Facultad de Ingeniería - UBA
Departamento de Electrónica
86.07/66.09 - Laboratorio de Microprocesadores

Escaneo y Decodificación de Códigos QR

Profesor:			Ing. Guillermo Campiglio								
Cuatrimestre/Año:			2º/2016								
Turno de las clases prácticas			Miércoles								
Jefe de trabajos prácticos:			Ing. Ricardo Arias								
Autores			Seguimiento del proyecto								
Nombre	Apellido	Padrón									
Lautaro	Estienne	96671									
Tomás	Rebollo	90986									

Observaciones:

.....
.....
.....
.....

Fecha de aprobación				Firma J.T.P

Coloquio	
Nota final	
Firma profesor	

Resumen

En el presente trabajo, se construyó y programó un dispositivo capaz de escanear un código QR impreso en una hoja de papel, decodificarlo y enviar el mensaje contenido en él a un dispositivo celular. A su vez, el mismo fue diseñado para que el control de encendido y apagado del escaneo y el envío del mensaje sean controlados por *Bluetooth*.

1. Breve Introducción

Un código QR (*Quick Response code*, "código de respuesta rápida") es un módulo para almacenar información en una matriz de puntos, que se utiliza hoy en día en distintas formas:

- Registro de repuestos en el área de la fabricación de vehículos (primera implementación de estos códigos)
- Administración de inventarios en una gran variedad de industrias
- Almacenamiento de direcciones de páginas WEB en publicidades y anuncios
- Venta de productos y marketing de fidelización (como por ejemplo, cupones o descuentos)
- Etiquetado de productos en la tienda (como un código de barras)

La información que contiene un código QR puede incluír caracteres ACII, Kanji, o combinaciones de ambos, y además de los caracteres codificados, la matriz contiene una serie de puntos que dan información acerca de cómo está almacenado el mensaje en el código.

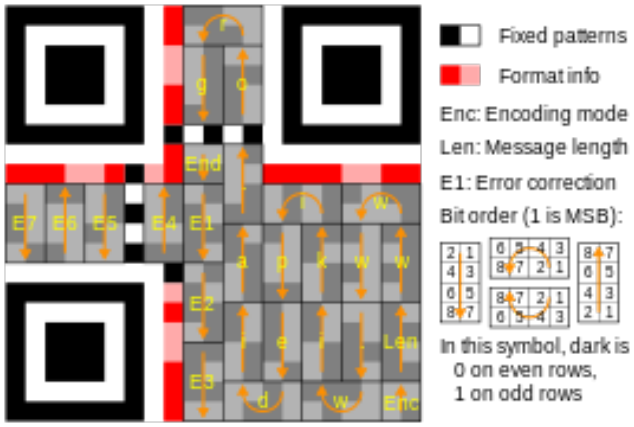


Figura 1: Disposición de la información en la matriz QR

La decodificación del código consiste básicamente en dos pasos: leer y decodificar la información auxiliar de código para entender cómo está dispuesta la información, y luego leer la matriz en la forma en que está dispuesto el código (respetando la información obtenida anteriormente). En la figura 1 se muestra un código QR con el mensaje "www.wikipedia.org", y en el mismo puede observarse qué parte del la matriz contiene la información del mensaje y qué parte contiene la información auxiliar (*format info*).

2. Objetivos Propuestos

La idea original del proyecto fue armar un dispositivo que permitiera escanear un código QR impreso en una hoja de papel, almacenar el mensaje contenido en él en la memoria temporal del microcontrolador, y finalmente, enviarlo a un dispositivo móvil por medio de una conexión bluetooth. También se decidió que el control de todo el proceso sea a través del mismo dispositivo.

3. Implementacion

Una vez propuestos los objetivos, se pasó a pensar en la forma de implementar el scanner. En primer lugar, el dispositivo debería contener un microcontrolador que se ocupe de sincronizar el proceso en su totalidad. Se sabe además que se debería constar de un sensor de luz para realizar el escaneo y a su vez, un motor que permita mover dicho sensor por la hoja. Por otro lado, la misma hoja debería ser desplazada por otro motor independiente del anterior. Por último, es necesario un circuito que se encargue de la transmisión por bluetooth (para lo cual ya existen módulos prefabricados).

Luego de algunas variantes se llegó a la versión final del dispositivo, mostrada en la figura 2:

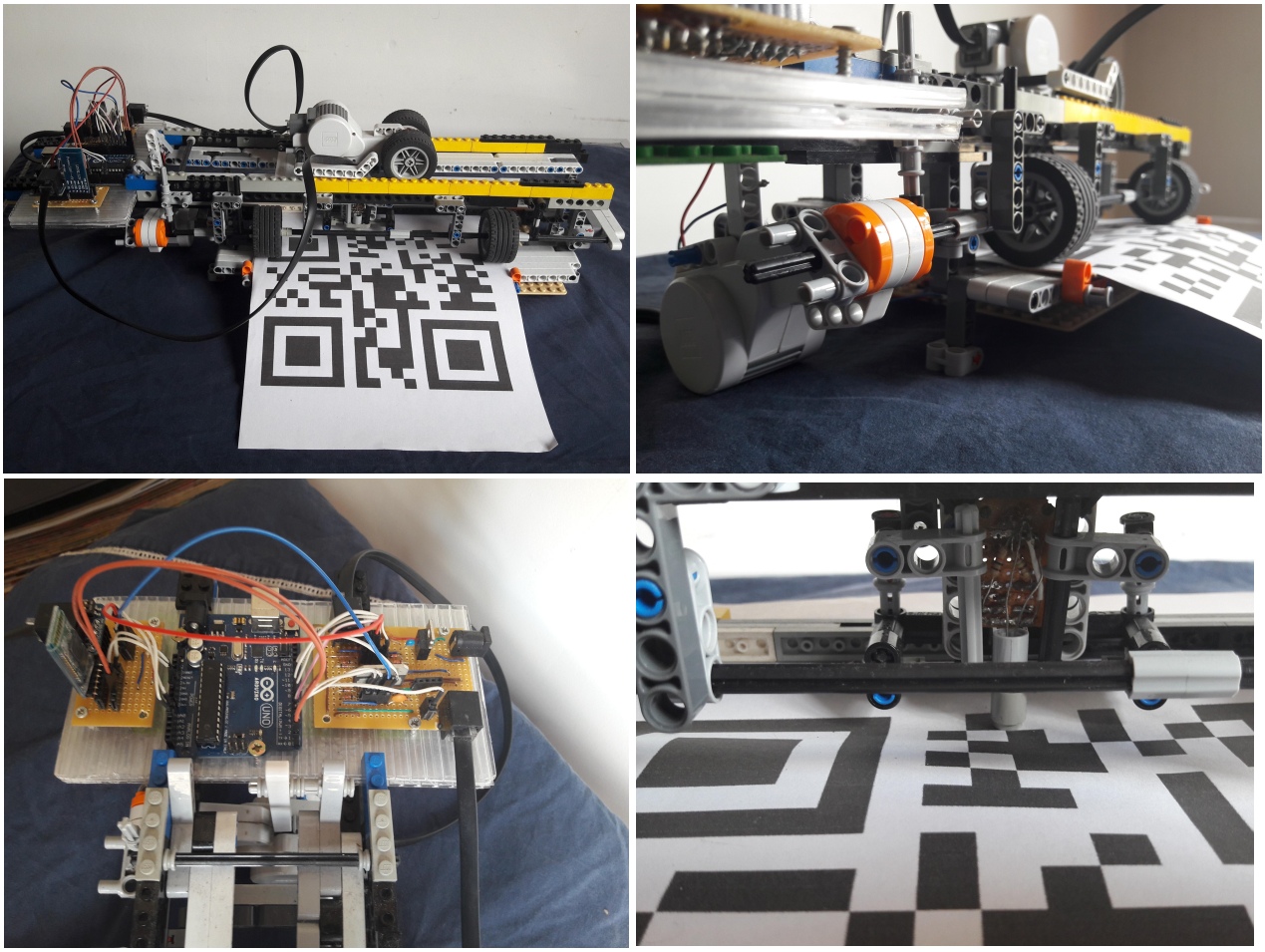


Figura 2: Fotografía del proyecto terminado.

Puede observase el código impreso en la hoja y los motores que controlan el rodillo que arrastra la hoja, y el carrito que mueve el sensor. A medida que la hoja avanza, el carrito hace recorrer al sensor por cada fila de la matriz. En cada barrido, se detectan los cambios de color y, con el dato de la posición del carrito, se calculan cuántos bits se leyeron de cada color. Como la matriz es un conjunto de puntos blancos y negros, entonces puede representarse como una serie de unos y ceros y almacenarse en la memoria temporal del micro. Una vez leído el código, el programa cargado en el microcontrolador decodifica el mensaje, almacenando en otra parte de memoria únicamente la sucesión de caracteres ASCII que componen el mensaje del código. Finalmente, el mensaje es transmitido al celular gracias al módulo bluetooth que se puede visualizar en la misma figura que antes.

Una vez pensada la forma en que el dispositivo trabajaría, se realizó el diagrama en bloques del circuito, el cual se muestra en la figura 3. Esto sirvió para definir los modelos de los materiales necesarios para el proyecto y posteriormente, la modularización del trabajo. En la siguiente sección se describe la lista de materiales y componentes utilizados en el circuito y en el resto del dispositivo.

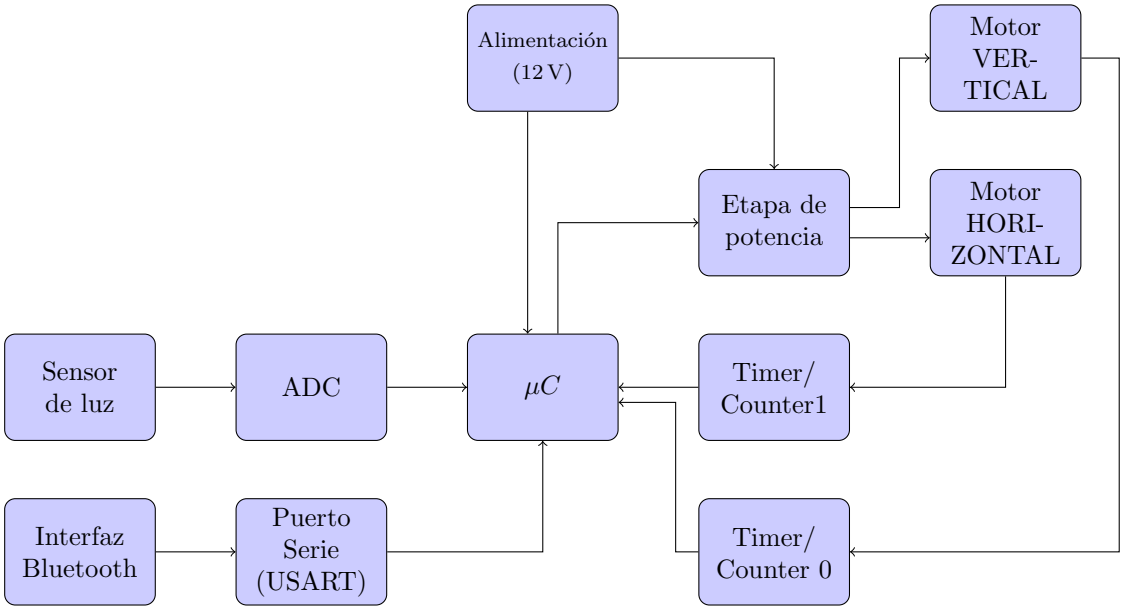


Figura 3: Diagrama en bloques del circuito completo.

4. Descripción del Hardware

El circuito contiene una placa Arduino UNO, la cual utiliza un microcontrolador Atmega328P, que fue el implementado en este trabajo. En principio, la elección de este microcontrolador se debe fundamentalmente a la cantidad de memoria Flash y SRAM disponible, y esto es consecuencia de la gran dimensión de código que se necesitó para programar la decodificación del QR y la memoria necesaria para almacenar la matriz. Sin embargo, al finalizar el trabajo, puede comprobarse que si fuera por un tema de memoria, otros microcontroladores como el Atmega168 o el Atmega328 también habrían servido. Ahora bien, en el trabajo se utilizó la instrucción `jmp` para salto incondicional (debido nuevamente a la extensión del código) y esa instrucción sólo está permitida en el Atmega328P o en el Atmega168PA.

Por otro lado, se tiene el sensor de luz. El mismo fue construído específicamente para el trabajo y consiste en un circuito como el que se muestra en el esquemático del apéndice. En éste, se muestra un fotoresistor de Cadmio que varía su valor con la luz. De esta forma se obtiene una señal analogica que entra a uno de los puertos del micro y se

determina a partir de ella si el color que se está leyendo es blanco o negro. Para que el contraste entre un color y el otro sea mayor, se utilizó una tira de LEDs del otro lado del papel, de manera que llegue más luz a este último. Esta parte del trabajo fue, sin duda, la más crítica, ya que se debió preparar al equipo para que el escaneo soporte la resolución del código. Luego de varias pruebas se logró un resultado satisfactorio, con lo cual se concluyó que este sensor de fácil implementación sería suficiente para cubrir los objetivos del trabajo práctico.

Continuando con el circuito, se llega a la etapa de potencia utilizada para controlar los dos servo motores que controlan el escaneo. Estos motores funcionan correctamente con una tensión de 9 V, por lo que fue necesario adaptar la salida del microcontrolador con un puente-H doble (ya que se tienen dos motores). Para ello se eligió el integrado L298, uno de los más comunes para esta tarea. El circuito de esta etapa no comprende demasiadas complicaciones y se muestra en el apéndice del informe. Sí es posible agregar, sin embargo, que estos motores se comunican mendiante 6 conectores, descritos a continuación. Los primeros dos son la salida del puente-H: el motor avanza al aplicarle tensión positiva y retrocede con una tensión negativa; cuando no hay tensión, el motor se encuetra quieto. Existen por otro lado, otros dos pines destinados a la tensión de referencia del motor (VCC y GND). Finalmente, el motor cuenta con un encoder de dos terminales, el cual permite medir la velocidad con la que se mueve y la dirección en la que avanza. Para el trabajo sólo fue necesario uno de ellos, ya que sólo se midió la posición.

Por último, la interfaz del bluetooth no presenta mayores complicaciones, ya que el módulo utilizado (el HC-05 configurable como maestro-esclavo) tiene dos pines para la comunicación por puerto serie, y otros dos para la alimentación.

La construcción del dispositivo que permite sostener los motores, el sensor y el resto del circuito fue realizada mediante el equipo Mindstorms NXT 2.0 desarrollado por la empresa LEGO en 2006. El mismo viene equipado con los motores mencionados anteriormente y con todas las piezas necesarias para el armado del proyecto, excluyendo por supuesto, los componentes del circuito. La mayor parte de la información del funcionamiento de los motores fue obtenida de Wikipedia (ver referencia [6]) y de los diversos tutoriales encontrados en la web y dirigidos por personas que ya descubrieron el funcionamiento interno de los componentes.

5. Descripción del Software

La figura **??** muestra el diagrama de flujo del programa. Para una explicación más detallada, el código se encuentra dividido en 7 grandes bloques, a saber:

- **Inicialización y configuraciones:** Dentro de este módulo se realizan todas las inicializaciones de variables y configuraciones a utilizar. Se utilizaron flags de estado y registros redefinidos con otros nombres para llevar más fácilmente el seguimiento del código. Las configuraciones realizadas fueron:
  - Configuración de puertos como entradas y salidas.
  - Configuración del ADC interno del microcontrolador.
  - Configuración de timers como contadores.
  - Configuración de USART como receptor de datos (el envío desde el microcontrolador será inicializado solo al momento en que se requiera enviar datos).

  Una vez finalizadas las configuraciones, se realiza la primera lectura del ADC para saber en donde se encuentra actualmente el sensor.

- **Recepción Bluetooth:** El programa quedará esperando la recepción de datos para iniciar el proceso de lectura. Se desarrolló una aplicación móvil que envía el caracter ASCII '1' para indicar el encendido de los motores, mientras que para desplazar la hoja verticalmente, utiliza '+' y '-'.
  Una vez recibido el caracter '1', se llama a la rutina "leer_columna"la cual se encarga de desplazar el motor horizontal para leer una línea del QR.

- **Movimiento de Motores:** El dispositivo cuenta con dos motores, uno para realizar el barrido horizontal y leer cada columna de cuadrados, y otro para desplazar la hoja a la siguiente columna. El motor horizontal se maneja seteando los pines 6, 7 del puerto D y 0 del puerto B para retroceso o avance, y un bit de stop general

respectivamente. El motor vertical se maneja con los pines 2, 3 del puerto D, y 1 del puerto B. Seteando o limpiando estos bits es como se manejaron los puentes H que terminaron controlando el desplazamiento de motores. Cada motor a su vez, enviaba al microcontrolador una señal de clock mientras estuviera funcionando, para llevar un control de su desplazamiento. Esto se contabilizó utilizando los timers para lograr precisión en la lectura.

- **Conteo de eventos:** Los enconders de los motores fueron conectados a los pines 4 y 5 del puerto D, utilizados como contadores. Los timers 0 y 1 fueron configurados en modo CTC (clear timer on compare), con clock externo en T0 y T1 respectivamente. Ambos conteos se compararon contra dos umbrales para tener dimensión de cuando se avanzaba una unidad de lo que se quería medir. Esto fue medido y seteado a partir de los valores obtenidos.

- **Conversión analógico digital:** Se utilizaron una fotorresistencia y una tira de leds por debajo de la hoja para lograr dos umbrales de tensión bien diferenciados para cada estado (blanco y negro). A partir de esto, se midió el voltaje con el ADC para saber en que color se encontraba el sensor.
  El principal problema que se encontró en la conversión, es que no se podía definir con absoluta precisión la longitud de cada cuadrado del código para saber que valor correspondía a cada instante. Para eso, se estableció una medida aproximada y se contabilizó cuanto avanzaba el carro en el mismo color, seteando un flag cada vez que este cambiaba. Así, se realizaba una división que terminaba por dar la cantidad de cuadrados leídos de cada color.
  El siguiente paso fue almacenar estos valores en registros para proceder a la decodificación correspondiente de cada línea. Para eso, se utilizaron los registros R20, R21 Y R22, llenados en el orden respectivo.
  Una vez finalizada cada columna, se llamaba al método 'desenmascarar' el cual procedía a aplicar la máscara seteada por defecto y posteriormente decodificar.

- **Decodificación:** Fueron utilizados 6 acumuladores (ACUM0, ACUM1, ACUM2, ACUM3, ACUM 4 y ACUM5) para almacenar los bits leidos en el orden decodificado correspondiente.
  La decodificación fue realizada bit a bit, rolleando cada registro al carry y dependiendo de su valor y la posición del mismo, seteando bits en los acumuladores. Cabe aclarar que el código QR tiene una estructura no cien por ciento homogénea, por lo cual dependiendo de la columna que se estaba leyendo, la forma de lectura y decodificado era distinta. Para las primeras 8 columnas, dos acumuladores se llenaron cada dos columnas (ACUM1 y ACUM2), mientras que otros (los ubicados horizontalmente, ACUM 0 Y ACUM3), se llenaron cada 4. Mientras que para las restantes, fueron llenados cada dos columnas (ACUM1, ACUM2, ACUM3 y ACUM4), y cada 4 (ACUM0 y ACUM5).
  Finalmente, a medida que cada acumulador era completamente llenado, se volcaba su valor en una tabla de memoria RAM en el orden correspondiente de lectura para que al finalizar, solo alcanzara con recorrer esta tabla. Para saber donde finalizaba, al terminar la lectura se agregó el caracter '/'.
  Si bien la estructura del código obligaba a un engorroso proceso de decodificado bit a bit, se pudo reutilizar bastante código ya que al final de cuentas, eran 4 maneras distintas de recorrer los registros donde se almacenó la lectura inicial.

- **Envío Bluetooth:** Para finalizar, se configuró al microcontrolador para enviar datos por puerto serie utilizando la interrupción de vaciado del UDRE0. A medida que se cargaban los datos, eran enviados y su valor limpiado, por lo cual la interrupción volvía a ejecutarse. A medida que esto ocurría, se incrementaban los punteros de la tabla para ir enviando letra a letra al dispositivo móvil. Una vez llegado al final, se inhabilitó nuevamente esta funcionalidad ya que sino queda enviando datos produciendo errores.

Las interrupciones utilizadas fueron:

- **OC1Aaddr:** Interrupción asociada al timer 1.
- **OC0Aaddr:** Interrupción asociada al timer 0.

- **ADCCAddr:** Interrupción asociada a la conversión analógica digital.
- **URXCAddr:** Interrupción asociada a la recepción de datos por USART.
- **UDREAddr:** Interrupción asociada al vaciado del UDRE0 para envío de datos.

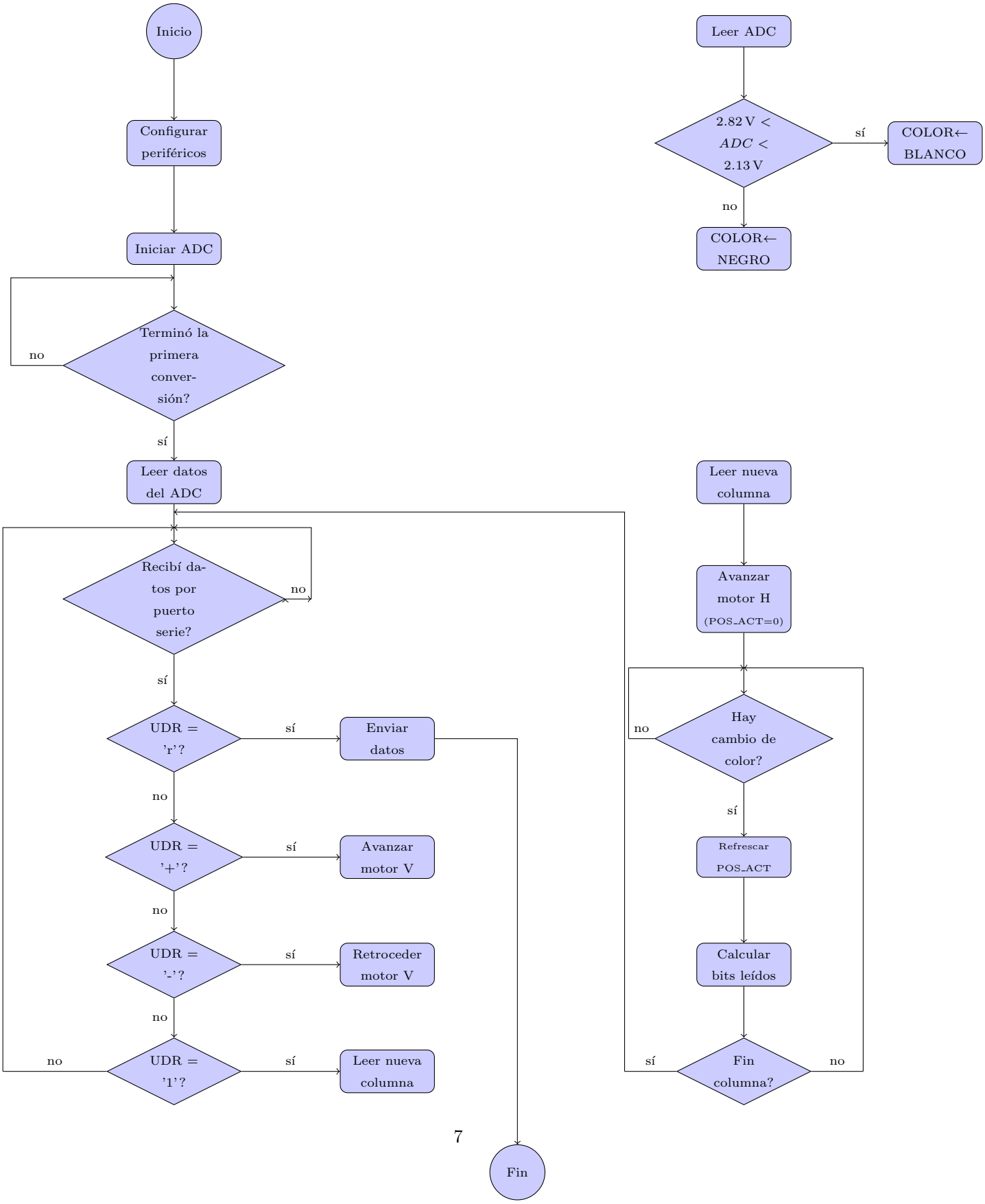


Figura 4: Diagrama de flujo del programa

6. Conclusiones y posibles mejoras

Conclusiones:

- El objetivo principal del proyecto se cumplió con éxito, y el dispositivo puede recibir, bajo ciertas limitaciones, una matriz QR impresa en una hoja de papel, guardar su contenido en la memoria del microcontrolador, procesarlo y enviar el mensaje que contiene el código a un dispositivo celular que tenga conexión por bluetooth y la aplicación adecuada. En este sentido, puede decirse que el trabajo realizado resultó muy satisfactorio, y que el objetivo de fondo que contiene la implementación de un trabajo de esta magnitud (el cual fue aprender a utilizar correctamente las herramientas del microprocesador) también se realizó con éxito.

- El proyecto entregado realizaba automáticamente casi todas las funciones esperadas, menos el desplazado vertical del sensor para leer las filas de la matriz en forma continua. En lugar de ello, se decidió realizar una calibración manual (controlada por bluetooth) al finalizar cada fila. La misma consistió en controlar los motores que manejan el desplazamiento vertical de la hoja con dos nuevos botones en la aplicación móvil desarrollada para el proyecto.

  El problema principal que se presentó fue que el control de motores para lograr precisión en el desplazado requería de más horas de trabajo y pruebas para poder calibrar la funcionalidad. Además, la estructura del dispositivo para mantener la hoja centrada y fijada era un poco endeble, lo cual terminaba provocando que los desplazamientos verticales fueran siempre distintos, no pudiendo normalizarlo a una sola medida.

  Por otra parte, fue posible realizar el escaneo para un solo tamaño del código, ya que la distinción entre puntos blancos y negros implicaba tener conocimiento previo del ancho de cada punto. Esto se hizo a modo de simplificación y con el objetivo de acotar la complejidad del programa y las posibles fallas.

- Junto con la programación necesaria para utilizar las herramientas del micro, también se realizaron diversas tareas de *hardware* que permitieron ampliar nuestros conocimientos sobre la electrónica.

- Luego de finalizar el proyecto, pudimos tomar noción de qué tan importante es la planificación y el conocimiento de lo que se va a hacer y cómo, ya que esto permite saber cuánto tiempo insumir en cada tarea y darle margen a los errores que se presentarán para poder resolverlos. Además, se entendió que existe la posibilidad de que una mala decisión al principio, afecte el desarrollo del proyecto, produciendo así que los tiempos no alcancen y no se logre cumplir el objetivo. El trabajo en equipo y la división de tareas al inicio fue fundamental para ir avanzando, ya que nos resultaba muy difícil juntarnos con frecuencia. Así, cada uno avanzó en las tareas correspondientes, logrando que los tiempos alcanzaran al final del cuatrimestre.

Posibles mejoras a realizar:

- La mejora fundamental que podría realizarse es la de automatizar completamente el proceso de lectura, logrando que el motor vertical dé pequeños saltos de manera constante sin desplazarse por demás o mover la hoja. Para ello, debería comprobarse su comportamiento con un motor paso a paso en lugar del utilizado. Además, podría implementarse la técnica del PWM para avanzar el motor horizontal. Este método no se consideró en los planes originales ya que, en un principio, la acción del motor para recorrer una fila se limita a avanzar el motor con una única velocidad. Además, utilizar este método hace que sea mucho más complejo el proceso de identificación de cada punto.

- Consideramos que sea posible mejorar la parte del programa que se encarga de la decodificación implementando una forma de almacenamiento por filas y un código hecho en C. Esto facilitaría la programación y daría la posiblidad de utilizar diferentes tipos de lectura, pero con la desventaja de que sería necesario más memoria para contener el código y las filas escaneados. Debe mencionarse que utilizar Assembler para la decodificación, siempre implica analizar bit a bit la situación, por lo cual no es posible reemplazarlo por algo que no sea engorroso de programar.

- El circuito que controlaba los motores y la tira de leds podría ser reemplazado por algo más pequeño y que realice un menor consumo de energía, ya que notamos que el regulador de tensión aumentaba bastante su temperatura, teniendo que apagar el dispositivo cada vez que se terminaba de utilizar o colocando un disipador.
- Otra cosa que se podría mejorar, es un algoritmo de control para saber cuándo es necesario realizar nuevamente la lectura de datos.

7. Código

```
1
2  ;*****************************************************************;
3  ; PROYECTO FINAL: ESCANEO, DECODIFICACIÓN Y ENVÍO DE UN CÓDIGO QR
4  ;*****************************************************************;
5
6
7
8
9  ;*************************************************************************************;
10 ;BLOQUE DE DEFINICIONES
11
12   .include "m328Pdef.inc"
13   .include "macros.inc"
14
15 ;;; SE DEFINE UN REGISTRO DE I/O, "SCANNER", CON LOS FLAGS NECESARIOS PARA EL ESCANEO
16 .equ  SCANNER = GPIOR0
17 .equ  COLOR = 0    ; Negro=1, Blanco=0
18 .equ  ESCANEANDO = 1     ; Flag para la conversión del ADC
19 .equ  CAMBIO_COLOR = 2   ; Flag para el cálculo de los bits leídos
20 .equ  FIN_COLUMNA = 3
21 .equ  AVANZO_CASILLA = 4
22 .equ  FLAG_BLUETOOTH = 5  ;Flags para la recepción por bluetooth
23 .equ  FLAG_BLUETOOTH_VACIO = 6
24
25 ;;; SE DEFINE UN REGISTRO, "MOTORES", CON LOS FLAGS NECESARIOS PARA VIGILAR EL ESTADO DE LOS MOTORES
26 .def  MOTORES = r31
27 .equ  MOTOR_H_ENABLE = 0
28 .equ  MOTOR_H_STOP_FLAG = 1
29 .equ  MOTOR_H_DIR = 2    ;DIR=FORWARD=1, DIR=REVERSE=0
30 .equ  MOTOR_V_ENABLE = 3
31 .equ  MOTOR_V_STOP_FLAG = 4
32 .equ  MOTOR_V_DIR = 5
33
34 ;;; REGISTROS Y CONSTANTES PERSONALIZADOS
35 .equ  LIMITE_FLANCOS = 255  ;Número de flancos que corresponden al tiempo que
36         ;tarda el motor Horizontal en recorrer el ancho de la página
37 .equ  TAMANIO_CASILLA = 13  ;Número de flancos que corresponden al largo de una casilla
38 .equ  CORTE_POR_REDONDEO = TAMANIO_CASILLA - 9
39 .equ  CANTIDAD_FILAS = 21
40 .equ  CANTIDAD_COLUMNAS = 21
41 .equ  ULTIMO_BIT_DE_FILA = CANTIDAD_COLUMNAS - 1
42 .equ  ULTIMO_BIT_DE_COLUMNA = CANTIDAD_FILAS - 1
43 .equ  BAUD_RATE = 0x67  ;Este es el valor que debe ponerse para tener un baud rate de 9600, que es el que usa el hc 05.
44 .equ  CARACTER_FIN = '/'  ;Caracter con el que terminará el código QR
45 .equ  INICIO_TABLA = 0x0100
46
47 .def  CERO = r0 ;Registros utilizados para una prueba de envío
48 .def  UNO = r1
49 .def  CANT = r3 ;Cantidad de bits leídos del mismo color
50 .def  SIZE = r4 ;Registro que guarda el tamaño de un bit
51 .def  SIZE_2 = r5
52
53 .def  rtemp1 = r16 ;Registros auxiliares
54 .def  rtemp2 = r17
55 .def  BIT_ACTUAL = r18 ;Posición actual del motor en una columna.
56 .def  POS_INICIAL = r8 ;Estos dos registros sirven para calcular la cantidad de bits que se leyeron del mismo color
57 .def  POS_FINAL = r9
58
59 ;;; REGISTROS PARA LA DECODIFICACIÓN
60 .def  CONT_H = r19 ;Cuenta la cantidad de columnas leídas
61 .def  CANT_LETRAS_LEIDAS = r6 ;Cuenta la cantidad de letras decodificadas
62 .def  LENGTH = r7
63 .def  ACUM1 = r23 ;La idea de estos acumuladores es ir sosteniendo los valores ya procesados, que luego se guardan en SRAM
64 .def  ACUM2 = r24
65 .def  ACUM3 = r25
66 .def  ACUM4 = r28
67 .def  ACUM5 = r29
68 .def  ACUM6 = r30
69
70 .equ  B7 = 0x80
71 .equ  B6 = 0x40
72 .equ  B5 = 0x20
73 .equ  B4 = 0x10
74 .equ  B3 = 0x08
75 .equ  B2 = 0x04
76 .equ  B1 = 0x02
77 .equ  B0 = 0x01
```

```
78
79  ;;; DEFINICIÓN DE LOS PUERTOS
80  .equ   MOTOR_H_OUT1_PORT_DIR = DDRD     ;Motores HORIZONTAL y VERTICAL.
81  .equ   MOTOR_H_OUT1_PORT = PORTD    ;Cada motor tiene dos pines (OUT1 y OUT2) que controlan la
82  .equ   MOTOR_H_OUT1_PIN = 6       ;su movimiento:
83  .equ   MOTOR_H_OUT2_PORT_DIR = DDRD     ; forward --> OUT1=0, OUTT2=1
84  .equ   MOTOR_H_OUT2_PORT = PORTD    ; reverse --> OUT1=1, OUT2=0
85  .equ   MOTOR_H_OUT2_PIN = 7       ; stop  --> OUT1=0, OUT2=0
86  .equ   MOTOR_H_ENABLE_PORT_DIR = DDRB     ;Además, cada motor tiene un ENABLE que habilita su funcionamiento.
87  .equ   MOTOR_H_ENABLE_PORT = PORTB
88  .equ   MOTOR_H_ENABLE_PIN = 0
89  .equ   MOTOR_V_OUT1_PORT_DIR = DDRD
90  .equ   MOTOR_V_OUT1_PORT = PORTD
91  .equ   MOTOR_V_OUT1_PIN = 2
92  .equ   MOTOR_V_OUT2_PORT_DIR = DDRD
93  .equ   MOTOR_V_OUT2_PORT = PORTD
94  .equ   MOTOR_V_OUT2_PIN = 3
95  .equ   MOTOR_V_ENABLE_PORT_DIR = DDRB
96  .equ   MOTOR_V_ENABLE_PORT = PORTB
97  .equ   MOTOR_V_ENABLE_PIN = 1
98
99  .equ   COUNTER0_PORT_DIR = DDRD     ;Configuración de los puertos utilizados como contadores externos,
100 .equ   COUNTER0_PIN = 4       ;conectados a los encoders del motors
101 .equ   COUNTER1_PORT_DIR = DDRD
102 .equ   COUNTER1_PIN = 5
103
104 .equ   SENSOR_INPUT_PORT_DIR = DDRC     ;Entrada analógica del sensor de luz.
105 .equ   SENSOR_INPUT_PIN = 1
106
107 .equ   SENSOR_LED_PORT_DIR = DDRC     ;Led que viene con el sensor. No fue necesario su utilización,
108 .equ   SENSOR_LED_PORT = PORTC      ;pero se confugura para mantenerlo apagado.
109 .equ   SENSOR_LED_PIN = 0
110
111 .equ     LED_PRUEBA_PORT_DIR =  DDRB      ;Led de prueba, de la placa de Arduino UNO.
112 .equ     LED_PRUEBA_PORT     =  PORTB
113 .equ     LED_PRUEBA_PIN      =  5
114
115
116
117 ;*********************************************************************************************;
118 ;COMIENZO DEL PROGRAMA
119
120 ;; Tabla de interrupciones
121 .org 0x0000
122   rjmp inicio
123 .org OC1Aaddr
124   rjmp counter1_int_handler
125 .org OC0Aaddr
126   rjmp counter0_int_handler
127 .org ADCCaddr
128   rjmp adc_int_handler
129 .org URXCaddr
130   rjmp urx_int_handler ;Interrupción producida por la recepción de datos
131 .org UDREaddr
132   rjmp udre_int_handler
133 .org INT_VECTORS_SIZE
134
135
136 inicio:
137 ;; Configuraciones varias (ver sección "FUNCIONES DE CONFIGURACIÓN")
138   ldi rtemp1,HIGH(RAMEND)
139   out sph,rtemp1
140   ldi rtemp1,LOW(RAMEND)
141   out spl,rtemp1
142   rcall configurar_puertos
143   rcall configurar_adc
144   rcall configurar_contadores
145 ;; Inicialización de los registros que se van a utilizar
146 inicializacion_registros:
147   ldi XL, LOW(INICIO_TABLA)    ;Puntero a la tabla donde se guarda el código
148   ldi XH, HIGH(INICIO_TABLA)
149   ldi rtemp1,'0'       ;Ascii para cero y uno (Esto se utilizó para una prueba de lectura y envío)
150   mov CERO,rtemp1
151   ldi rtemp1,'1'
152   mov UNO,rtemp1
153   ldi rtemp1,TAMANIO_CASILLA   ;Esto se usa después para calcular los bits escaneados. Ver "calcular_bits_leidos".
154   mov SIZE,rtemp1
155   ldi rtemp1,CORTE_POR_REDONDEO
156   mov SIZE_2,rtemp1
157   eor r20,r20       ;Registros varios que se usan en el escaneo. r20, r21 y r22 se usan
158   eor r21,r21       ;para guardar los bits de cada columna.
159   eor r22,r22
160   eor CONT_H,CONT_H
161   eor BIT_ACTUAL,BIT_ACTUAL
162   eor CANT,CANT
163   cbi SCANNER,FLAG_BLUETOOTH   ;Flags varios
164   cbi SCANNER,AVANZO_CASILLA
165   cbi SCANNER,FIN_COLUMNA
166   eor ACUM1, ACUM1     ;Registros para la decodificación
167   eor ACUM2, ACUM2
168   eor ACUM3, ACUM3
169   eor ACUM4, ACUM4
170   eor ACUM5, ACUM5
171   eor ACUM6, ACUM6
172   ldi rtemp1,0x00
173   mov CANT_LETRAS_LEIDAS, rtemp1
174   ldi rtemp1,0x1A
175   mov LENGTH, rtemp1 ;Inicializo el valor de length con la mayor cantidad de datos que puedo llegar a leer.
```

```
176
177
178 ;; Hasta acá se realizaron las configuraciones necesarias para empezar el programa.
179 ;; Ahora se realiza la primera lectura del ADC.
180 primera_conversion_adc:
181   sei ;Se habilitan las interrupciones.
182   sbi SCANNER,ESCANEANDO
183   lds rtemp1,ADCSRA
184   sbr rtemp1,1<<ADSC ;Se indica que el ADC está escaneando, y se habilita para que empiece
185   sts ADCSRA,rtemp1
186 esperar_primera_conversion:
187   sbic SCANNER,ESCANEANDO ;Si no terminó la conversión del ADC, ESCANEANDO=1
188   rjmp esperar_primera_conversion
189   sbi SCANNER,ESCANEANDO
190   rcall leer_datos_adc ;Con esto, se configura el color que se está leyendo cuando empieza a leer
191   cbi SCANNER,CAMBIO_COLOR
192
193 ;; Se espera a recibir datos del bluetooth por interrupción. Pueden pasar tres cosas:
194 ;; 1: se pide enviar datos al celular,
195 ;; 2: se pide mover el motor VERTICAL, o
196 ;; 3: se pide mover el motor HORIZONTAL para leer una nueva columna.
197 esperar_recibir_datos:
198   sbic  SCANNER,FLAG_BLUETOOTH ;Si FLAG_BLUETOOTH=1, se recibieron datos.
199   rjmp RECIBIR_DATOS
200   rjmp esperar_recibir_datos
201 RECIBIR_DATOS:
202   cbi SCANNER,FLAG_BLUETOOTH
203   LOAD rtemp2, UDR0 ;Levanto en R17 el valor recibido por puerto serie.
204   CPI rtemp2, 'r' ;La app envía el caracter 'r' si solicita recibir datos. En caso de ser así, va a ENVIAR_DATOS
205   BREQ ENVIAR_DATOS
206   CPI rtemp2, '1' ;La app envía el caracter '1' si desea encender motores.
207   breq leer_columna
208   cpi rtemp2,'+'
209   breq avanzar_motor_V
210   cpi rtemp2,'-'
211   breq retroceder_motor_V
212   rjmp esperar_recibir_datos
213
214
215 ;; Opción 1: se pide enviar datos al celular
216 VOLVER_A_ENVIAR: ;Función para enviar datos al celular
217   sbis SCANNER,FLAG_BLUETOOTH_VACIO ;Espera a que el valor se haya enviado.
218   rjmp VOLVER_A_ENVIAR
219   LDI rtemp1, (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0) ;Deshabilito la interrupción de UDRE0.
220   STORE UCSR0B, rtemp1
221   cbi SCANNER,FLAG_BLUETOOTH_VACIO
222 ENVIAR_DATOS: ;Se lee la tabla de SRAM hasta el CARACTER_FIN (donde termina el mensaje), y se envían por puerto serie.
223   ld rtemp2,X+
224   cpi rtemp2,CARACTER_FIN ;Si llegó al final del mensaje, termina el envío.
225   breq SALIR
226   STORE UDR0, rtemp2 ;Carga en UDR0 el valor a enviar
227   LDI rtemp1, (1<<RXEN0)|(1<<TXEN0)|(1<<UDRIE0)|(1<<RXCIE0) ;Habilita la interrupción de UDRE0 vacío
228   STORE UCSR0B, rtemp1 ;Carga el valor de la configuración
229   RJMP VOLVER_A_ENVIAR
230
231 SALIR:  ;Función para deshabilitar el envío de datos.
232   LDI rtemp2, ' '
233   STORE UDR0, rtemp2 ;Cargo este caracter para darle un cierre al mensaje.
234   LDI rtemp1, (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0) ;Deshabilito la interrupción de UDRE0 para que no enloquezca.
235   STORE UCSR0B, rtemp1
236   ldi XL,LOW(INICIO_TABLA)
237   ldi XH,HIGH(INICIO_TABLA)
238   cbi SCANNER,FLAG_BLUETOOTH
239   rjmp esperar_recibir_datos
240
241
242 ;;Opción 2: Se pide mover el motor VERTICAL en alguna dirección, para calibrar la hoja.
243 avanzar_motor_V:
244   cbi SCANNER,AVANZO_CASILLA
245   rcall motor_B_forward
246 esperar_motorV:
247   sbis SCANNER,AVANZO_CASILLA
248   rjmp esperar_motorV
249   rcall motor_B_stop
250   cbi SCANNER,AVANZO_CASILLA
251   cbi SCANNER,FLAG_BLUETOOTH
252   rjmp esperar_recibir_datos
253
254 retroceder_motor_V:
255   rcall motor_B_reverse
256   rcall delay
257   rcall motor_B_stop
258   cbi SCANNER,FLAG_BLUETOOTH
259   rjmp esperar_recibir_datos
260
261
262 ;; Opción 3: se pide leer una nueva columna. Aquí comienza el proceso de escaneo y decodificación.
263 leer_columna:
264   eor BIT_ACTUAL,BIT_ACTUAL
265   eor r20,r20
266   eor r21,r21
267   eor r22,r22
268   rcall guardar_primer_bit ; Se guarda el primer bit de la columna, mientras el carro está quieto
269   ldi rtemp1,0x00
270   sts TCNT1H,rtemp1
271   sts TCNT1L,rtemp1
272   rcall motor_A_forward
273 avanzar_con_motor_H:
274   sbic SCANNER,FIN_COLUMNA
```

```
  rjmp leer_ultimos_bit
  rcall leer_datos_adc
  sbis SCANNER,CAMBIO_COLOR
  rjmp avanzar_con_motor_H
  lds POS_FINAL,TCNT1L
  cbi SCANNER,CAMBIO_COLOR
  rcall calcular_bits_leidos
  mov POS_INICIAL,POS_FINAL
  rcall guardar_bits_en_registros
  rjmp avanzar_con_motor_H
leer_ultimos_bit:
  rcall motor_A_stop
  ldi rtemp1,LIMITE_FLANCOS
  mov POS_FINAL,rtemp1
  rcall calcular_bits_leidos
  mov POS_INICIAL,POS_FINAL
  ldi rtemp1,(1<<COLOR) ;toggle del color para seguir guardando con la misma lógica
  in rtemp2,SCANNER
  eor rtemp2,rtemp1
  out SCANNER,rtemp2
  rcall guardar_bits_en_registros
  lsl r22
  lsl r22
  lsl r22
volver_a_inicio:
  cbi SCANNER,FIN_COLUMNA
  ldi rtemp1,0x00
  sts TCNT1H,rtemp1
  sts TCNT1L,rtemp1
  rcall motor_A_reverse
seguir_retrocediendo:
  sbis SCANNER,FIN_COLUMNA
  rjmp seguir_retrocediendo
  rcall motor_A_stop
  cbi SCANNER,FIN_COLUMNA
  rcall decodificar_linea
  cbi SCANNER,FLAG_BLUETOOTH
  rjmp esperar_recibir_datos


;*********************************************************************************************
;FUNCIONES DE CONFIGURACIÓN

;; Función para configurar los puertos utilizados.
configurar_puertos:
  push rtemp1
  ;; LED DE PRUEBA
  ldi  rtemp1,0xff
  out  LED_PRUEBA_PORT_DIR,rtemp1
  out  LED_PRUEBA_PORT,rtemp1
  cbi  LED_PRUEBA_PORT,LED_PRUEBA_PIN
  ;; INPUT SENSOR
  cbi SENSOR_INPUT_PORT_DIR,SENSOR_INPUT_PIN  ;Input = Sensor de luz
  sbi SENSOR_LED_PORT_DIR,SENSOR_LED_PIN    ;Output = Led del sensor de luz
  cbi SENSOR_LED_PORT,SENSOR_LED_PIN    ;Se configura el estado inicial del led reflectivo en apagado.
  ;; CONTADORES
  cbi COUNTER0_PORT_DIR,COUNTER0_PIN    ;Input = Encoder del motor A
  cbi COUNTER1_PORT_DIR,COUNTER1_PIN    ;Input = Encoder del motor B
  ;; MOTOR HORIZONTAL (MUEVE EL CARRITO)
  sbi MOTOR_H_OUT1_PORT_DIR,MOTOR_H_OUT1_PIN  ;Output = motor horizontal
  sbi MOTOR_H_OUT2_PORT_DIR,MOTOR_H_OUT2_PIN
  sbi MOTOR_H_ENABLE_PORT_DIR,MOTOR_H_ENABLE_PIN
  rcall motor_A_stop
  sbi MOTOR_H_ENABLE_PORT_DIR,MOTOR_H_ENABLE_PIN
  ;; MOTOR VERTICAL (MUEVE EL RODILLO)
  sbi MOTOR_V_OUT1_PORT_DIR,MOTOR_V_OUT1_PIN  ;Output = motor vertical
  sbi MOTOR_V_OUT2_PORT_DIR,MOTOR_V_OUT2_PIN
  sbi MOTOR_V_ENABLE_PORT_DIR,MOTOR_V_ENABLE_PIN
  rcall motor_B_stop
  sbi MOTOR_V_ENABLE_PORT_DIR,MOTOR_V_ENABLE_PIN
  ;; BLUETOOTH Y PUERTO SERIE.
  ;; Setear el bit rxcie0 habilita la interrupción del flag rxc del ucsr0a.
  ;; Al completar la recepción, rxc se setea a high.
  ;; Si rxcie0 = 1, cambiar rxc a uno fuerza la interrupción.
  ;; Setear el bit udrie0 (usart data register empty interrupt enable).
  ;; Cuando el udr0 esta listo para recibir nuevos datos, el UDRE (usart data register empty flag) se pone en 1.
  ;; Si UDRIE0 = 1 y si se pone UDRE en 1, fuerza la interrupción.
  ldi rtemp1, (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0) ;habilito como transmisor, receptor y habilito interrupciones de recepcion
  store UCSR0B, rtemp1
  LDI rtemp1, (1<<UCSZ01)|(1<<UCSZ00)|(1<<UMSEL01) ;8 bit data, sin paridad y 1 bit de parada.
  STORE UCSR0C, rtemp1
  LDI rtemp1, BAUD_RATE ;9600 baud rate
  STORE UBRR0L, rtemp1
  pop rtemp1
  ret

;; Función para configurar el ADC
configurar_adc:
  push rtemp1
  ldi rtemp1,(1<<REFS0)|0x01          ;Referencia=VCC. Entrada=ADC1=PC1.
  sts ADMUX,rtemp1            ;Ajustado a izquierda
  ldi rtemp1,(1<<ADEN)|(1<<ADIE)|(1<<ADATE)|0x04    ;Habilitar ADC. Habilitar interrupcion.
  sts ADCSRA,rtemp1           ;Prescaler=ck/16.
  ldi rtemp1,0
  sts ADCSRB,rtemp1
  pop rtemp1
  ret

```

```
373 ;; Función de configuración de los timers/contadores.
374 configurar_contadores:
375   rcall config_timer1
376   rcall config_timer0
377   ret
378 config_timer1:
379   push rtemp1
380   ldi rtemp1,HIGH(LIMITE_FLANCOS)
381   sts OCR1AH,rtemp1
382   ldi rtemp1,LOW(LIMITE_FLANCOS)
383   sts OCR1AL,rtemp1
384   ldi rtemp1,0
385   sts TCCR1A,rtemp1
386   ldi rtemp1,(1<<WGM12)|(1<<CS12)|(1<<CS11)|(1<<CS10)    ;Configurado en modo CTC, con clock externo en T1
387   sts TCCR1B,rtemp1
388   ldi rtemp1,(1<<OCIE1A)            ;Habilitar interrupciones
389   sts TIMSK1,rtemp1
390   ldi rtemp1,(1<<OCF1A)
391   out TIFR1,rtemp1
392   pop rtemp1
393   ret
394 config_timer0:
395   push rtemp1
396   ldi rtemp1,2
397   out OCR0A,rtemp1
398   ldi rtemp1,(1<<WGM01)
399   out TCCR0A,rtemp1
400   ldi rtemp1,(1<<CS02)|(1<<CS01)|(1<<CS00)   ;Configurado en modo CTC, con clock externo en T0
401   out TCCR0B,rtemp1
402   ldi rtemp1,(1<<OCIE0A)        ;Habilitar interrupciones
403   sts TIMSK0,rtemp1
404   ldi rtemp1,(1<<OCF0A)
405   out TIFR0,rtemp1
406   pop rtemp1
407   ret
408
409 ;; Manejo de las interrupciones
410 adc_int_handler:
411   cbi SCANNER,ESCANEANDO
412   reti
413 counter1_int_handler:
414   sbi SCANNER,FIN_COLUMNA
415   reti
416 counter0_int_handler:
417   sbi SCANNER,AVANZO_CASILLA
418   reti
419 udre_int_handler:
420   sbi SCANNER,FLAG_BLUETOOTH_VACIO
421   reti
422 urx_int_handler:
423   sbi SCANNER,FLAG_BLUETOOTH
424   reti
425
426 ;*********************************************************************************************;
427 ;FUNCIONES AUXILIARES
428
429 ;; Rutina de delay usando el Timer2
430 delay:
431   push rtemp2
432   eor rtemp2,rtemp2
433 loop_delay:
434   rcall delay_timer2
435   inc rtemp2
436   cpi rtemp2,3
437   brne loop_delay
438   pop rtemp2
439   ret
440 delay_timer2:
441   push rtemp1
442   ldi rtemp1,0
443   sts TCNT2, rtemp1     ;Valor inicial
444   ldi rtemp1,100
445   sts OCR2A,rtemp1     ;Umbral
446   ldi rtemp1,(1<<WGM21)
447   sts TCCR2A,rtemp1     ;Modo CTC
448   ldi rtemp1,(1<<CS22)|(1<<CS21)|(1<<CS20)
449   sts TCCR2B,rtemp1
450 again:
451   in rtemp1,TIFR2
452   sbrs rtemp1,OCF2A
453   rjmp again
454   ldi rtemp1,0          ;Detener Timer
455   sts TCCR2B,rtemp1
456   ldi rtemp1,(1<<OCF2A)
457   out TIFR2,rtemp1
458   pop rtemp1
459   ret
460
461 ;; Funciones para controlar los motores
462 motor_A_forward:
463   cbi MOTOR_H_OUT1_PORT,MOTOR_H_OUT1_PIN
464   sbi MOTOR_H_OUT2_PORT,MOTOR_H_OUT2_PIN
465   sbr MOTORES,1<<MOTOR_H_DIR
466   cbr MOTORES,1<<MOTOR_H_STOP_FLAG
467   ret
468 motor_A_stop:
469   cbi MOTOR_H_OUT1_PORT,MOTOR_H_OUT1_PIN
470   cbi MOTOR_H_OUT2_PORT,MOTOR_H_OUT2_PIN
471   sbr MOTORES,1<<MOTOR_H_STOP_FLAG
```

```
472   ret
473 motor_A_reverse:
474   sbi MOTOR_H_OUT1_PORT,MOTOR_H_OUT1_PIN
475   cbi MOTOR_H_OUT2_PORT,MOTOR_H_OUT2_PIN
476   sbr MOTORES,1<<MOTOR_H_DIR
477   cbr MOTORES,1<<MOTOR_H_STOP_FLAG
478   ret
479 motor_B_forward:
480   cbi MOTOR_V_OUT1_PORT,MOTOR_V_OUT1_PIN
481   sbi MOTOR_V_OUT2_PORT,MOTOR_V_OUT2_PIN
482   sbr MOTORES,1<<MOTOR_V_DIR
483   cbr MOTORES,1<<MOTOR_V_STOP_FLAG
484   ret
485 motor_B_stop:
486   cbi MOTOR_V_OUT1_PORT,MOTOR_V_OUT1_PIN
487   cbi MOTOR_V_OUT2_PORT,MOTOR_V_OUT2_PIN
488   sbr MOTORES,1<<MOTOR_V_STOP_FLAG
489   ret
490 motor_B_reverse:
491   sbi MOTOR_V_OUT1_PORT,MOTOR_V_OUT1_PIN
492   cbi MOTOR_V_OUT2_PORT,MOTOR_V_OUT2_PIN
493   cbr MOTORES,1<<MOTOR_V_DIR
494   cbr MOTORES,1<<MOTOR_V_STOP_FLAG
495   ret
496
497 ;; Función para procesar los datos del ADC. Cuando se sale de esta función,
498 ;; se sabe qué color se está leyendo y si hubo un cambio de color con
499 ;; respecto a la última vez que se llamó a esta función.
500 leer_datos_adc:
501   push rtemp1
502   push rtemp2
503   lds rtemp1,ADCL
504   lds rtemp2,ADCH
505 comparar_con_umbral:
506   cpi rtemp2,0x01
507   breq chequear_blanco
508   cpi rtemp2,0x02
509   breq chequear_blanco2    ;Si pasa esto, veo si es blanco
510   cpi rtemp2,0x03
511   breq chequear_negro    ;Si pasa esto, veo si es negro
512   rjmp terminar_conversion  ;Si llegó hasta acá, estoy leyendo ruido
513 chequear_blanco:
514   cpi rtemp1,180     ;Cota inferior de blanco
515   brsh es_blanco
516   rjmp terminar_conversion
517 chequear_blanco2:
518   cpi rtemp1,80
519   brlo es_blanco
520   rjmp terminar_conversion
521 es_blanco:
522   sbic SCANNER,COLOR
523   sbi SCANNER,CAMBIO_COLOR
524   cbi SCANNER,COLOR
525   rjmp terminar_conversion
526 chequear_negro:
527   cpi rtemp1,0     ;Cota inferior negro
528   brsh es_negro
529   rjmp terminar_conversion
530 es_negro:
531   sbis SCANNER,COLOR
532   sbi SCANNER,CAMBIO_COLOR
533   sbi SCANNER,COLOR
534   rjmp terminar_conversion
535 terminar_conversion:
536   pop rtemp1
537   pop rtemp2
538   ret
539
540 ;; Función para guardar el primer bit de la columna.
541 guardar_primer_bit:
542   set
543   sbis SCANNER,COLOR
544   clt
545   bld r20,0
546   eor POS_INICIAL,POS_INICIAL
547   ret
548
549 ;; Esta función se llama cuando hubo un cambio de color y se utiliza para
550 ;; calcular cuántos bits del mismo color se leyeron.
551 calcular_bits_leidos:
552   push rtemp1
553   mov rtemp1,POS_FINAL
554   sub rtemp1,POS_INICIAL
555   cp rtemp1,SIZE
556   brlo redondear
557 dividir:
558   sub rtemp1,SIZE
559   inc CANT
560   cp rtemp1,SIZE
561   brlo redondear
562   rjmp dividir
563 redondear:
564   cp rtemp1,SIZE_2
565   brlo terminar_division
566   inc CANT
567 terminar_division:
568   pop rtemp1
569   ret
570
```

```
;; Luego de calcular la cantidad de bits que se leyeron y se guardan en CANT,
;; se llama a esta fucnión para guardarlos en r20, r21 y r22.
guardar_bits_en_registros:
  set
  sbic SCANNER,COLOR
  clt        ; Se guarda el color leído en flag T.
donde_estoy:
  cpi BIT_ACTUAL,0x10
  brsh guardar_en_r22
  cpi BIT_ACTUAL,0x08
  brsh guardar_en_r21
guardar_en_r20:
  tst CANT
  breq terminar_guardado
  lsl r20
  bld r20,0
  dec CANT
  inc BIT_ACTUAL
  rjmp donde_estoy
guardar_en_r21:
  tst CANT
  breq terminar_guardado
  lsl r21
  bld r21,0
  dec CANT
  inc BIT_ACTUAL
  rjmp donde_estoy
guardar_en_r22:
  tst CANT
  breq terminar_guardado
  lsl r22
  bld r22,0
  dec CANT
  inc BIT_ACTUAL
  rjmp donde_estoy
terminar_guardado:
  ret


;; Función de decodificación QR. (Desde acá hasta el final del código se encuentra la función que se encarga de
    decodificar).
desenmascarar:
  ldi rtemp1,0xff
  eor r20,rtemp1
  eor r21,rtemp1
  eor r22,rtemp1
  rjmp continuar_sin_mascara
decodificar_linea:
; Lo primero que se hace es sacarle la máscara al código, la cual se conoce previamente.
  cpi CONT_H,20
  breq desenmascarar
  cpi CONT_H,17
  breq desenmascarar
  cpi CONT_H,11
  breq desenmascarar
  cpi CONT_H,8
  breq desenmascarar
  cpi CONT_H,5
  breq desenmascarar
  cpi CONT_H,2
  breq desenmascarar
continuar_sin_mascara:
;Veo en que columna estoy parado y en base a eso llamo al método para decodificar
  CP LENGTH, CANT_LETRAS_LEIDAS
  BREQ salto_a_fin
  CPI CONT_H, 0x00 ;Es primera columna (col 0)
  BRNE no_es_cero
  JMP leo_tipo_1A
salto_a_fin:
  JMP FIN
no_es_cero:
  CPI CONT_H, 0x01 ;Es segunda columna (col 1)
  BRNE no_es_uno
  JMP leo_tipo_1B
no_es_uno:
  CPI CONT_H, 0x02 ;Es tercera columna (col 2)
  BRNE no_es_dos
  JMP leo_tipo_2A
no_es_dos:
  CPI CONT_H, 0x03 ;Es cuarta columna (col 3)
  BRNE no_es_tres
  JMP leo_tipo_2B
no_es_tres:
  CPI CONT_H, 0x04 ;Es quinta columna (col 4)
  BRNE no_es_cuatro
  JMP leo_tipo_1A
no_es_cuatro:
  CPI CONT_H, 0x05 ;Es sexta columna (col 5)
  BRNE no_es_cinco
  JMP leo_tipo_1B
no_es_cinco:
  CPI CONT_H, 0x06 ;Es septima columna (col 6)
  BRNE no_es_seis
  JMP leo_tipo_2A
no_es_seis:
  CPI CONT_H, 0x07 ;Es octava columna (col 7)
  BRNE no_es_siete
  JMP leo_tipo_2B
no_es_siete:
```

```
669   CPI CONT_H, 0x08 ;Es novena columna (col 8)
670   BRNE no_es_ocho
671   JMP leo_tipo_1A
672 no_es_ocho:
673   CPI CONT_H, 0x09 ;Es decima columna (col 9)
674   BRNE no_es_nueve
675   JMP leo_tipo_1B
676 no_es_nueve:
677   CPI CONT_H, 0x0A ;Es onceava columna (col 10)
678   BRNE no_es_diez
679   JMP leo_tipo_2A
680 no_es_diez:
681   CPI CONT_H, 0x0B ;Es doceava columna (col 11)
682   BRNE no_es_once
683   JMP leo_tipo_2B
684 no_es_once:
685   CPI CONT_H,21 ;Es doceava columna (col 11)
686   BREQ FIN
687   INC CONT_H
688   RET
689 FIN:
690   LDI ACUM1, CARACTER_FIN ;Guardo caracter de fin para luego saber hasta donde tengo que leer por bluetooth
691   RCALL guardo_acum1
692   LDI XL, LOW(INICIO_TABLA) ;Reinicio punteros
693   LDI XH, HIGH(INICIO_TABLA)
694   INC XL
695   INC XL
696 buscar_fin_mensaje:    ;Si el byte que se lee empieza con 0000 significa que ahí es donde debería estar el caracter final
697       ld rtemp1,X+
698       cpi rtemp1,16
699       brsh buscar_fin_mensaje
700 encontrado:
701     dec XL
702     cpi XL,0xff
703     breq decrementar_parte_baja
704 guardar_caracter:
705       ldi rtemp1,CARACTER_FIN
706     st X,rtemp1
707     ldi XL,LOW(INICIO_TABLA)
708     ldi XH,HIGH(INICIO_TABLA)
709   RET
710 decrementar_parte_baja:
711     dec XH
712     rjmp guardar_caracter
713
714 ; ACÁ EMPIEZAN LOS MÉTODOS DE LECTURA Y DECODIFICACIÓN. LA REGLA ES: EL METODO DE BRCC
715 ; HACE REFERENCIA AL ORDEN DE LECTURA (VER EXCEL), Y EL SBR HACE REFERENCIA AL BIT A SETEAR.
716
717 ;PRIMER MÉTODO DE LECTURA
718 ;La columna tipo 1A tiene dos variantes, la primera es la que llega hasta la fila 12, la segunda finaliza en la 21
719 leo_tipo_1A:
720   ROL R20 ;En c ahora tengo el primer bit
721   BRCC quinto_bit_acum1_sigo ;Acum1.3
722   SBR ACUM1, B3
723 quinto_bit_acum1_sigo:
724   RCALL incremento_contador_verticalr20
725   BRCC septimo_bit_acum1_sigo ;Acum1.1
726   SBR ACUM1, B1
727   ;Acá termina primera parte de acum1 y empieza primera parte de acum2
728 septimo_bit_acum1_sigo:
729   RCALL incremento_contador_verticalr20
730   BRCC primer_bit_acum2_sigo ;Acum2.7
731   SBR ACUM2, B7
732 primer_bit_acum2_sigo:
733   RCALL incremento_contador_verticalr20
734   BRCC tercer_bit_acum2_sigo ;Acum2.5
735   SBR ACUM2, B5
736 tercer_bit_acum2_sigo:
737   RCALL incremento_contador_verticalr20
738   BRCC quinto_bit_acum2_sigo ;Acum2.3
739   SBR ACUM2, B3
740 quinto_bit_acum2_sigo:
741   RCALL incremento_contador_verticalr20
742   BRCC septimo_bit_acum2_sigo ;Acum2.1
743   SBR ACUM2, B1
744   ;Acá termina primera parte de acum2 y empieza primera parte de acum3
745 septimo_bit_acum2_sigo:
746   RCALL incremento_contador_verticalr20
747   BRCC primer_bit_acum3_sigo ;Acum3.7
748   SBR ACUM3, B7
749 primer_bit_acum3_sigo:
750   RCALL incremento_contador_verticalr20
751   BRCC tercer_bit_acum3_sigo ;Acum3.5
752   SBR ACUM3, B5
753   ;Acá termina r20
754 tercer_bit_acum3_sigo:
755   RCALL incremento_contador_verticalr21
756   BRCC quinto_bit_acum3_sigo ;Acum3.3
757   SBR ACUM3, B3
758 quinto_bit_acum3_sigo:
759   RCALL incremento_contador_verticalr21
760   BRCC septimo_bit_acum3_sigo ;Acum3.1
761   SBR ACUM3, B1
762   ;Acá termina primera parte de acum3 y empieza primera parte de acum4
763 septimo_bit_acum3_sigo:
764   RCALL incremento_contador_verticalr21
765   BRCC primer_bit_acum4_sigo ;Acum4.7
766   SBR ACUM4, B7
```

```
767 primer_bit_acum4_sigo:
768   RCALL incremento_contador_verticalr21
769   BRCC chequeo_fila ;Acum4.5
770   SBR ACUM4, B5
771 ;Se terminó la lectura de la primera columna, acá se debería esperar a que se vuelvan a llenar R20, R21 y R22
772 ;Si es tipo 1 C o D, debo seguir guardando en el acumulador el bit vertical, sino salto al final
773 chequeo_fila:
774   CPI CONT_H, 0x08
775   BRNE fin_columna1
776 ; Esta parte corresponde a si la columna es la ocho (hay que hacer otras cosas con los acumuladores).
777 es_columna8:
778   RCALL incremento_contador_verticalr21
779   BRCC septimo_bit_acum4_sigo_c8 ;Acum4.3
780   SBR ACUM4, B3
781 septimo_bit_acum4_sigo_c8:
782   RCALL incremento_contador_verticalr21
783   BRCC salto_linea ;Acum4.1
784   SBR ACUM4, B1
785 salto_linea: ;Linea que no tiene datos
786   RCALL incremento_contador_verticalr21 ;Roto un bit que no me interesa
787   RCALL incremento_contador_verticalr21
788   BRCC primer_bit_acum5_sigo_c8 ;Acum4.7
789   SBR ACUM5, B7
790 primer_bit_acum5_sigo_c8:
791   RCALL incremento_contador_verticalr22
792   BRCC tercer_bit_acum5_sigo_c8 ;Acum4.5
793   SBR ACUM5, B5
794   ;Acá termina r20
795 tercer_bit_acum5_sigo_c8:
796   RCALL incremento_contador_verticalr22
797   BRCC quinto_bit_acum5_sigo_c8 ;Acum4.3
798   SBR ACUM5, B3
799 quinto_bit_acum5_sigo_c8:
800   RCALL incremento_contador_verticalr22
801   BRCC septimo_bit_acum5_sigo_c8 ;Acum4.1
802   SBR ACUM5, B1
803 septimo_bit_acum5_sigo_c8:
804   RCALL incremento_contador_verticalr22
805   BRCC primer_bit_acum6_sigo_c8 ;Acum6.7
806   SBR ACUM6, B7
807   ;Acá termina r22
808 primer_bit_acum6_sigo_c8:
809   RCALL incremento_contador_verticalr22
810   BRCC fin_columna1 ;Acum6.5
811   SBR ACUM6, B5
812   ;Finalizó toda la columna
813 fin_columna1:
814   INC CONT_H
815   RET
816
817 ;SEGUNDO MÉTODO DE LECTURA
818 leo_tipo_1B:
819   ROL R20 ;En c ahora tengo el primer bit
820   BRCC sexto_bit_acum1_sigo ;Acum1.2
821   SBR ACUM1, B2
822 sexto_bit_acum1_sigo:
823   RCALL incremento_contador_verticalr20
824   BRCC octavo_bit_acum1_sigo  ;Acum1.0
825   SBR ACUM1, B0
826   ;Acá termina primera parte de acum1 y empieza primera parte de acum2
827 octavo_bit_acum1_sigo:
828   RCALL incremento_contador_verticalr20
829   BRCC segundo_bit_acum2_sigo ;Acum2.6
830   SBR ACUM2, B6
831 segundo_bit_acum2_sigo:
832   RCALL incremento_contador_verticalr20
833   BRCC cuarto_bit_acum2_sigo ;Acum2.4
834   SBR ACUM2, B4
835 cuarto_bit_acum2_sigo:
836   RCALL incremento_contador_verticalr20
837   BRCC sexto_bit_acum2_sigo ;Acum2.2
838   SBR ACUM2, B2
839 sexto_bit_acum2_sigo:
840   RCALL incremento_contador_verticalr20
841   BRCC octavo_bit_acum2_sigo ;Acum2.0
842   SBR ACUM2, B0
843   ;Acá termina primera parte de acum2 y empieza primera parte de acum3
844 octavo_bit_acum2_sigo:
845   RCALL incremento_contador_verticalr20
846   BRCC segundo_bit_acum3_sigo ;Acum3.6
847   SBR ACUM3, B6
848 segundo_bit_acum3_sigo:
849   RCALL incremento_contador_verticalr20
850   BRCC cuarto_bit_acum3_sigo ;Acum3.4
851   SBR ACUM3, B4
852   ;Acá termina r20
853 cuarto_bit_acum3_sigo:
854   RCALL incremento_contador_verticalr21
855   BRCC sexto_bit_acum3_sigo ;Acum3.2
856   SBR ACUM3, B2
857 sexto_bit_acum3_sigo:
858   RCALL incremento_contador_verticalr21
859   BRCC octavo_bit_acum3_sigo ;Acum3.0
860   SBR ACUM3, B0
861   ;Acá termina primera parte de acum3 y empieza primera parte de acum4
862 octavo_bit_acum3_sigo:
863   RCALL incremento_contador_verticalr21
864   BRCC segundo_bit_acum4_sigo ;Acum4.6
```

```
  SBR ACUM4, B6
segundo_bit_acum4_sigo:
  RCALL incremento_contador_verticalr21
  BRCC chequeo_fila2 ;Acum4.4
  SBR ACUM4, B4
chequeo_fila2:
  CPI CONT_H, 0x09
  BRNE fin_columna2
; Esta parte corresponde a si la columna es la nueve (hay que hacer otras cosas con los acumuladores).
es_columna9:
  RCALL incremento_contador_verticalr21
  BRCC sexto_bit_acum4_sigo_c9 ;Acum4.2
  SBR ACUM4, B2
sexto_bit_acum4_sigo_c9:
  RCALL incremento_contador_verticalr21
  BRCC salto_linea_c9 ;Acum4.0
  SBR ACUM4, B0
salto_linea_c9: ;Linea que no tiene datos
  RCALL incremento_contador_verticalr21 ;Roto un bit que no me interesa
  RCALL incremento_contador_verticalr21
  BRCC segundo_bit_acum5_sigo_c9 ;Acum5.6
  SBR ACUM5, B6
segundo_bit_acum5_sigo_c9:
  RCALL incremento_contador_verticalr22
  BRCC cuarto_bit_acum5_sigo_c9 ;Acum5.4
  SBR ACUM5, B4
cuarto_bit_acum5_sigo_c9:
  RCALL incremento_contador_verticalr22
  BRCC sexto_bit_acum5_sigo_c9 ;Acum5.2
  SBR ACUM5, B2
sexto_bit_acum5_sigo_c9:
  RCALL incremento_contador_verticalr22
  BRCC octavo_bit_acum5_sigo_c9 ;Acum5.0
  SBR ACUM5, B0
octavo_bit_acum5_sigo_c9:
  RCALL incremento_contador_verticalr22
  BRCC segundo_bit_acum6_sigo_c9 ;Acum6.6
  SBR ACUM6, B6
segundo_bit_acum6_sigo_c9:
  RCALL incremento_contador_verticalr22
  BRCC fin_columna2 ;Acum6.4
  SBR ACUM6, B4
;Se terminó la lectura de la columna, ahora hay que guardar los valores leidos en los acumuladores en SRAM
;Guardo los valores que ya fueron completados en memoria ram.
;A continuación reviso que no esté en tipo 1 sección D
fin_columna2:
  CPI CONT_H, 0x01 ;Si es la segunda columna, tengo que guardar acum2 en length, sino en sram
  BRNE guardo_en_ram
  MOV LENGTH, ACUM2
  INC LENGTH
guardo_en_ram:
  RCALL guardo_acum1
  RCALL guardo_acum2
  RCALL guardo_acum3
  CPI CONT_H, 0x09
  BRNE empiezo_columna_3 ;Si no es la columna 9, no guardo acumuladores 4 y 5
  RCALL guardo_acum4
  RCALL guardo_acum5
  ;Empiezo columna 3
empiezo_columna_3:
  INC CONT_H
  RET

; TERCER MÉTODO DE LECTURA
leo_tipo_2A:
  ROL R20 ;En c ahora tengo el primer bit
  BRCC tercer_bit_acum1_sigo_c3 ;Acum1.5
  SBR ACUM1, B5
tercer_bit_acum1_sigo_c3:
  RCALL incremento_contador_verticalr20
  BRCC primer_bit_acum1_sigo_c3 ;Acum1.7
  SBR ACUM1, B7
  ;Acá termina primera parte de acum1 y empieza primera parte de acum2
primer_bit_acum1_sigo_c3:
  RCALL incremento_contador_verticalr20
  BRCC septimo_bit_acum2_sigo_c3 ;Acum2.1
  SBR ACUM2, B1
septimo_bit_acum2_sigo_c3:
  RCALL incremento_contador_verticalr20
  BRCC quinto_bit_acum2_sigo_c3 ;Acum2.3
  SBR ACUM2, B3
quinto_bit_acum2_sigo_c3:
  RCALL incremento_contador_verticalr20
  BRCC tercer_bit_acum2_sigo_c3 ;Acum2.5
  SBR ACUM2, B5
tercer_bit_acum2_sigo_c3:
  RCALL incremento_contador_verticalr20
  BRCC primer_bit_acum2_sigo_c3 ;Acum2.7
  SBR ACUM2, B7
  ;Acá termina primera parte de acum2 y empieza primera parte de acum3
primer_bit_acum2_sigo_c3:
  RCALL incremento_contador_verticalr20
  BRCC septimo_bit_acum3_sigo_c3 ;Acum3.1
  SBR ACUM3, B1
septimo_bit_acum3_sigo_c3:
  RCALL incremento_contador_verticalr20
  BRCC quinto_bit_acum3_sigo_c3 ;Acum3.3
  SBR ACUM3, B3
```

```
963   ;Acá termina r20
964 quinto_bit_acum3_sigo_c3:
965   RCALL incremento_contador_verticalr21
966   BRCC tercer_bit_acum3_sigo_c3 ;Acum3.5
967   SBR ACUM3, B5
968 tercer_bit_acum3_sigo_c3:
969   RCALL incremento_contador_verticalr21
970   BRCC primer_bit_acum3_sigo_c3 ;Acum3.7
971   SBR ACUM3, B7
972   ;Acá termina primera parte de acum3 y empieza primera parte de acum4
973 primer_bit_acum3_sigo_c3:
974   RCALL incremento_contador_verticalr21
975   BRCC septimo_bit_acum4_sigo_c3 ;Acum4.1
976   SBR ACUM4, B1
977 septimo_bit_acum4_sigo_c3:
978   RCALL incremento_contador_verticalr21
979   BRCC chequeo_fila3 ;Acum4.3
980   SBR ACUM4, B3
981 ;Se terminó la lectura de la tercer columna, acá se debería esperar a que se vuelvan a llenar R20, R21 y R22.
982 ;Empiezo cuarta columna. Reviso que no sea tipo 2 sección C
983 chequeo_fila3:
984   CPI CONT_H, 0xA
985   BRNE fin_columna3
986 ; Esta parte corresponde a si la columna es la diez (hay que hacer otras cosas con los acumuladores).
987 es_columna10:
988   RCALL incremento_contador_verticalr21
989   BRCC tercer_bit_acum4_sigo_c10 ;Acum4.5
990   SBR ACUM4, B5
991 tercer_bit_acum4_sigo_c10:
992   RCALL incremento_contador_verticalr21
993   BRCC salto_linea_c10 ;Acum4.7
994   SBR ACUM4, B7
995   ;Acá termina primera parte de Acum4, salto de línea y empieza acum5
996 salto_linea_c10: ;Linea que no tiene datos
997   RCALL incremento_contador_verticalr21 ;Roto un bit que no me interesa
998   RCALL incremento_contador_verticalr21
999   BRCC septimo_bit_acum5_sigo_c10 ;Acum5.1
1000  SBR ACUM5, B1
1001 septimo_bit_acum5_sigo_c10:
1002  RCALL incremento_contador_verticalr22
1003  BRCC quinto_bit_acum5_sigo_c10 ;Acum5.3
1004  SBR ACUM5, B3
1005 quinto_bit_acum5_sigo_c10:
1006  RCALL incremento_contador_verticalr22
1007  BRCC tercer_bit_acum5_sigo_c10 ;Acum5.5
1008  SBR ACUM5, B5
1009 tercer_bit_acum5_sigo_c10:
1010  RCALL incremento_contador_verticalr22
1011  BRCC primer_bit_acum5_sigo_c10 ;Acum5.7
1012  SBR ACUM5, B7
1013  ;Acá termina primera parte de Acum5, empieza acum6
1014 primer_bit_acum5_sigo_c10:
1015  RCALL incremento_contador_verticalr22
1016  BRCC septimo_bit_acum6_sigo_c10 ;Acum6.1
1017  SBR ACUM6, B1
1018 septimo_bit_acum6_sigo_c10:
1019  RCALL incremento_contador_verticalr22
1020  BRCC fin_columna3 ;Acum6.3
1021  SBR ACUM6, B3
1022 fin_columna3:
1023  INC CONT_H
1024  RET
1025
1026 ; CUARTO MÉTODO DE LECTURA
1027 leo_tipo_2B:
1028  ROL R20 ;En c ahora tengo el primer bit
1029  BRCC cuarto_bit_acum1_sigo_c3 ;Acum1.4
1030  SBR ACUM1, B4
1031 cuarto_bit_acum1_sigo_c3:
1032  RCALL incremento_contador_verticalr20
1033  BRCC segundo_bit_acum1_sigo_c3  ;Acum1.6
1034  SBR ACUM1, B6
1035  ;Acá termina primera parte de acum1 y empieza primera parte de acum2
1036 segundo_bit_acum1_sigo_c3:
1037  RCALL incremento_contador_verticalr20
1038  BRCC octavo_bit_acum2_sigo_c3 ;Acum2.0
1039  SBR ACUM2, B0
1040 octavo_bit_acum2_sigo_c3:
1041  RCALL incremento_contador_verticalr20
1042  BRCC sexto_bit_acum2_sigo_c3 ;Acum2.2
1043  SBR ACUM2, B2
1044 sexto_bit_acum2_sigo_c3:
1045  RCALL incremento_contador_verticalr20
1046  BRCC cuarto_bit_acum2_sigo_c3 ;Acum2.4
1047  SBR ACUM2, B4
1048 cuarto_bit_acum2_sigo_c3:
1049  RCALL incremento_contador_verticalr20
1050  BRCC segundo_bit_acum2_sigo_c3 ;Acum2.6
1051  SBR ACUM2, B6
1052  ;Acá termina primera parte de acum2 y empieza primera parte de acum3
1053 segundo_bit_acum2_sigo_c3:
1054  RCALL incremento_contador_verticalr20
1055  BRCC octavo_bit_acum3_sigo_c3 ;Acum3.0
1056  SBR ACUM3, B0
1057 octavo_bit_acum3_sigo_c3:
1058  RCALL incremento_contador_verticalr20
1059  BRCC sexto_bit_acum3_sigo_c3 ;Acum3.2
1060  SBR ACUM3, B2
```

```
1061    ;Acá termina r20
1062  sexto_bit_acum3_sigo_c3:
1063    RCALL incremento_contador_verticalr21
1064    BRCC cuarto_bit_acum3_sigo_c3 ;Acum3.4
1065    SBR ACUM3, B4
1066  cuarto_bit_acum3_sigo_c3:
1067    RCALL incremento_contador_verticalr21
1068    BRCC segundo_bit_acum3_sigo_c3 ;Acum3.6
1069    SBR ACUM3, B6
1070    ;Acá termina primera parte de acum3 y empieza primera parte de acum4
1071  segundo_bit_acum3_sigo_c3:
1072    RCALL incremento_contador_verticalr21
1073    BRCC octavo_bit_acum4_sigo_c3 ;Acum4.0
1074    SBR ACUM4, B0
1075  octavo_bit_acum4_sigo_c3:
1076    RCALL incremento_contador_verticalr21
1077    BRCC chequeo_fila4 ;Acum4.2
1078    SBR ACUM4, B2
1079  ;Se terminó la lectura de la cuarta columna. Acá se debería esperar a que se vuelvan a llenar R20, R21 y R22.
1080  ;Empiezo quinta columna. Reviso que no sea tipo 2 sección D.
1081  chequeo_fila4:
1082    CPI CONT_H, 0x0B
1083    BRNE fin_columna4
1084  ; Esta parte corresponde a si la columna es la once (hay que hacer otras cosas con los acumuladores).
1085  es_columna11:
1086    RCALL incremento_contador_verticalr21
1087    BRCC cuarto_bit_acum4_sigo_c11 ;Acum4.4
1088    SBR ACUM4, B4
1089  cuarto_bit_acum4_sigo_c11:
1090    RCALL incremento_contador_verticalr21
1091    BRCC salto_linea_c11 ;Acum4.6
1092    SBR ACUM4, B6
1093    ;Acá termina primera parte de Acum4, salto de línea y empieza acum5
1094  salto_linea_c11: ;Linea que no tiene datos
1095    RCALL incremento_contador_verticalr21 ;Roto un bit que no me interesa
1096    RCALL incremento_contador_verticalr21
1097    BRCC octavo_bit_acum5_sigo_c11 ;Acum5.0
1098    SBR ACUM5, B0
1099  octavo_bit_acum5_sigo_c11:
1100    RCALL incremento_contador_verticalr22
1101    BRCC sexto_bit_acum5_sigo_c11 ;Acum5.2
1102    SBR ACUM5, B2
1103  sexto_bit_acum5_sigo_c11:
1104    RCALL incremento_contador_verticalr22
1105    BRCC cuarto_bit_acum5_sigo_c11 ;Acum5.4
1106    SBR ACUM5, B4
1107  cuarto_bit_acum5_sigo_c11:
1108    RCALL incremento_contador_verticalr22
1109    BRCC segundo_bit_acum5_sigo_c11 ;Acum5.6
1110    SBR ACUM5, B6
1111    ;Acá termina primera parte de Acum5, empieza acum6
1112  segundo_bit_acum5_sigo_c11:
1113    RCALL incremento_contador_verticalr22
1114    BRCC octavo_bit_acum6_sigo_c11 ;Acum6.0
1115    SBR ACUM6, B0
1116  octavo_bit_acum6_sigo_c11:
1117    RCALL incremento_contador_verticalr22
1118    BRCC fin_columna4 ;Acum6.2
1119    SBR ACUM6, B2
1120  fin_columna4:
1121    CPI CONT_H, 0x0B
1122    BRNE guardo_parcial_sram ;Por como viene el orden, primero se guardan acum6 y acum5
1123    RCALL guardo_acum6
1124    RCALL guardo_acum5
1125  guardo_parcial_sram:
1126    RCALL guardo_acum4
1127    RCALL guardo_acum3
1128    RCALL guardo_acum2
1129    INC CONT_H
1130    RET
1131
1132  ; MÉTODOS AUXILIARES DE LA FUNCIÓN DE DECODIFICACIÓN:
1133  ;Rollea R20 a izquierda
1134  incremento_contador_verticalR20:
1135    ROL R20
1136    RET
1137  ;Rollea R21 a izquierda
1138  incremento_contador_verticalR21:
1139    ROL R21
1140    RET
1141  ;Rollea R22 a izquierda
1142  incremento_contador_verticalR22:
1143    ROL R22
1144    RET
1145  ;Incrementa contador horizontal (variable de control)
1146  incremento_contador_horizontal:
1147    INC CONT_H
1148    RET
1149  ;Guarda ACUM1 en SRAM, lo limpia e incrementa la cantidad de letras leidas.
1150  guardo_acum1:
1151    MOV R14, ACUM1
1152    ST X+, R14
1153    EOR ACUM1, ACUM1
1154    INC CANT_LETRAS_LEIDAS
1155    RET
1156  ;Guarda ACUM2 en SRAM, lo limpia e incrementa la cantidad de letras leidas.
1157  guardo_acum2:
1158    MOV R14, ACUM2
```

```
1159    ST X+, R14
1160    EOR ACUM2, ACUM2
1161    INC CANT_LETRAS_LEIDAS
1162    RET
1163  ;Guarda ACUM3 en SRAM, lo limpia e incrementa la cantidad de letras leidas.
1164  guardo_acum3:
1165    MOV R14, ACUM3
1166    ST X+, R14
1167    EOR ACUM3, ACUM3
1168    INC CANT_LETRAS_LEIDAS
1169    RET
1170  ;Guarda ACUM4 en SRAM, lo limpia e incrementa la cantidad de letras leidas.
1171  guardo_acum4:
1172    MOV R14, ACUM4
1173    ST X+, R14
1174    EOR ACUM4, ACUM4
1175    INC CANT_LETRAS_LEIDAS
1176    RET
1177  ;Guarda ACUM5 en SRAM, lo limpia e incrementa la cantidad de letras leidas.
1178  guardo_acum5:
1179    MOV R14, ACUM5
1180    ST X+, R14
1181    EOR ACUM5, ACUM5
1182    INC CANT_LETRAS_LEIDAS
1183    RET
1184  ;Guarda ACUM6 en SRAM, lo limpia e incrementa la cantidad de letras leidas.
1185  guardo_acum6:
1186    MOV R14, ACUM6
1187    ST X+, R14
1188    EOR ACUM6, ACUM6
1189    INC CANT_LETRAS_LEIDAS
1190    RET
```

Referencias

[1] Muhammad Ali Mazidi, Zarmad Naimi, Sepehr Naimi. *The AVR Microcontroller and Embedded System Using Assembly and C*. Prentice Hall.

[2] Atmel 8-bit Microcontroller 4/8/16/32Kbytes In-system Programmable Flash Datasheet.

[3] Atmel AVR 8-bit Instruction Set Manual

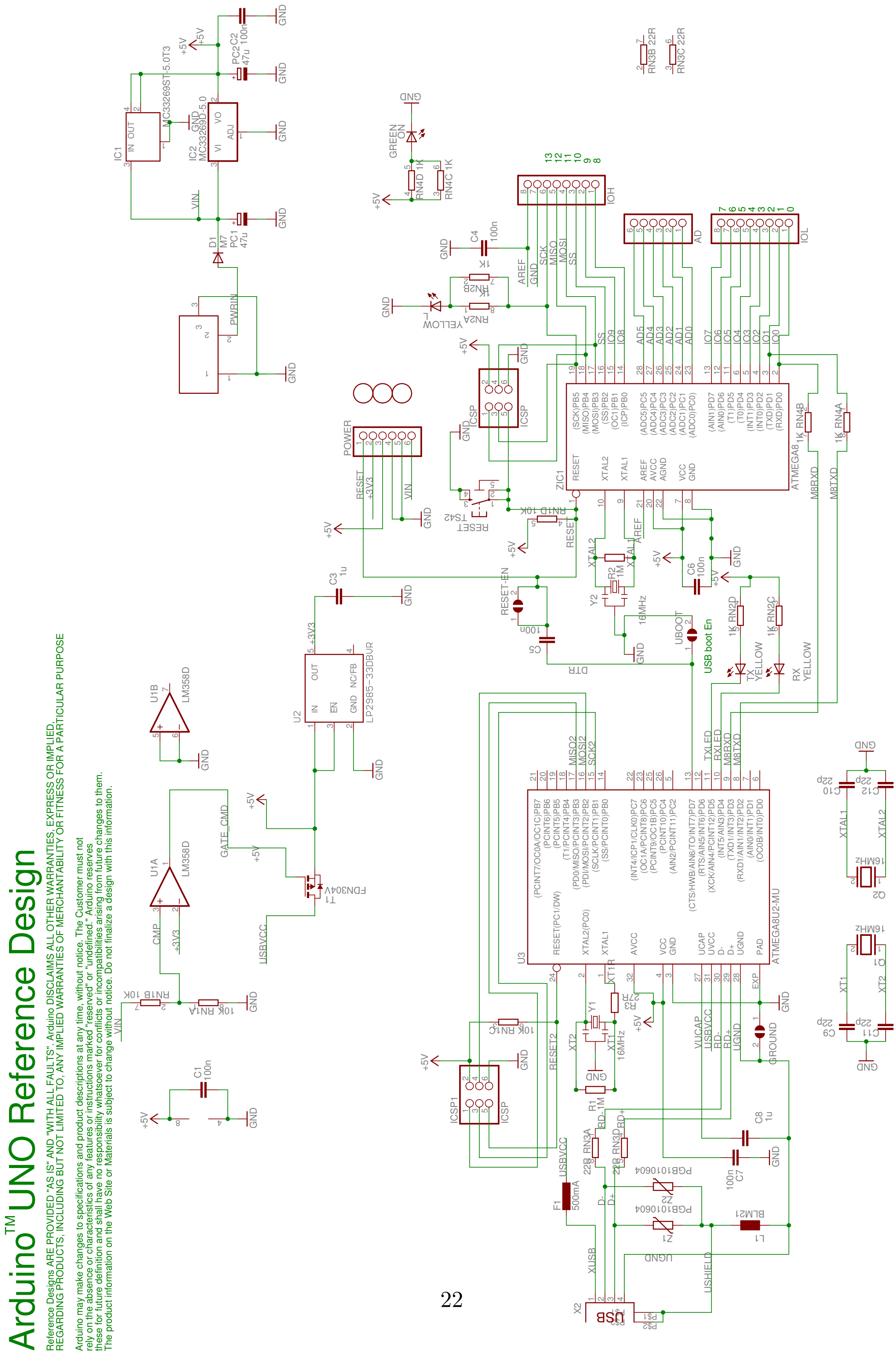
[4] `https://en.wikipedia.org/wiki/QRcode`

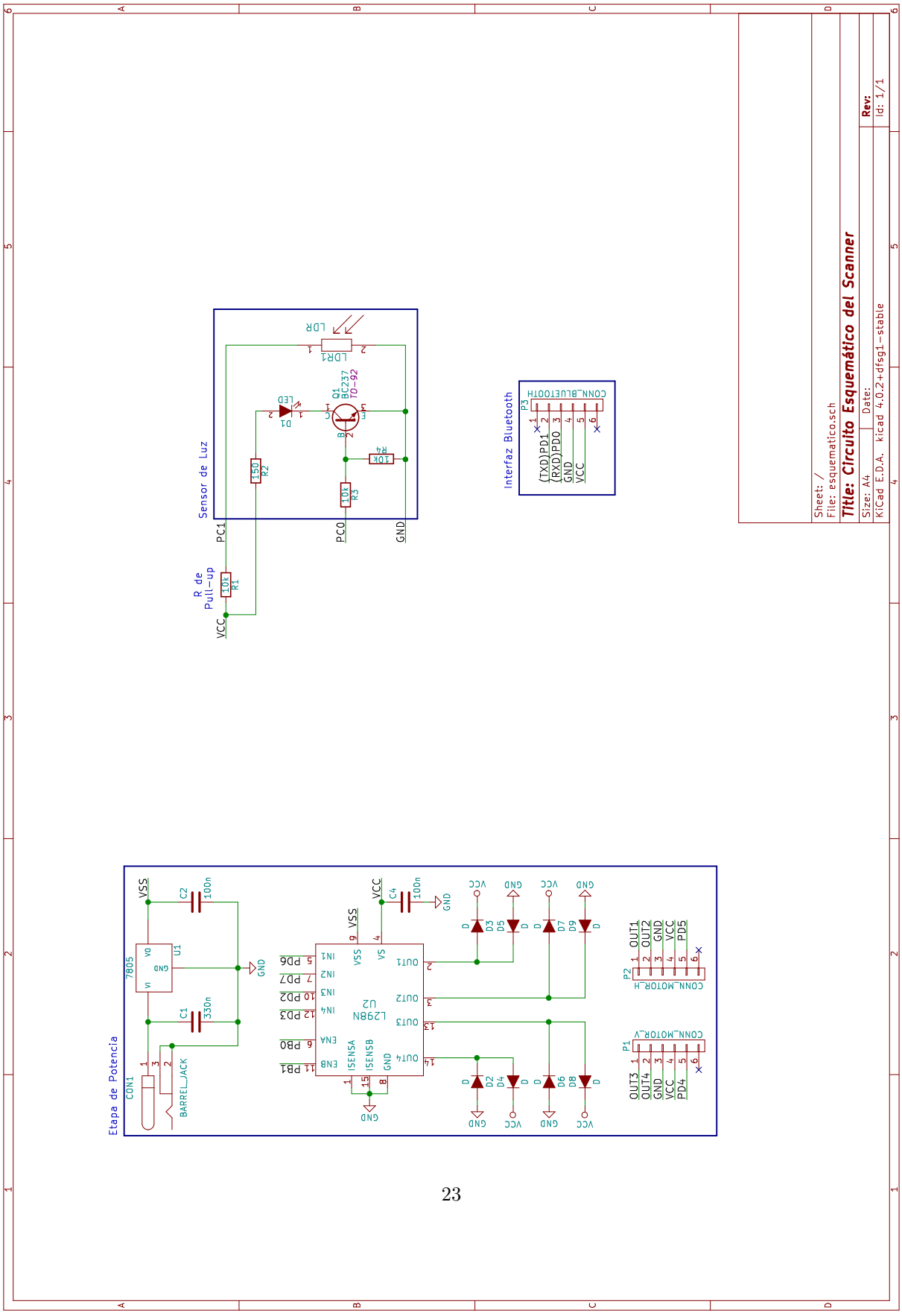
[5] `http://www.ams.org/samplings/feature-column/fc-2013-02`

[6] `https://en.wikipedia.org/wiki/Lego_Mindstorms_NXT`

Apéndice

A continuación se muestran los esquemáticos de la placa de Arduino UNO y del circuito diseñado para la implementación del escaneo.


Arduino™ UNO Reference Design
Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS". Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED,
REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE
Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not
rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves
these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.
The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.


Sensor de Luz
LDR
LDR1
Q1
BC237
TO-92
LED
D1
150
R2
10K
R4
10K
R3
PC1
PC0
GND
R de
Pull-up
10K
R1
VCC
Interfaz Bluetooth
P3
CONN_BLUETOOTH
(TXD)PD1
(RXD)PD0
GND
VCC
Sheet: /
File: esquematico.sch
Title: Circuito Esquemático del Scanner
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.2+dfsg1-stable
Id: 1/1
Etapa de Potencia
CON1
BARREL_JACK
C1
330n
C2
100n
U1
7805
VSS
GND
VCC
C4
100n
L298N
U2
VSS
VS
IN1
IN2
IN3
IN4
ENA
ENB
ISENSA
ISENSB
OUT1
OUT2
OUT3
OUT4
PB0
PB1
PD2
PD3
PD6
PD7
D1
D2
D3
D4
D5
D6
D7
D8
D9
P1
CONN_MOTOR_V
P2
CONN_MOTOR_H
PD4
PD5