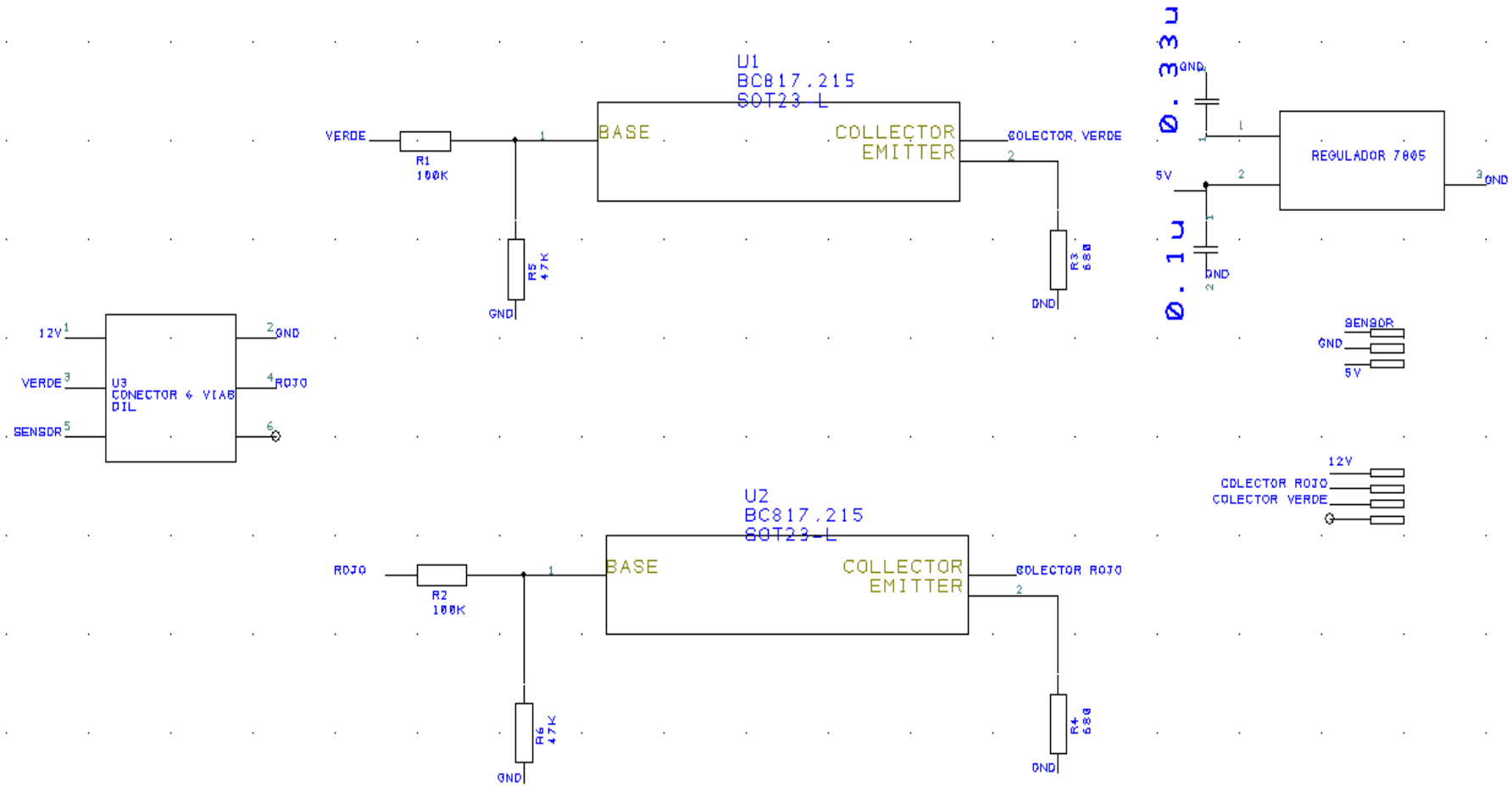


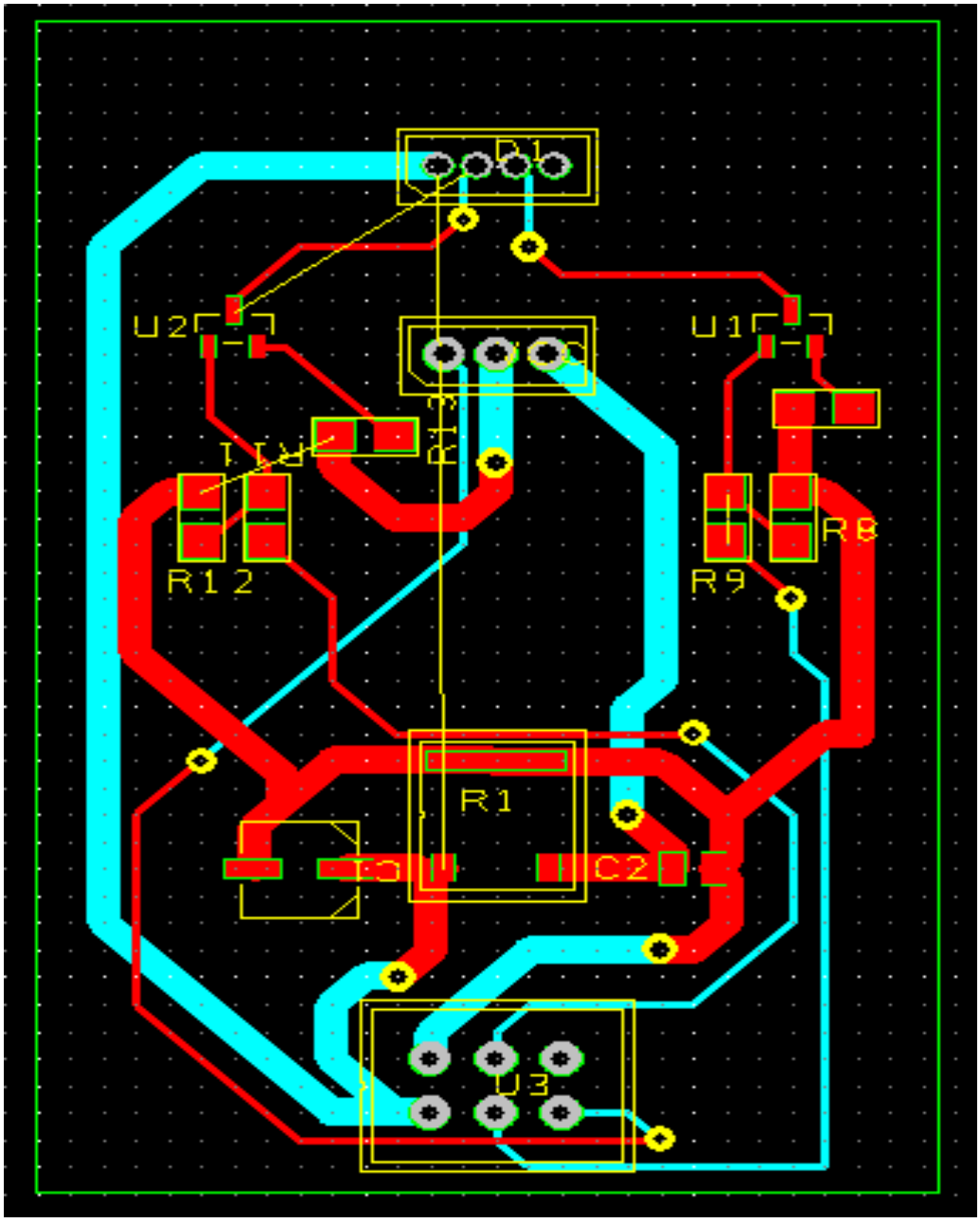


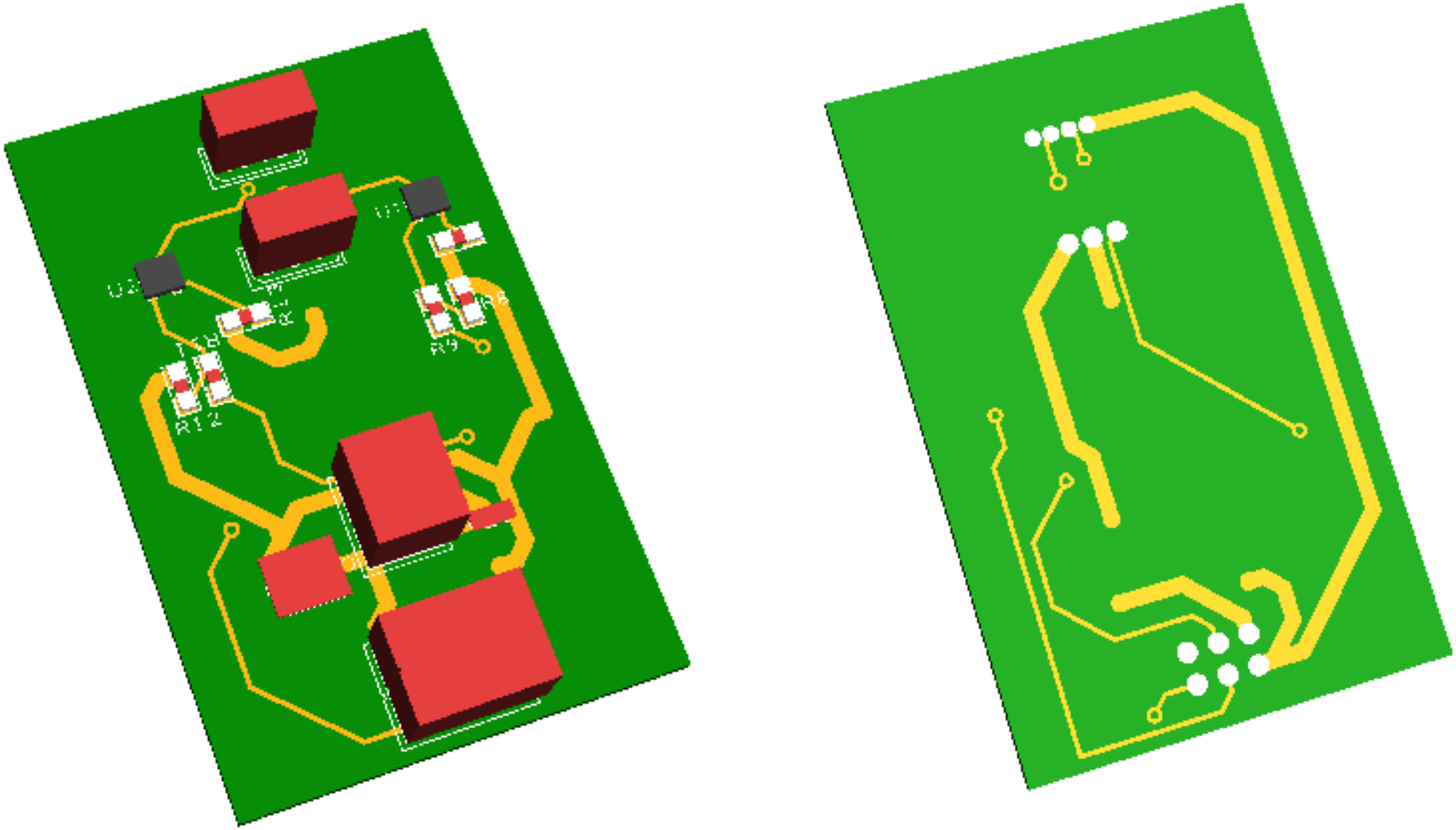
10.11 Tarjeta principal



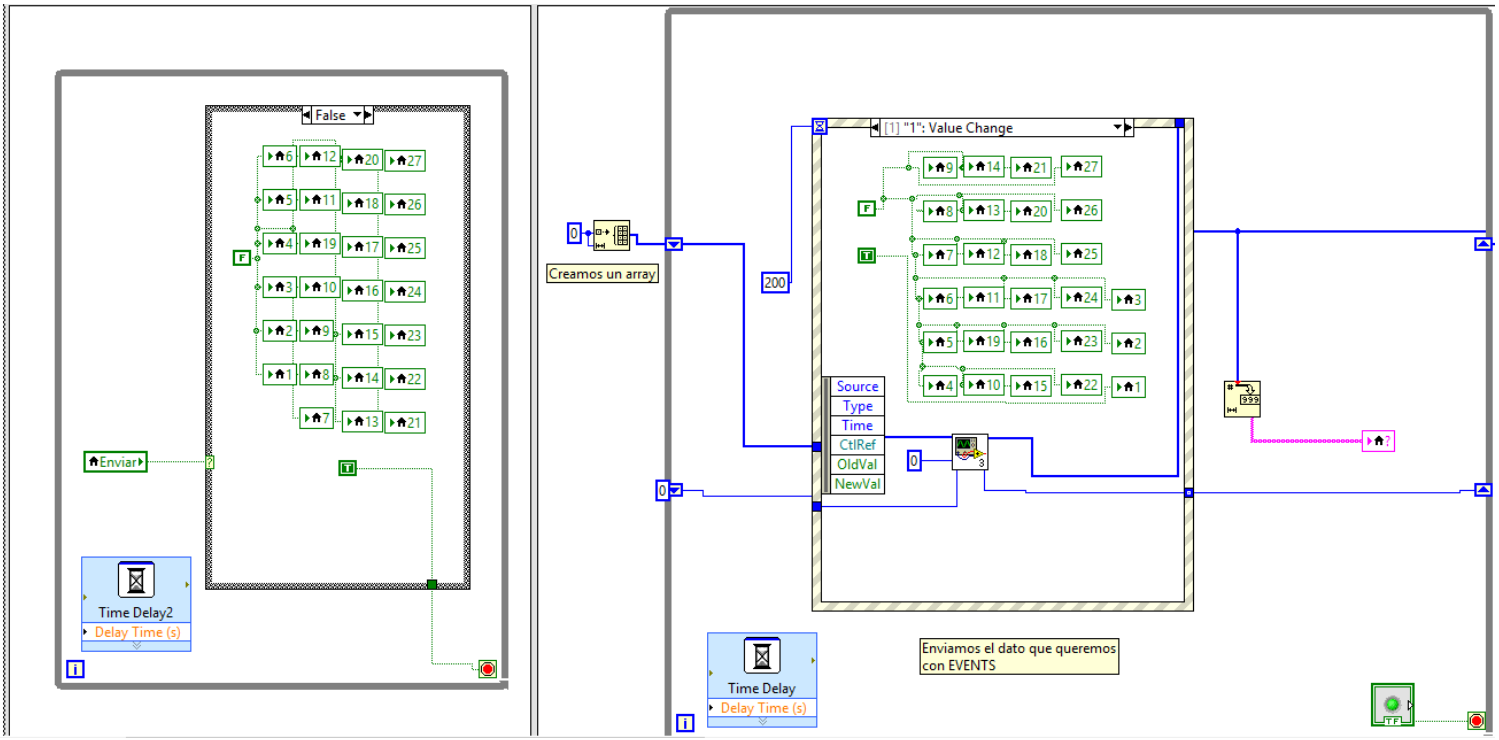
10.12 Tarjeta caja LEDs y sensores

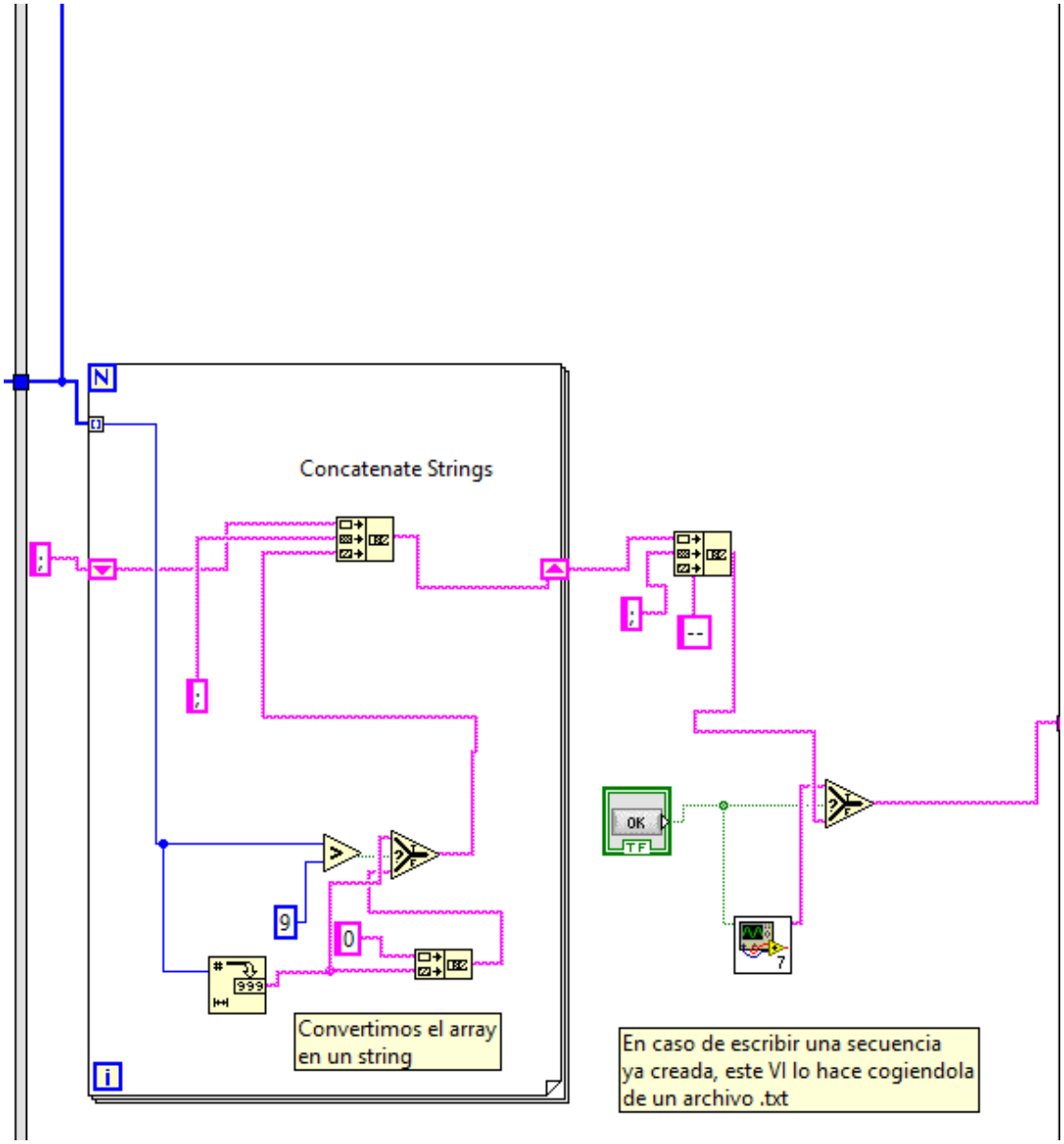


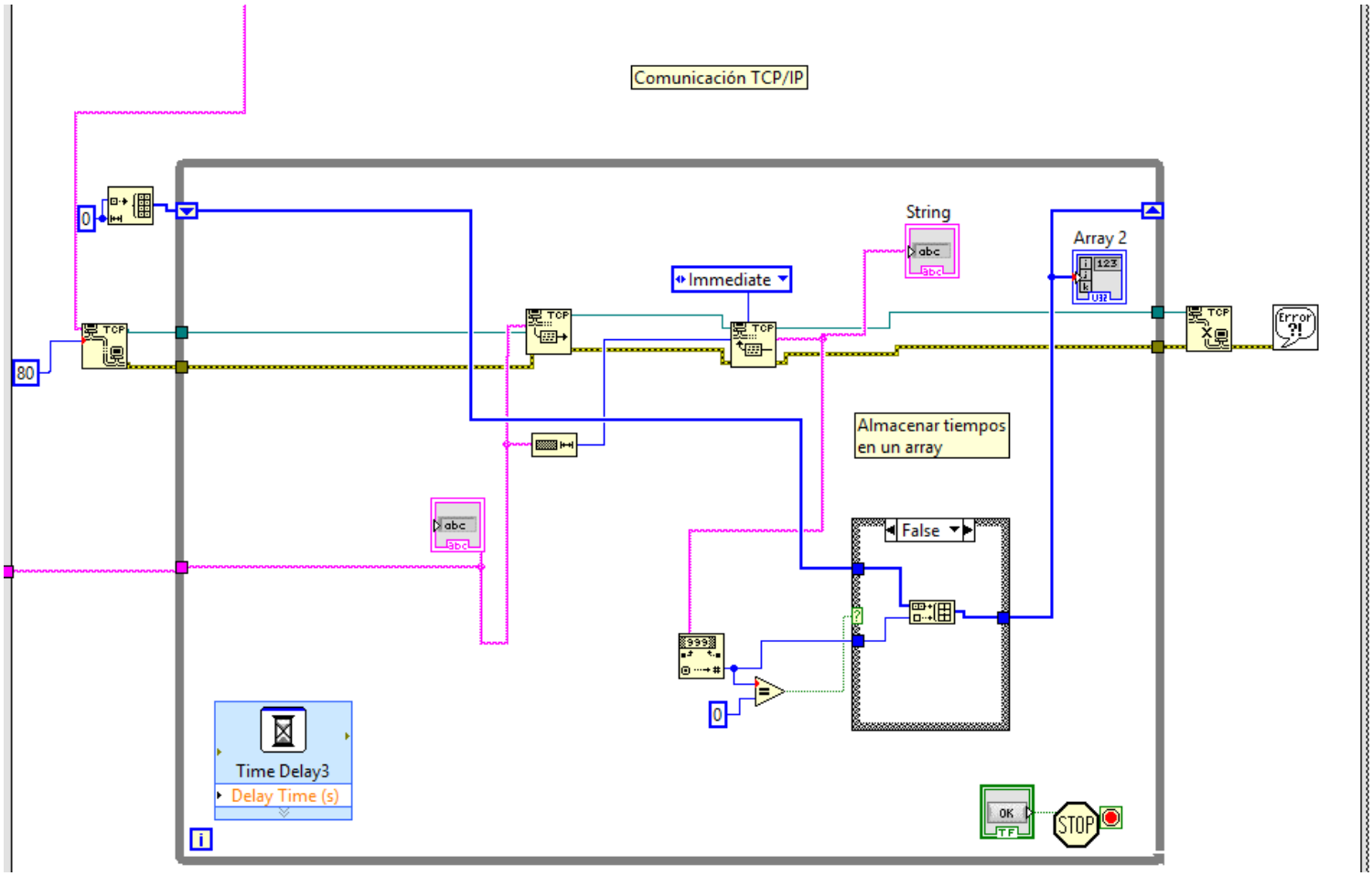

P1
U2
U1
R12
R9
R8
R1
C2
U3

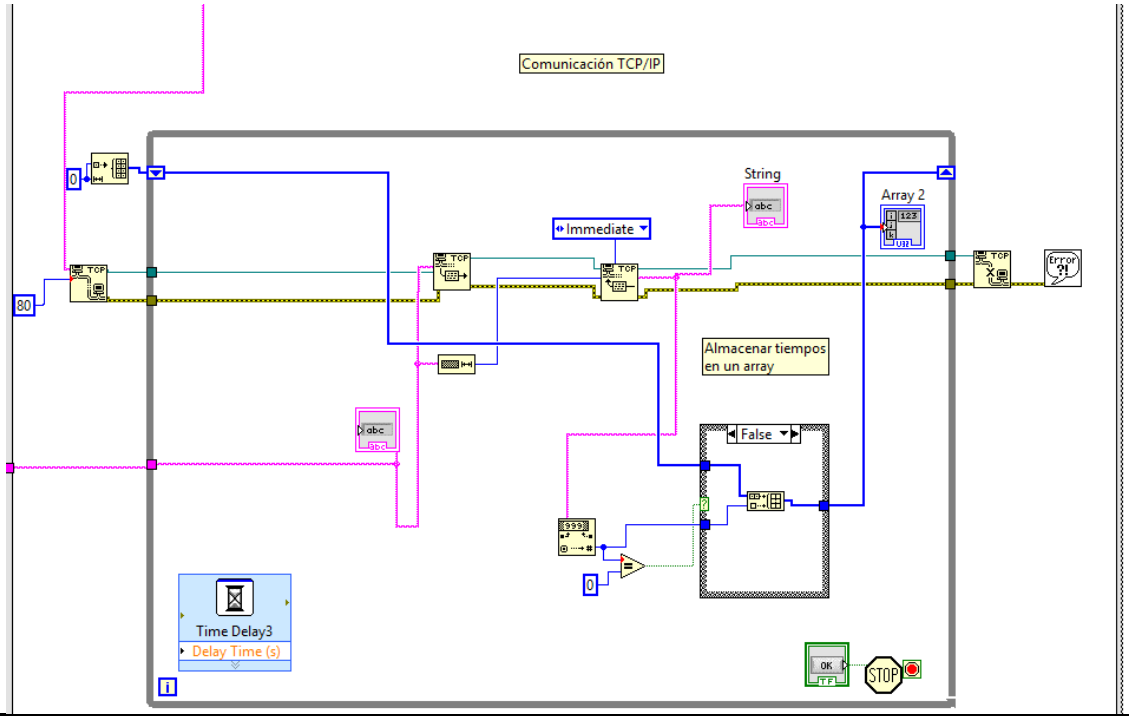


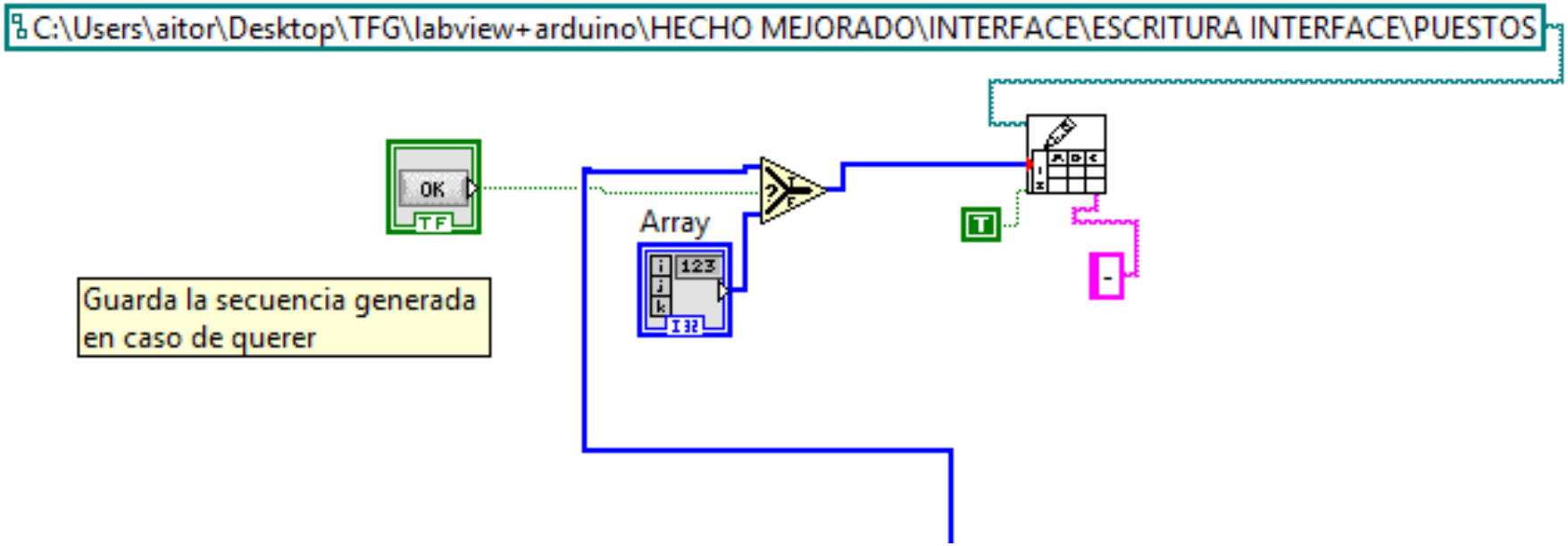
10.13 Programa LabView




N
Concatenate Strings
9
0
999
Convertimos el array en un string
OK
TF
En caso de escribir una secuencia ya creada, este VI lo hace cogiendola de un archivo .txt


Comunicación TCP/IP
String
Array 2
Immediate
Almacenar tiempos en un array
False
Time Delay3
Delay Time (s)
OK
STOP


Comunicación TCP/IP
String
Array 2
Immediate
Almacenar tiempos en un array
False
Time Delay3
Delay Time (s)
OK
STOP


C:\Users\aitor\Desktop\TFG\labview+arduino\HECHO MEJORADO\INTERFACE\ESCRITURA INTERFACE\PUESTOS
OK
TF
Array
Guarda la secuencia generada en caso de querer


GRUPO AZKOYEN
PUESTOS DE TRABAJO
7
IP
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27
ENVIAR
SECUENCIA
0
GUARDAR
ESCRIBIR
MEDIR TIEMPO
STOP
Array 2
0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

10.14 Programa Arduino

```
#include <SPI.h>          //incluir librería de comunicación SPI

#include <Ethernet.h>     //incluir librería de ethernet

#include <EEPROM.h>       //incluir librería EEPROM


byte mac[]={0xDE, 0xAD, 0xBE, 0xEF , 0xFE, 0xED}; //MAC estandar del shield
ethernet

IPAddress ip(169,254,183,255);   //IP utilizada en la empresa

EthernetServer server(80);       //puerto remoto para comunciarse


unsigned long tiempoInicio,tiempoFinal;    //variables para medir tiempo de ciclo


float tiempoTotal;                  //variable para medir el tiempo de ciclo


String  bufferString;          //buffer donde se almacena el string que viene de
LabView


int numDatos;                  //numero de datos leidos desde LabView


boolean disponible=0;          //vble. para entrar a la secuencia de LEDS


int noVolatil=0;               //VARIABLE PARA ALMACENAR EN MEMORIA NO VOLATIL


int noComun=0;                 //NO VOLATIL VBLE.
```

```
boolean primero=0;          //vble. para saber cual es el primer numero de la
secuencia


void setup() {


Serial.begin(9600);


Ethernet.begin(mac,ip);        //inicializa la librería ethernet y las configuraciones de
red

server.begin();


pinMode(7,OUTPUT);

pinMode(6,OUTPUT);     //CONTROL LEDS VERDES

pinMode(5,OUTPUT);


pinMode(4,OUTPUT);     //CONTROL ENCENDIDO ROJO Y VERDE ENABLE

pinMode(3,OUTPUT);


pinMode(46,OUTPUT);

pinMode(38 ,OUTPUT);   //CONTROL LEDS ROJOS

pinMode(40,OUTPUT);


for(int i=22;i<37;i++){

pinMode(i,INPUT);       // LECTURA SENSORES

}


}
```

```
void loop() {

 delay(10000);                    //esperar 10sg a ver si hay comunicacion o no

Serial.println("fin delay");


EthernetClient client=server.available();      //crea un cliente que se conecta a una IP y puerto

if(client) {                     //indica si el cliente esta preparado


 bufferString="";                //limpiamos el buffer para leer la secuencia

 while(client.connected()){      //mientras el ciente esté conectado


  if(client.available()>0){      //si el numero de bytes a leer es mayor a 0 (hay algo para leer)


   numDatos=client.available();  //cantidad de datos que hay para leer

   delay(20);                    //esperamos 20 milisegundos


   while(client.available()>0){  //mientras haya algo para leer


    bufferString += (char) client.read();  //leemos y se lo sumamos a lo leido anteriormente

   }



}
```

```
}

 client.stop();                    //desconecta el cliente del servidor

 disponible=1;                     //dejamos que entre en la secuencia de LEDS
adelante

 noVolatil=1;                      //dejamos que almacene la secuencia en la memoria
EEPROM


}

 //////////LO SIGUIENTE SE EJECUTA AL DESCONECTAR EL CLIENTE DEL
SERVIDOR///////////////

    unsigned long tablaTiempo;

    int toca;                      //PARA LEDS ROJOS saber cual toca encender

    int j=0,i=0;                   //VBLES. PASAR POR LAS TABLAS

    byte numSeparadosVolatil;      //para almacenar en la memoria la
longitud de la secuencia

    byte val1;                     //para almacenar datos y despues pasaros a la
EEPROM

    int direccion=0;               //direccion donde se almacenan los datos (a
partir de esta)
```

```
    int primerNumero;                    //primer numero de la secuencia

    int numSeparados=0;                  //cantidad de numeros separados que hay

    boolean contador=0;                  //contador para hacer por 10 o no el
numero

    byte suma=0;                         //variables donde se almacena la suma del
numero

    int parar=0;                         //variable que contiene '-' para parar

    int tablaInt[numDatos];              //tabla con los numeros de char a enteros
pasados

    byte tablaSuma[numDatos];            //tabla con los numeros separados

    byte contarojo=tablaSuma[numDatos];

    char tabla[numDatos];                //tabla con los numeros pero en tipo char

    char stringToChar[numDatos];         //variable donde se almacenara el
string del buffer en forma de char del tamaño de los datos leidos desde LabView

    bufferString.toCharArray(stringToChar,numDatos);   //convertir string a char de
tamaño de datos leidos desde LabView
```

```
    char *ptr=stringToChar;                    //puntero para recorrer la variable y
poder separar los caracteres


    for (int i=0; i<numDatos;i++){


      tabla[i]=(*ptr++);                       //separa cada numero en cada parte del
array


    }


    for (int i=0;i<numDatos;i++){


      tablaInt[i]=tabla[i]-'0';                //convertimos los numeros char a enteros


    }




    //NO NECESITO PUNTERO YA XQ TENGO UNA TABLA


    //CONCATENAR COMO LO HACIA EN ETHERNET ---> BUFFERNUEVO+=TABLA[I]


    for (int i=0;i<numDatos;i++){              //for para juntar los numeros (si son
11,12 etc hay que juntar el 1 con el 2 por ejemplo)


     if (tabla[i]==';'){
```

```
        NULL;                       //si hay ; no hacemos nada

    }

    else if (tabla[i]=='-'){

        tablaSuma[j]=0;             //se almacena un 0 para saber que se acaba la
secuencia
        parar=j;                    //parar=j para saber la cantidad de datos que hay
en la tabla
        j++;

    }

    else if (contador==0){

        suma=tablaInt[i]*10;        //si es la primera parte del numero
multiplicamos por 10
        contador=1;                 //el numero sera= parte1*10 + parte2

    }

    else if (contador==1) {

        suma=suma+tablaInt[i];      //si es la segunda parte sumamos el
numero y la guardamos en tablaSuma
```

```
        tablaSuma[j]=suma;
        j++;
        contador=0;
        numSeparados++;                    //variable para saber cuantos numero hay


      }


    }

/////PARA NO VOLATIL--> ALMACENAR LA TABLASUMA, NUMSEPARADOS///////

    if (noVolatil==1){                     //si ha habido comunicacion con el
ordenador
      noVolatil=0;                         //cambiamos vble. para no tener que
hacer esto otra vez

      EEPROM.write(direccion,numSeparados);          //se guarda en la
direccion la cantidad de numeros que hay

      for (j=0;j<(numSeparados);j++){
```

```
      EEPROM.write(direccion+1+j,tablaSuma[j]);          //se almacenan los
numeros a partir de la anterior direccion


    }


  }



    numSeparadosVolatil=EEPROM.read(direccion);          //se guarda en
numSeparadosVolatil la cantidad de numeros separados
     Serial.println("numSeparados=");
    Serial.println(numSeparados);
    Serial.println("numSeparadosVolatil=");              //mensajes por si queremos
visualizarlo por el puerto serie
    Serial.println(numSeparadosVolatil);



    if (disponible==0){                                  //si no ha habido comunicación con
el ordenador hay que leer desde la EEPROM la anterior secuencia guardada


     for (int i=0;i<numSeparadosVolatil;i++){
      val1=EEPROM.read(direccion+i+1);                   //se lee lo que hay en la
EEPROM
      tablaSuma[i]=val1;                                 //y se pone en a tablaSuma
      Serial.println(tablaSuma[i]);
     }
```

```
      disponible=1;                          // se cambia la vble. para que no
vuelva a entrar aqui

      parar=numSeparadosVolatil;             //almacena la cantidad de numeros
que tiene la secuencia

    }


    for (j=0;j<(numSeparadosVolatil);j++){          //un for para visualizar
por puerto serie los datos


       val1= EEPROM.read(direccion+j+1);            //le pasamos el valorque
esta en EEPROM a esta vble.


       Serial.print("En la direccion ");

       Serial.print(direccion+j+1);

       Serial.print("se encuentra la información ");      //para visualizar si se ha
guardado bien

       Serial.print(val1,DEC);

        Serial.println();



    }



  if (disponible==1){                   //si esta ya todo listo (si se ha escrito dsd
EEPROM o se a almacenado) para empezar el programa principal


   j=0;
```

```
int contador=tablaSuma[j];                    //PARA EVITAR REBOTE

Serial.println("empiezaa");

primerNumero=tablaSuma[j];                    //indicamos cual es el primer numero de la secuencia

Serial.println(tablaSuma[j]);


while(1){                                     //BUCLE INFINITO para ejecutar el programa


 EthernetClient client=server.available();    //ABRIMOS COMUNICACIÓN CON EL ORDENADOR


 if (j==parar) {                              //SI LLEGA AL FINAL DE LA SECUENCIA


  j=0;                                        //EMPEZAMOS LA TABLA DE NUEVO

  i=0;


  tiempoFinal=millis();                       //medimos el tiempo final de la secuencia

  tiempoTotal=(tiempoFinal-tiempoInicio);     //la diferencia de tiempos será el tiempo que le cuesta un ciclo

  tablaTiempo=(tiempoTotal);


  tiempoTotal=tiempoTotal/60000;              //ESTE ES EL TIEMPO TOTAL DEL CICLO

  Serial.println(tiempoTotal);


  primero=0;

  primerNumero=tablaSuma[j];                  //REINICIAMOS LAS VBLES.

  contador=tablaSuma[j];
```

```
 if(client) {                    //indica si el cliente esta preparado

delay(20);
server.println(tablaTiempo);

 delay(100);



 }
  delay(100);
}
```

```
 if (((digitalRead(26)==HIGH))||(primerNumero==0)) {    //SI EL SENSOR DETECTA O ES EL PRIMER NUMERO
    delay(100);

    Serial.println("estas en el cero");

   if (primerNumero==tablaSuma[j]){              //SI ES EL PRIMER NUMERO

    digitalWrite(7,LOW);
    digitalWrite(6,LOW);
```

```
        digitalWrite(5,LOW);                    //ENCENDEMOS AMARILLO (ROJO+VERDE)
        digitalWrite(4,LOW);
        digitalWrite(3,LOW);
        digitalWrite(46,LOW);
        digitalWrite(38,LOW);
        digitalWrite(40,LOW);

        primerNumero=35;                        //para que no vuelva a entrar aqui(se le da
cualquier valor mayor de 30)

        Serial.println(tablaSuma[j]);

        while(digitalRead(26)==LOW){
          NULL;                                 //se queda aqui hasta que detecta algo

        }

      }

      delay(50);
      if (contador==0){                         //si contador es 0
        while(digitalRead(26)==HIGH){           //mientras el sensor de la posicion 0
esta detectando algo
          NULL;                                 //espera en el bucle
        }                                       //PARA EVITAR REBOTES ESTE IF
      j++;                                      //avanzamos en las tablas de rojo y verde
```

```
      i++;

      contador=tablaSuma[j];              //le damos el siguiente valor de la secuencia
al contador


      byte valor=tablaSuma[j];            //pasamos a esta vble. el siguiente LED a
encender

      LEDD(valor);                        //FUNCION ENCENDER LED VERDE


      }


      if ((contarojo!=0)&&(tablaSuma[j-1]!=0)) {  //se enciende el rojo si no es ni el
LED 0 ni el anterior al que le tocaba encenderse en verde

       toca=0;                            //vble para encender LED rojo


       ROJO(toca);                        //FUNCION ENCENDER LED ROJO

       contarojo=tablaSuma[i];            //avanzamos por la tabla de rojo

      }

      if (primero==0){                    //PARA MEDIR TIEMPOS


      primero=1;

      tiempoInicio=millis();

    }


    }


    /////////EL RESTO SON IGUALES QUE ESTE ANTERIOR, POR LO QUE NO SE
COMENARA///////////////
```

```
    else if (((digitalRead(22)==HIGH))||(primerNumero==1)){//SI EL SENSOR DETECTA
O ES EL PRIMER NUMERO

      delay(50);


       if (primerNumero==tablaSuma[j]){ //SI ES EL PRIMER NUMERO


        primerNumero=35;//para que no vuelva a entrar aqui(se le da cualquier valor
mayor de 30)

        digitalWrite(7,HIGH);

        digitalWrite(6,LOW);

        digitalWrite(5,LOW);      //ENCENDEMOS AMARILLO (ROJO+VERDE)

        digitalWrite(4,LOW);

        digitalWrite(3,LOW);

        digitalWrite(46,HIGH);

        digitalWrite(38,LOW);

        digitalWrite(40,LOW);

      Serial.println(tablaSuma[j]);


      while(digitalRead(22)==LOW){ //se queda aqui hasta que detecta algo

         NULL;

       }



      }


      Serial.println("no puede ser");
```

```
  delay(50);
 if (contador==1){
  while(digitalRead(22)==HIGH){
   NULL;
  }
j++;
i++;
contador=tablaSuma[j];

byte valor=tablaSuma[j];
LEDD(valor);
 }

if ((contarojo!=1)&&(tablaSuma[j-1]!=1)) {
 toca=1;

 ROJO(toca);                    //FUNCION ENCENDER LED ROJO
 contarojo=tablaSuma[i];
}

if (primero==0){

primero=1;
tiempoInicio=millis();
```

```
    }



    }

    else if (((digitalRead(27)==HIGH))||(primerNumero==2)){//SI EL SENSOR DETECTA
O ES EL PRIMER NUMERO
     delay(50);

      if (primerNumero==tablaSuma[j]){ //SI ES EL PRIMER NUMERO

        primerNumero=35;//para que no vuelva a entrar aqui(se le da cualquier valor
mayor de 30)
     digitalWrite(7,LOW);
     digitalWrite(6,HIGH);
     digitalWrite(5,LOW);      //ENCENDEMOS AMARILLO (ROJO+VERDE)
     digitalWrite(4,LOW);
     digitalWrite(3,LOW);
     digitalWrite(46,LOW);
     digitalWrite(38,HIGH);
     digitalWrite(40,LOW);
     Serial.println(tablaSuma[j]);
     while(digitalRead(27)==LOW){
        NULL; //se queda aqui hasta que detecta algo
       }
```

```
 }



   delay(50);
 if (contador==2){
  while(digitalRead(27)==HIGH){
     NULL;
    }
 j++;
 i++;
contador=tablaSuma[j];

 byte valor=tablaSuma[j];
 LEDD(valor);
 }

if ((contarojo!=2)&&(tablaSuma[j-1]!=2)) {
   toca=2;

   ROJO(toca);                    //FUNCION ENCENDER LED ROJO
  contarojo=tablaSuma[i];
 }



 if (primero==0){
```

```
      primero=1;
      tiempoInicio=millis();
    }



    }

   else if (((digitalRead(23)==HIGH))||(primerNumero==3)) {    //SI EL SENSOR DETECTA O ES EL PRIMER NUMERO
      delay(50);

     if (primerNumero==tablaSuma[j]){ //SI ES EL PRIMER NUMERO

       digitalWrite(7,HIGH);
       digitalWrite(6,HIGH);
       digitalWrite(5,LOW);      //ENCENDEMOS AMARILLO (ROJO+VERDE)
       digitalWrite(4,LOW);
       digitalWrite(3,LOW);
       digitalWrite(46,HIGH);
       digitalWrite(38,HIGH);
       digitalWrite(40,LOW);

       primerNumero=35;              //para que no vuelva a entrar aqui(se le da cualquier valor mayor de 30)

       Serial.println(tablaSuma[j]);
```

```
    while(digitalRead(23)==LOW){ //se queda aqui hasta que detecta algo
      NULL;

    }

  }


   Serial.println("esta aki");        //IGUAL QUE EL ANTERIOR
  delay(50);
  if (contador==3){
    while(digitalRead(23)==HIGH){
      NULL;
    }
  j++;
  i++;
  contador=tablaSuma[j];

  byte valor=tablaSuma[j];
  LEDD(valor);
  }
if ((contarojo!=3)&&(tablaSuma[j-1]!=3)) {
   toca=3;

   ROJO(toca);                    //FUNCION ENCENDER LED ROJO
   contarojo=tablaSuma[i];
```

```
    }
    if (primero==0){

    primero=1;
    tiempoInicio=millis();
  }

  }

  else if (( (digitalRead(28)==HIGH ))||(primerNumero==4)){//SI EL SENSOR DETECTA O ES EL PRIMER NUMERO

    delay(50);

     if (primerNumero==tablaSuma[j]){ //SI ES EL PRIMER NUMERO

      primerNumero=35;//para que no vuelva a entrar aqui(se le da cualquier valor mayor de 30)
      digitalWrite(7,LOW);
      digitalWrite(6,LOW);
      digitalWrite(5,HIGH);      //ENCENDEMOS AMARILLO (ROJO+VERDE)
      digitalWrite(4,LOW);
      digitalWrite(3,LOW);
      digitalWrite(46,LOW);
      digitalWrite(38,LOW);
      digitalWrite(40,HIGH);
    Serial.println(tablaSuma[j]);
```

```
 while(digitalRead(28)==LOW){ //se queda aqui hasta que detecta algo
    NULL;
  }


}


delay(50);
if (contador==4){
 while(digitalRead(28)==HIGH){
    NULL;
  }
j++;
i++;
contador=tablaSuma[j];
                    //IGUAL QUE EL ANTERIOR
byte valor=tablaSuma[j];
LEDD(valor);
 }


if ((contarojo!=4)&&(tablaSuma[j-1]!=4)) {
  toca=4;


  ROJO(toca);                  //FUNCION ENCENDER LED ROJO
 contarojo=tablaSuma[i];
```

```
    }



    if (primero==0){

    primero=1;

    tiempoInicio=millis();
  }



  }

  else if (((digitalRead(24)==HIGH))||(primerNumero==5)){//SI EL SENSOR DETECTA
O ES EL PRIMER NUMERO

    delay(50);

     if (primerNumero==tablaSuma[j]){ //SI ES EL PRIMER NUMERO

      primerNumero=35;//para que no vuelva a entrar aqui(se le da cualquier valor
mayor de 30)
      digitalWrite(7,HIGH);
      digitalWrite(6,LOW);
      digitalWrite(5,HIGH);     //ENCENDEMOS AMARILLO (ROJO+VERDE)
      digitalWrite(4,LOW);
      digitalWrite(3,LOW);
      digitalWrite(46,HIGH);
```

```
 digitalWrite(38,LOW);
 digitalWrite(40,HIGH);
Serial.println(tablaSuma[j]);
 while(digitalRead(24)==LOW){
   NULL;
  }


 }                          //IGUAL QUE EL ANTERIOR


delay(50);
if (contador==5){
 while(digitalRead(24)==HIGH){ //se queda aqui hasta que detecta algo
   NULL;
  }
j++;
i++;
contador=tablaSuma[j];

byte valor=tablaSuma[j];
LEDD(valor);
 }
if ((contarojo!=5)&&(tablaSuma[j-1]!=5)) {
 toca=5;
```

```
        ROJO(toca);                    //FUNCION ENCENDER LED ROJO
        contarojo=tablaSuma[i];
       }



       if (primero==0){

       primero=1;
       tiempoInicio=millis();
     }



     }

    else if (((digitalRead(29)==HIGH))||(primerNumero==6)){//SI EL SENSOR DETECTA
O ES EL PRIMER NUMERO

      delay(50);

       if (primerNumero==tablaSuma[j]){ //SI ES EL PRIMER NUMERO

        primerNumero=35;//para que no vuelva a entrar aqui(se le da cualquier valor
mayor de 30)
        digitalWrite(7,LOW);
        digitalWrite(6,HIGH);
        digitalWrite(5,HIGH);      //ENCENDEMOS AMARILLO (ROJO+VERDE)
        digitalWrite(4,LOW);
```

```
  digitalWrite(3,LOW);
  digitalWrite(46,LOW);
  digitalWrite(38,HIGH);
  digitalWrite(40,HIGH);
 Serial.println(tablaSuma[j]);
  while(digitalRead(29)==LOW){ //se queda aqui hasta que detecta algo
     NULL;
    }



 }
                              //IGUAL QUE EL ANTERIOR


delay(50);
if (contador==6){
 while(digitalRead(29)==HIGH){
     NULL;
    }
j++;
i++;
contador=tablaSuma[j];

byte valor=tablaSuma[j];
LEDD(valor);
 }
```

```
    if ((contarojo!=6)&&(tablaSuma[j-1]!=6)) {
      toca=6;

      ROJO(toca);                    //FUNCION ENCENDER LED ROJO
      contarojo=tablaSuma[i];
    }


    if (primero==0){

    primero=1;
    tiempoInicio=millis();
  }


  }

  //ESTA LOW XK AUN NO TENGO SENSOR Y SINO ESTA BENGA DETECTAR

  else if (( (digitalRead(25)==LOW))||(primerNumero==7)){//SI EL SENSOR DETECTA O ES EL PRIMER NUMERO

    delay(50);

     if (primerNumero==tablaSuma[j]){ //SI ES EL PRIMER NUMERO
```

```
        primerNumero=35;//para que no vuelva a entrar aqui(se le da cualquier valor mayor de 30)
        digitalWrite(7,HIGH);
        digitalWrite(6,HIGH);
        digitalWrite(5,HIGH);      //ENCENDEMOS AMARILLO (ROJO+VERDE)
        digitalWrite(4,LOW);
        digitalWrite(3,LOW);
        digitalWrite(46,HIGH);
        digitalWrite(38,HIGH);
        digitalWrite(40,HIGH);
      Serial.println(tablaSuma[j]);
       while(digitalRead(25)==LOW){ //se queda aqui hasta que detecta algo
          NULL;
        }

                                    //IGUAL QUE EL ANTERIOR
      }


     delay(50);
     if (contador==7){
       while(digitalRead(25)==HIGH){
          NULL;                     //para encender el siguiente led esperamos a sakar la mano
        }                           //flanco de bajada
     j++;
     i++;
```

```
    contador=tablaSuma[j];


    byte valor=tablaSuma[j];
    LEDD(valor);
     }


    if ((contarojo!=7)&&(tablaSuma[j-1]!=7)) {
      toca=7;


      ROJO(toca);                    //FUNCION ENCENDER LED ROJO
     contarojo=tablaSuma[i];
    }


    if (primero==0){


    primero=1;
    tiempoInicio=millis();
  }



  }


  else NULL;                    //SI NO ES NINGUNO NO SE HACE NADA (SE PODRÍA
PONER UN ERROR POR EL PUERTO SERIE POR EJEMPLO)


 }
```

```
  }



}



void LEDD(byte valor)                    //funcion para encender el siguiente LED de la secuencia VERDE
{
  Serial.println("entra en la función");
 Serial.println(valor);

if (valor==0){            //si le toca al 0

   digitalWrite(7,LOW);
   digitalWrite(6,LOW);
   digitalWrite(5,LOW);      //VERDE 0
   digitalWrite(4,LOW);
   digitalWrite(3,HIGH);



}
```

```
else if (valor==1){       //si le toca al 1

  digitalWrite(7,HIGH);
  digitalWrite(6,LOW);
  digitalWrite(5,LOW);      //VERDE 1
  digitalWrite(4,LOW);
  digitalWrite(3,HIGH);


}

else if (valor==2){       //si le toca al 2

  digitalWrite(7,LOW);
  digitalWrite(6,HIGH);
  digitalWrite(5,LOW);      //VERDE 2
  digitalWrite(4,LOW);
  digitalWrite(3,HIGH);


}

else if (valor==3){       //si le toca al 3

  digitalWrite(7,HIGH);
```

```
  digitalWrite(6,HIGH);
  digitalWrite(5,LOW);      //VERDE 3
  digitalWrite(4,LOW);
  digitalWrite(3,HIGH);



}

else if (valor==4){       //si le toca al 4

  digitalWrite(7,LOW);
  digitalWrite(6,LOW);
  digitalWrite(5,HIGH);   //VERDE 4
  digitalWrite(4,LOW);
  digitalWrite(3,HIGH);



}

 else if (valor==5){      //si le toca al 5

   digitalWrite(7,HIGH);
   digitalWrite(6,LOW);
   digitalWrite(5,HIGH);     //VERDE 5
   digitalWrite(4,LOW);
   digitalWrite(3,HIGH);
```

```
}

else if (valor==6){       //si le toca al 6

    digitalWrite(7,LOW);
    digitalWrite(6,HIGH);
    digitalWrite(5,HIGH);     //VERDE 6
    digitalWrite(4,LOW);
    digitalWrite(3,HIGH);


}

else if (valor==7){       //si le toca al 7

    digitalWrite(7,HIGH);
    digitalWrite(6,HIGH);
    digitalWrite(5,HIGH);
    digitalWrite(4,LOW);   //VERDE 7
    digitalWrite(3,HIGH);


}
return;                   //volvemos de la función al programa principal
```

```
}



void ROJO(int toca){              //FUNCION PARA ENCENDER LED ROJO

if (toca==0){                 //si le toca al 0


   digitalWrite(46,LOW);
   digitalWrite(38,LOW);
   digitalWrite(40,LOW);      //ROJO 0
   digitalWrite(3,LOW);

  delay(500);
  digitalWrite(3,HIGH);
}

 else if (toca==1){           //si le toca al 1


   digitalWrite(46,HIGH);
   digitalWrite(38,LOW);
   digitalWrite(40,LOW);      //ROJO 1
   digitalWrite(3,LOW);
```

```
  delay(500);
  digitalWrite(3,HIGH);
}


 else if (toca==2){          //si le toca al 2


   digitalWrite(46,LOW);
   digitalWrite(38,HIGH);
   digitalWrite(40,LOW);     //ROJO 2
   digitalWrite(3,LOW);
  delay(500);
  digitalWrite(3,HIGH);
}


 else if (toca==3){          //si le toca al 3


   digitalWrite(46,HIGH);
   digitalWrite(38,HIGH);
   digitalWrite(40,LOW);     //ROJO 3
   digitalWrite(3,LOW);
  delay(500);
  digitalWrite(3,HIGH);
```

```
}


 else if (toca==4){          //si le toca al 4


   digitalWrite(46,LOW);
   digitalWrite(38,LOW);
   digitalWrite(40,HIGH);     //ROJO 4
   digitalWrite(3,LOW);
  delay(500);
  digitalWrite(3,HIGH);
}


 else if (toca==5){          //si le toca al 5


   digitalWrite(46,HIGH);
   digitalWrite(38,LOW);
   digitalWrite(40,HIGH);     //ROJO 5
   digitalWrite(3,LOW);
  delay(500);
  digitalWrite(3,HIGH);
}
```

```
 else if (toca==6){          //si le toca al 6


   digitalWrite(46,LOW);
   digitalWrite(38,HIGH);
   digitalWrite(40,HIGH);     //ROJO 6
   digitalWrite(3,LOW);
  delay(500);
  digitalWrite(3,HIGH);
}


 else if (toca==7){          //si le toca al 7


   digitalWrite(46,HIGH);
   digitalWrite(38,HIGH);
   digitalWrite(40,HIGH);     //ROJO 7
   digitalWrite(3,LOW);
   delay(500);
  digitalWrite(3,HIGH);


}
```

```
return;                    //volvemos de la función al programa principal


}
```

///FINAL PROGRAMA///

11. BIBLIOGRAFÍA

- http://www.ni.com/es-es/shop/labview.html
- https://www.arduino.cc/
- www.prometec.net/indice-tutoriales/
- http://es.rs-online.com/web/
- http://www.alldatasheet.com/
- https://www.arduino.cc/en/Guide/ArduinoEthernetShield
- Circuitos electrónicos : análisis, diseño y simulación / N. R. Malik ; traducción Miguel Angel Pérez García, M Antonia Menéndez Ordas, Cecilio Blanco Viejo ; revisión técnica Juan Meneses Chaus... [et al.]. (1996)
  Malik, N. R. (Norbert R.) (1936-)
  Editorial: Madrid [etc.] : Prentice Hall, [1996]
  ISBN: 84-89660-03-4 --
- Fundamentos de sistemas digitales / Thomas L. Floyd ; traducción [de la 9 ed. en inglés] Vuelapluma ; revisión técnica Eduardo Barrera López de Turiso. (2006)
  Floyd, Thomas L.
  Editorial: Madrid [etc.] : Pearson Prentice Hall, [2006]
  ISBN: 978-84-8322-085-6 --