

**(86.07) LABORATORIO DE MICROPROCESADORES**

*Detector de humedad de filamentos empleados en impresión 3D*

<b>Profesor:</b>	<b>Ing. Guillermo Campiglio</b>
<b>Cuatrimestre / Año:</b>	<b>1ºC 2019</b>
<b>Turno de clases prácticas:</b>	<b>Miércoles (19 a 22hs)</b>
<b>Jefe de Trabajos Prácticos:</b>	<b>Ing. Ricardo Arias</b>
<b>Docente guía:</b>	<b>Ing. Ricardo Arias</b>

Autores			Seguimiento del proyecto							
Nombre	Apellido	Padrón								
Kevin	Michalewicz	100978								
Rosario	Szuplat	100798								

**Observaciones:**

---



---



---



---



---



---



---



---



---

Fecha de aprobación		

Firma J.T.P.

COLOQUIO	
Nota final	
Firma Profesor	

# Índice

1. Introducción.....	2
2. Objetivos.....	2
3. Descripción del Proyecto.....	3
4. Hardware .....	4
4.1 Diagrama de Bloques.....	4
4.2 Esquemático .....	4
5. Software.....	8
5.1 Diagrama de flujo .....	8
5.2 Explicación del código .....	9
6. Resultados.....	10
7. Conclusiones.....	12
8. Apéndice.....	12

## 1. Introducción

El filamento de impresión 3D es un material plástico higroscópico. Hay muchos tipos disponibles con diferentes propiedades que, a su vez, requieren diversas temperaturas para imprimir. Además, los diámetros de los filamentos están estandarizados a valores de 1,75 mm.

Se ha observado<sup>1</sup> que si estos adquieren un cierto nivel de humedad, el producto que se obtiene al imprimir en 3D es de menor calidad o bien posee imperfecciones. Si bien existen dispositivos capaces de secar los filamentos, por sus elevados precios en el país se suelen emplear hornos o microondas por largos períodos de tiempo para asegurar baja humedad. Sería de gran ayuda conocer de antemano si una cierta porción de filamento está lista para ser usada, o si su elevado nivel de humedad provocará resultados no deseados en la pieza final.

## 2. Objetivos

Por lo dicho en la sección anterior, el proyecto propuesto consiste en un dispositivo sencillo capaz de notificar al usuario si el filamento de impresión 3D de 1,75 mm de *nylon* (poliamida) está húmedo o en condiciones de ser empleado. Para lograr esto, se

---

<sup>1</sup> “*Thermal Analysis Application Brief Measurement of Moisture Effects on the Mechanical Properties of 66 Nylon*”. TA Instruments - Thermal Analysis & Rheology -

estudiaron filamentos en diversas condiciones conocidas a fin de determinar experimentalmente a partir de qué nivel de humedad el producto de la impresión deja de ser el esperado. Por último, se incorporó una interfaz interactiva para que el usuario pueda acceder a las instrucciones de uso o leer los resultados deseados.

### 3. Descripción del Proyecto

Se usó un sistema de placas conductoras plano-paralelas con acrílico entre las mismas. Realizando un pequeño orificio en el centro de este último material, pueden insertarse piezas de filamento. Esta construcción no es otra cosa que un capacitor. Se puede observar en la *Figura 3* (página 6). Su capacidad varía al introducir filamentos de diversos niveles de humedad debido a que la permitividad relativa aumenta con el grado de humedad absorbido por el plástico. Esta información fue obtenida en el *paper* “A Polyimide-Based Capacitive Humidity Sensor”<sup>2</sup>.

Este capacitor forma parte de un circuito RC, donde la resistencia es conocida y fija, que determina el período en la salida de un monoestable. Dado que el tiempo en alto de la señal es proporcional al producto del valor de la resistencia con el del capacitor, se mide el valor medio de esta señal. Es decir, si aumenta la capacidad, aumenta esta tensión. Como la geometría del capacitor es fija, queda directamente vinculada a la permitividad.

La permitividad  $\epsilon$  de un dieléctrico es una cantidad compleja; se puede probar que la parte real está directamente relacionada con la humedad. Se mide la capacidad del sistema en ausencia del filamento ( $C_{aire}$ ) y luego, sabiendo que la permitividad es proporcional a  $C$ , se compara la capacidad medida – en presencia del filamento – con  $C_{aire}$ .

Se toma el valor medio de la tensión a la salida del monoestable (analógica) con un conversor analógico digital de 24 bits, el cual le devuelve dicho valor al microcontrolador de manera digital. No se usa el ADC del microprocesador porque su resolución no es suficiente para distinguir las pequeñas diferencias de tensión con las que se trabaja. Notar que la precisión es igual al cociente de la tensión de alimentación y 2 elevado a una potencia igual al número de bits que usa el ADC.

---

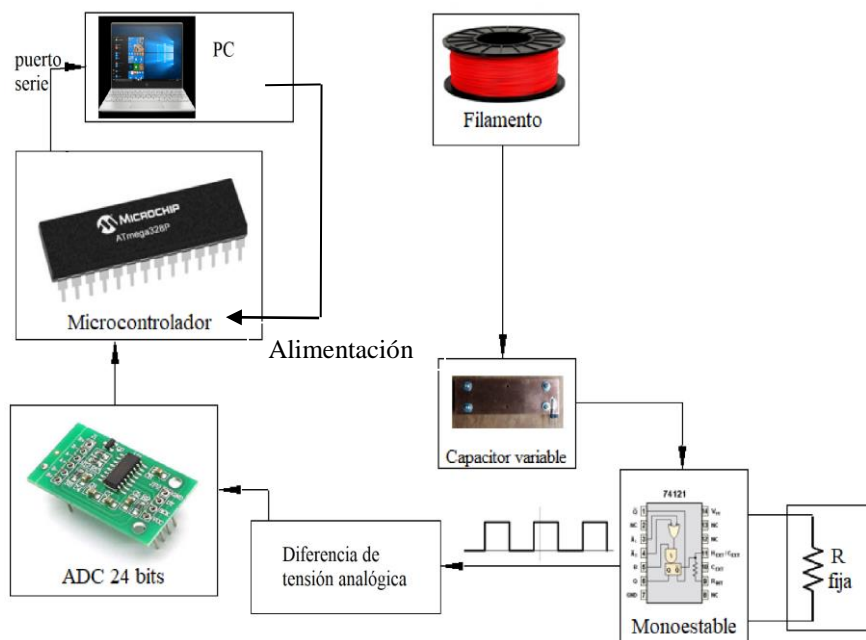
<sup>2</sup> Peter J. Schubert Y Joseph H. Nevin, Member, IEEE - Institute of Electrical and Electronics Engineers – *IEEE Transactions on Electron Devices*, VOL. ED-32, NO. 7, JULY 1985

La comunicación con el usuario se da a través del puerto serie, eligiendo qué resultados desea y viéndolos en la pantalla de la computadora. Esto es un reemplazo del display que se iba a utilizar inicialmente, pero con el beneficio de la bidireccionalidad de la comunicación.

## 4. Hardware

### 4.1. Diagrama de Bloques

En la *Figura 1* se implementa un diagrama de bloques del prototipo, donde cada bloque representa una de las etapas principales del diseño.



*Figura 1: diagrama de bloques.*

### 4.2. Esquemático y Descripción del Circuito Utilizado

En la *Figura 2* se presenta el circuito esquemático del circuito eléctrico implementado, detallando su conexión con la placa *Arduino Uno*, donde se encuentra el microprocesador.

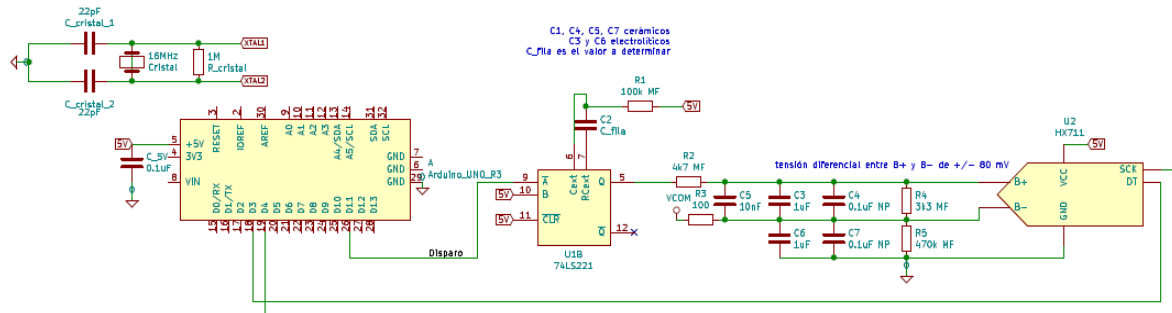


Figura 2: circuito esquemático.

A continuación, se detallan los elementos utilizados y sus valores y se justifica su uso.

Las hojas de datos de los integrados se adjuntan en el *Apéndice*.

Se eligió la placa *Arduino Uno* debido a que, además de incluir un microprocesador *ATmega328p* (uno de los indicados para la realización del proyecto), cuenta con una cantidad de puertos de entrada/salida suficiente, pero no excesiva, para el trabajo a realizar. También, la misma permite una conexión USB con la computadora, lo cual facilita la alimentación del circuito (5 V).

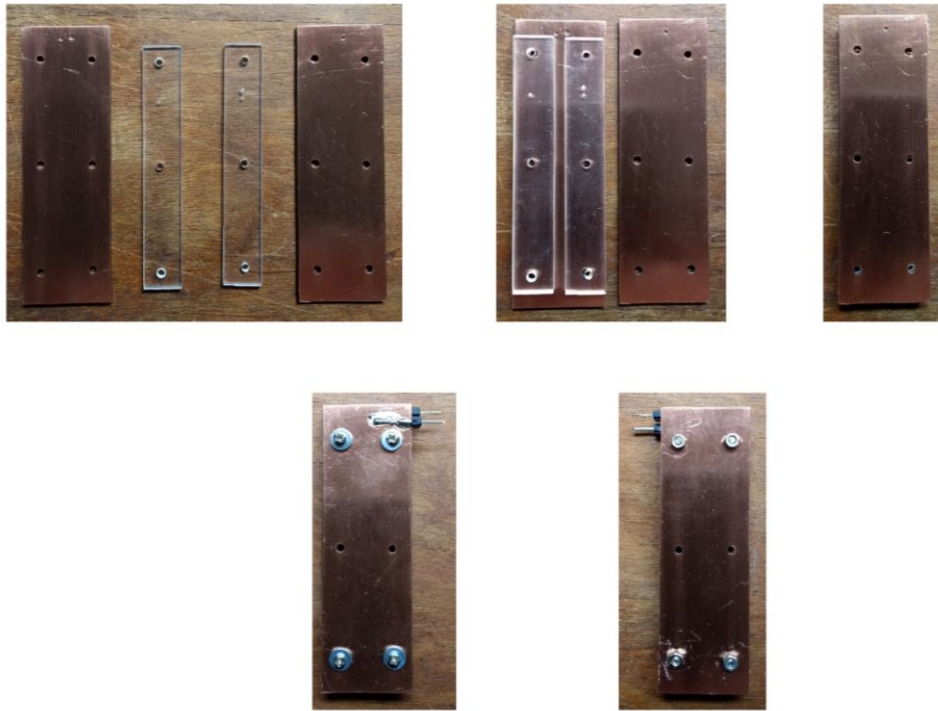
Las funciones del microprocesador en este proyecto son:

- Generar una señal cuadrada en el pin 3 del puerto B.
- Generar señal de *clock* en el pin 4 del puerto D.
- Guardar en memoria los valores recibidos en el pin 3 del puerto D.
- Promediar los valores recibidos.
- Recibir instrucciones por puerto serie.
- Devolver la información solicitada por puerto serie con una frecuencia razonable.

Las mismas se desarrollan en detalle en la descripción del *Software* implementado.

El monoestable utilizado es uno de los dos incluidos en el *chip* 74LS221. El mismo recibe una señal de disparo desde el pin 3 del puerto B del microprocesador, el cual coincide con el pin digital 11 del *Arduino*. Esta señal es cuadrada y su período fue configurado de manera acorde a la carga RC: es necesario que el mismo permita tiempos en alto y en bajo apreciables en la señal de salida para poder medir su valor medio correctamente y que éste se encuentre dentro del rango medible por el ADC. Se trabajó de manera tal que el tiempo en alto sea igual al tiempo en bajo. El monoestable se conecta al circuito RC y este, mediante un capacitor de 0,1  $\mu\text{F}$ , a la alimentación (5 V) que provee el *Arduino*.

El circuito RC está formado por una resistencia fija de  $100\text{ k}\Omega$  de *metal film* y el capacitor construido manualmente, tal como se explicó anteriormente. En la *Figura 3* se muestran fotos de como se realizó el montaje de éste.



*Figura 3: montaje del capacitor que permite introducir el filamento en su interior.*

El tiempo en alto de la señal de salida del monoestable es aproximadamente el producto de  $R$ ,  $C$  y  $0,7$ . Esta relación fue utilizada para verificar el correcto funcionamiento del circuito conectando un valor de capacidad conocido. También, para obtener los valores de capacidad con y sin filamento y verificar así la relación esperada. Se obtuvieron  $78,29$  y  $77,14\text{ pF}$  respectivamente. Esta medición se realizó con un filamento seco.

Luego, se adecuó esta señal de salida del monoestable para maximizar la precisión con la que la convierte el ADC. En primer lugar, se implementó un filtro RC pasabajos a fin de dejar pasar sólo la componente continua de la señal, única parte importante ya que es la proporcional al tiempo en alto del monoestable, quien a su vez es proporcional a la capacidad a medir. El ADC lee la tensión diferencial entre los pines  $B+$  y  $B-$ , la cual debe estar en el rango de  $\pm 80\text{ mV}$ . Además, dado que la alimentación es de  $5\text{ V}$ , los valores de continua medibles en el circuito están entre  $0$  y  $5\text{ V}$ . Como la tensión diferencial que convierte el ADC debe estar sobre un nivel de continua entre  $1,2$  y  $4,9\text{ V}$ , tal como especifica la hoja de datos, se usó el divisor resistivo  $R_4$ - $R_5$ , que fija unos  $1,9\text{ V}$  de pedestal. En la *Ecuación 1* se muestra el cálculo de la tensión diferencial  $V_d$  a

partir de la tensión en B+  $V_{B+}$ . Ésta se usó para verificar que los valores enviados al ADC eran correctos.

$$V_d = V_{B+} \frac{R_4}{R_5 + R_4} \quad (1)$$

Reemplazando  $R_4$  y  $R_5$  por sus valores:

$$V_d = V_{B+} \frac{3,3}{470 + 3,3}$$

Por otro lado, se usa el paralelo de los capacitores de 0,1 y 1  $\mu\text{F}$  para filtrar altas y bajas frecuencias respectivamente, disminuyendo el ruido.

El ADC mide la tensión diferencial analógica y la convierte a un valor digital. Cuando termina una conversión, es decir, cada aproximadamente 100 ms, su salida DT cambia de '1' a '0'. El microprocesador detecta este flanco y comienza a recibir los datos. Para esto, es necesario que el MCU genere una señal de clock en el pin SCK del ADC. Con cada pulso del mismo, el ADC envía un bit de datos por el DT. Cabe aclarar que el ADC tiene una ganancia, por lo que el valor digitalizado es 32 veces mayor que el analógico. Para verificar que el valor de tensión digital (en base 10) dado por el ADC  $V_{ADC}$  es correcto se utilizó la Ecuación 2 y se comparó el valor de  $V_{B+}$  así calculado con el obtenido previamente.

$$V_{B+} = 5 V \cdot \frac{V_{ADC}}{2^{24} - 1} \cdot \frac{470 + 3,3}{3,3} \cdot \frac{1}{32} \quad (2)$$

Se puede probar que la tensión  $V_{B+}$  es proporcional a la capacidad del circuito RC usado. Pensando que introducir el filamento equivale a tener dos capacitores en paralelo,  $V_{B+}$  es proporcional a la suma de las capacidades con y sin filamento. Por lo tanto, la diferencia de tensión  $\Delta V_{B+}$  obtenida como la resta de  $V_{B+}$  con y sin introducir el filamento es proporcional a la capacidad con el mismo y esta es proporcional a la permitividad relativa del material. Es por esto que se concluye que un aumento en la tensión convertida por el ADC implica un aumento en la permitividad del filamento y se puede probar que, mientras más alta sea ésta, más húmedo se encuentra.

El precesamiento de los valores de tensión  $V_{B+}$  digitalizados que realiza el microprocesador se detalla en la siguiente sección.

## 5. Software

### 5.1. Diagrama de flujo

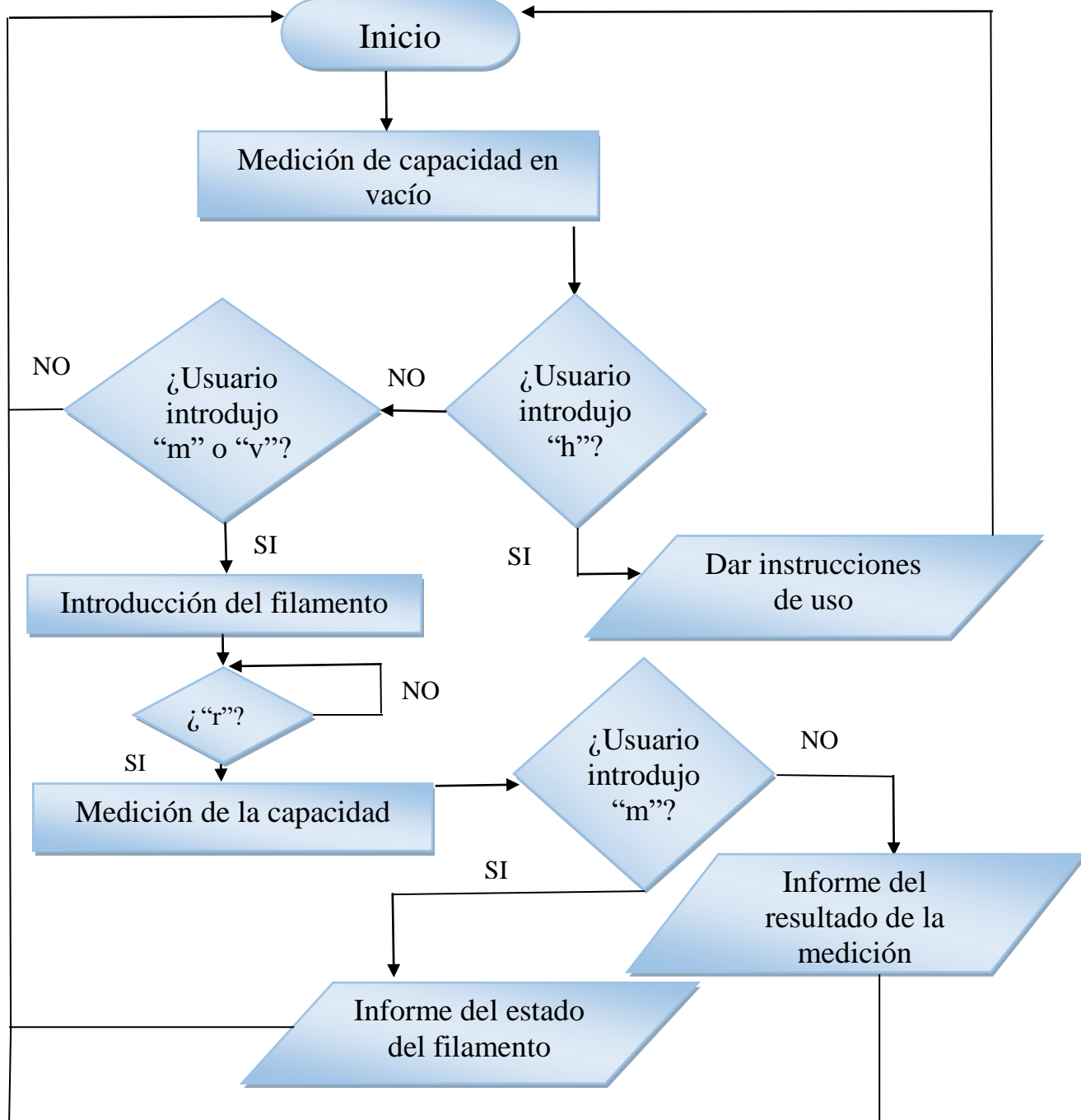


Figura 4: diagrama de flujo.

En la *Figura 4* se presenta un diagrama de flujo para sintetizar de manera clara el código implementado para lograr el funcionamiento del proyecto.



## 5.2. Explicación del Código Implementado

El código completo se adjunta en el *Apéndice*. A continuación, se hace una breve explicación del mismo.

Se implementó un código en *Assembly* que ejecuta las instrucciones mostradas en el diagrama de flujo de la *Figura 4*. El mismo se desarrolló en la plataforma *AtmelStudio 7*.

Inicialmente se lee por el pin 3 del puerto D los datos mandados por el DT del ADC, cada uno luego de un pulso de la señal de *clock*, que sale por el pin 4 del mismo puerto. Dado que el ADC es de 24 bits, se usan 3 bytes de memoria RAM para cada medición. Se dejan disponibles 24 posiciones de memoria para esto, es decir, se guardan simultáneamente hasta 8 mediciones: la novena pisa la primera y así sucesivamente. El último byte de las mediciones fluctúa considerablemente. Por lo tanto, tras cada medición, se realiza un promedio de las 8 que se tienen guardadas. (Ni bien se inicia el programa no se tienen 8 mediciones, por lo que los primeros promedios son desechados). Cada promedio se guarda en otra posición de memoria, la cual es leída cuando se desea transmitir por el puerto serie el valor de la medición más reciente.

Cuando el usuario introduce “m” o “v” en la terminal asociada al puerto serie, se imprime en la misma “Introduzca el Filamento” y se espera a que el usuario lo haga y presione “r”. Luego de esto, se vuelve a medir la tensión y, si el usuario había introducido “m”, se la compara con la medida al comienzo, es decir, en vacío y se imprime por pantalla un mensaje que indica si el filamento está o no húmedo. En el caso de que hubiera introducido “v”, se imprimen por pantalla el valor medido luego de introducir el filamento y los últimos 2 bytes – especificados entre paréntesis – del medido en vacío. Si el usuario introduce “h”, se imprime un manual de instrucciones. En todos los casos, luego de imprimir lo que corresponde, el programa vuelve a comenzar.

La estructura principal del programa es un registro llamado “eventos” cuyos bits se utilizan como *flags*. En la *Figura 5* se lo esquematiza.

D7	D6	D5	D4	D3	D2	D1	D0
(sin uso)	EVENTO_	EVENTO_	EVENTO_	EVENTO_	EVENTO_	EVENTO_	EVENTO_

	V	FILAMENTO	M	PROMEDIAR	1SEG	ADC_FIN	RX_SERIE
--	---	-----------	---	-----------	------	---------	----------

Figura 5: esquema del registro Eventos.

Cada bit pasa de ‘0’ a ‘1’ cuando sucede algo determinado, indicando qué rutina tiene que tener lugar:

Cuando entra un dato por el puerto serie, es decir, cuando el usuario presiona *enter*, se activa el *flag* EVENTO\_RX\_SERIE. Esto indica que debe ejecutarse la rutina “Ver\_si\_Medir\_o\_Valor\_o\_Ready”, la cual determina qué letra ingresó el usuario. Si se detectó “m” o “v” cambia a ‘1’ el *flag* EVENTO\_M o EVENTO\_V respectivamente. Si se detecta “h” llama a la rutina “Help\_TX”, la cual imprime la ayuda. En el caso en el que se ingrese “r”, verifica que esté en ‘1’ EVENTO\_M o EVENTO\_V ya que en caso contrario no tiene sentido la instrucción. Por lo tanto, si ambos *flags* están en ‘0’, no se ejecuta ninguna rutina y sigue esperando que se introduzca una instrucción válida.

Si se ingresó “r” estando en alto el *flag* EVENTO\_M, se activa EVENTO\_FILAMENTO, el cual indica que debe ejecutarse la rutina “Msj\_y\_Comparación”. La misma resta el valor medido luego de ingresar el filamento con el anterior y luego compara este resultado con el umbral determinado experimentalmente. Si es mayor, imprime un mensaje diciendo que está húmedo mientras que si es menor, que está seco. La justificación de esta relación se indicó en el *Inciso 5.2*.

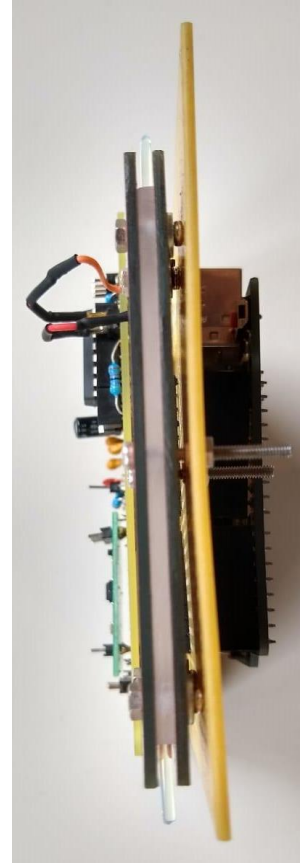
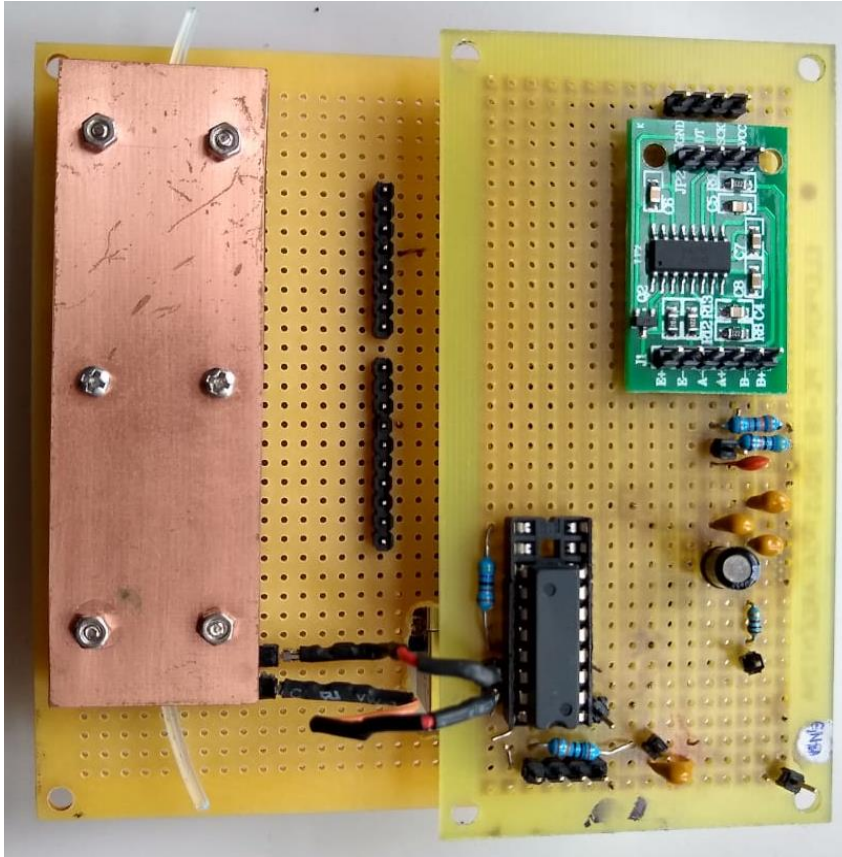
Por otro lado, EVENTO\_ADC\_FIN se activa cuando hay un flanco descendente en el DT del ADC e induce la ejecución de “Leer\_Bits”. Esta rutina guarda en RAM la información proveniente del ADC tal como se explicó anteriormente. Cada vez que se guarda una nueva muestra (3 *bytes*), se pone en alto EVENTO\_PROMEDIAR, produciendo la ejecución de “Promediar\_Muestras”.

EVENTO\_1SEG se activa cada vez que pasa un segundo (está controlado por un *timer*). Al suceder esto, se ejecuta “Leer\_ADC”, la cual sirve para transmitir el valor de promedio que se tiene en RAM en ese instante por el puerto serie.

En todos los casos, luego de ser utilizados, los *flags* se limpian a fin de poder ser usados nuevamente.

## 6. Resultados

En las *Figuras 6 y 7* se presenta una imagen del prototipo terminado y en las *Figuras 8 a 11* se muestra mediante capturas de pantalla el correcto funcionamiento del dispositivo en los casos en los que se introduce “h”, “m” o “v” antes de “r”, “r” después de “v” y “r” después de “m” respectivamente.



*Figuras 6 y 7: vistas superior y lateral del prototipo del proyecto.*



*Figura 8: funcionamiento con instrucción “h”.*

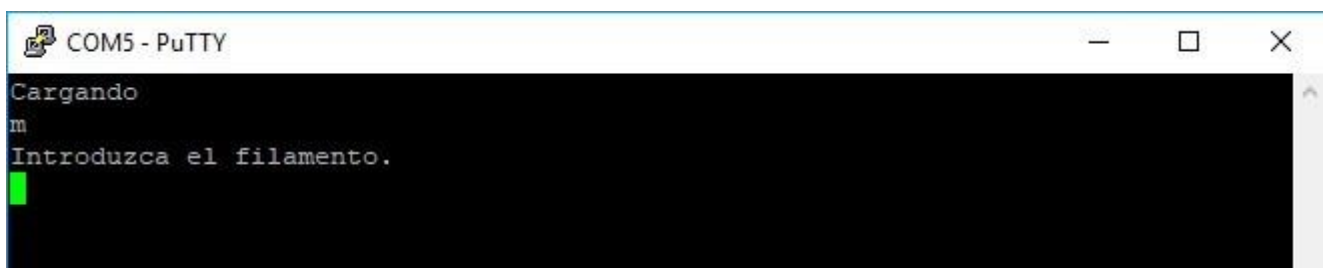


Figura 9: funcionamiento con instrucción “m” antes de introducir “r”. Es análogo para “v”.

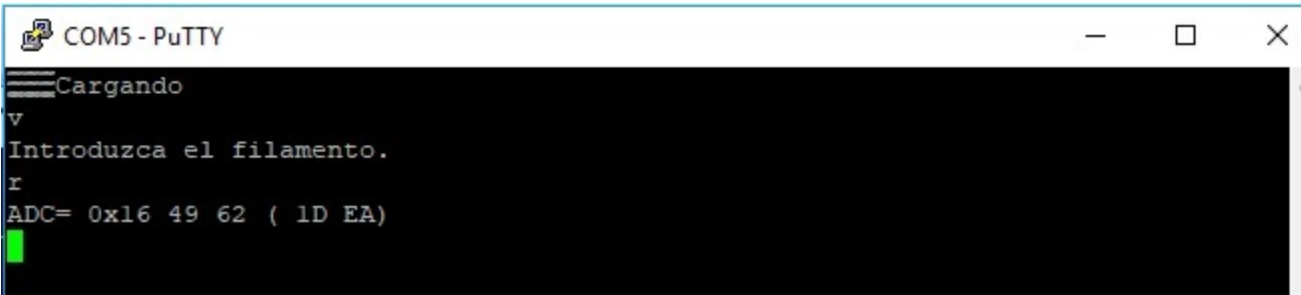


Figura 10: funcionamiento con “v” y luego “r”.

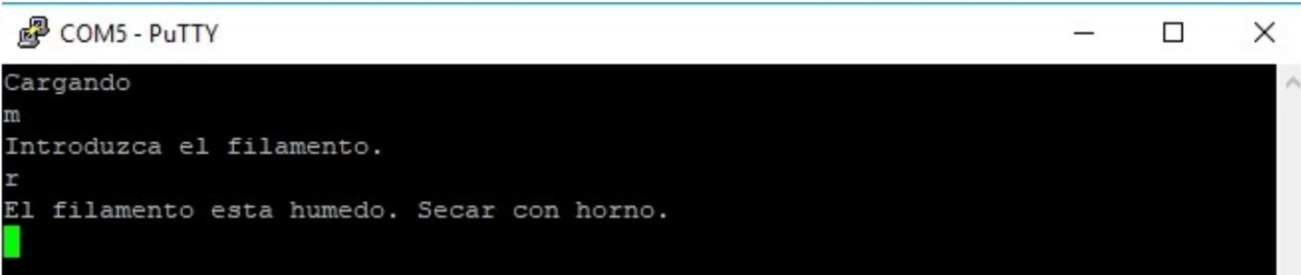


Figura 11 : funcionamiento con instrucciones “m” y luego “r”.

## 7. Conclusiones

Teniendo en cuenta lo expuesto en este informe y en el anteproyecto, se concluye que las especificaciones previas principales se cumplieron correctamente. Se logró determinar precisamente la diferencia de tensión proporcional a la permitividad relativa del filamento y, por lo tanto, diferenciar un filamento seco de uno húmedo. Además, se implementó la interfaz interactiva por medio del puerto serie, permitiéndole al usuario elegir la información que desea. Esto no se había planteado en el anteproyecto ya que la comunicación con el usuario iba a darse por medio de un *display*, que es un periférico únicamente de salida.

Sin embargo, no se pudo implementar la posibilidad de elegir el material con el que se trabaja ni la corrección de la capacidad por diferencias en el diámetro del filamento. Esto se propone como una mejora futura que podría realizar otro grupo.

## 8. Apéndice

### 8.1. Código Implementado

```
-----
;
;
; CÓDIGO PROYECTO DETECTOR DE HUMEDAD - FILAMENTO IMPRESIÓN 3D
; Última actualización: 2019-JUN-26 12:00
;
-----

;
; MCU: ATmega8/ATmega328P
; Según el entorno de desarrollo, el MCU (MicroController Unit) se elige
; a) desde Project->Properties se elige el microcontrolador
; b) con una directiva de include.
; En ambos casos se termina incluyendo un archivo m8def.inc/m328Pdef.inc
; que definen un símbolo _M8DEF_INC/_M328PDEF_INC_ que se usa en toda
; este código para distinguir entre registros de uno y otro micro.
;
-----
; Si el modo de seleccionar el MCU es el b) descomentar el include que
; corresponda:
; include m8def.inc          ; define los registros y constantes del Mega8
; include "m328Pdef.inc"    ; define los registros y constantes del Mega328P
;
-----

#define F_CPU 16000000      ; frecuencia del reloj del micro (F_SYS)

;
; Puerto donde se conecta un LED
;
-----
; En una placa Arduino uno el led está en PINB.5 y se enciende con '1'
; para otros circuitos verificar la ubicación de algún led y definir lo
; siguiente:
.equ    PORT_LED      = PORTB ; registro del puerto
.equ    DIR_LED       = DDRB  ; registro de direcciones del puerto
.equ    LED           = 5     ; nro. de bit (contando de 0 a 7=MSbit)
                                ; nota: En 1/0 prende/apaga LED

.equ    DIR_ADC       = DDRD
.equ    ADC_DT        = 3
.equ    ADC_SCK       = 4
.equ    PORT_ADC      = PORTD
.equ    PIN_ADC       = PIND

;
; MACROS
;
-----
; Inclusión de macros. El *.inc debe estar en la misma carpeta que los
; demás archivos fuente o bien incluir un path al mismo mediante:
; Project->Properties->Toolchain->AVR Assembler->General->Include Paths
; include "avr_macros.inc"
; listmac                      ; permite ver las macros en el listado *.lss
;
-----
; CONSTANTES
;
-----
.equ    BUF_SIZE      = 64     ; tamaño en bytes del buffer de transmisión
```

```

.equ    LF      = 13           ; '\n' caracter ascii de incremento de línea
.equ    CR      = 10           ; '\r' caracter ascii de retorno de carro
.equ    ciclos = 61
.equ    OV_PARA_1S = 63

.equ    IT_VACIO= 20           ; cantidad de mediciones que se hacen en vacío
.equ    UMBRAL  = 35
.equ    N_BUFFER = 8           ; cantidad de mediciones promediadas del adc
.equ    IT_MEMORIA = 8         ; esto ya no se usa porque fue reemplazado por el anterior
;-----
; variables en SRAM
;-----
                .dseg          ; Segmento de datos (RAM)

TX_BUF: .byte    BUF_SIZE      ; buffer de transmisión serie
RX_BUF: .byte    BUF_SIZE      ; buffer de recepción de datos por puerto serie
VERSION:.byte    1             ; nro. de versión del programa
ADC_BUFFER: .byte N_BUFFER*3
ADC_PROMEDIO: .byte 3
ADC_VACIO: .byte 2

;-----
; variables en registros
;-----

.def      restan_para_introducir = r5
.def      ptr_tx_L = r8         ; puntero al buffer de datos a transmitir
.def      ptr_tx_H = r9
.def      bytes_a_tx = r14      ; nro. de bytes a transmitir desde el buffer
.def      bytes_recibidos = r13

.def      t0      = r16         ; variable global auxiliar
.def      t1      = r17         ; variable global auxiliar
.def      veces_ov_0 = r19
.def      contador2 = r21

.def      nro_veces = r22

.def      eventos = r20
.equ      EVENTO_RX_SERIE = 0
.equ      EVENTO_ADC_FIN = 1
.equ      EVENTO_1SEG = 2
.equ      EVENTO_PROMEDIAR = 3
.equ      EVENTO_M = 4
.equ      EVENTO_FILAMENTO = 5
.equ      EVENTO_V = 6

;-----
; CODIGO
;-----
                .cseg
                rjmp    RESET          ; interrupción del reset

                .org    OV_F0addr
                rjmp    ISR_TOV0

```

```

.org    INT1addr      ; DT bajo, aviso a main que tiene que leer dato
rjmp    ISR_MUESTRA_ADC

.org    URXCaddr      ; USART, Rx Complete
rjmp    ISR_RX_USART_COMPLETA

.org    UDREaddr      ; USART Data Register Empty
rjmp    ISR_REG_USART_VACIO

RESET:
.org    INT_VECTORS_SIZE ; salteo todos los vectores de interrupción

ldi     r16,LOW(RAMEND)
out     spl,r16
ldi     r16,HIGH(RAMEND)
out     sph,r16      ; inicialización del puntero a la pila

ldi     r16,          IT_VACIO
mov     restan_para_introducir, r16

LDI     YH,           HIGH(ADC_BUFFER)
LDI     YL,           LOW(ADC_BUFFER)

LDI     nro_veces,    N_BUFFER

sbi     DIR_LED, LED   ; configuro como salida el puerto para manejar el LED
cbi     PORT_LED, LED  ; PRENDI el LED

CBI     DIR_ADC,      ADC_DT ; inicializo pin 3 de puerto D como entrada
SBI     DIR_ADC,      ADC_SCK ; inicializo pin 4 del puerto D como
salida D.4=ADC_SCK

CBI     PORT_ADC,     ADC_SCK ;adc en bajo consumo
SBI     PORT_ADC,     ADC_DT ;dt es una entrada con pull up interno

ldi     t0, 0x13
sts     VERSION, t0      ; versión actual de este módulo

rcall   USART_init      ; Configuro el puerto serie para tx/rx a 19.2 kbps
                        ; y habilito la interrupción de recepción de datos.

lds     R16, EICRA
                        ;configuro int 1 por flanco descendente
ori     R16, (1<<ISC11)
andi   R16, ~(1<<ISC10)
sts     EICRA, R16

;habilito interrupciones

in      R16, EIMSK
ori     R16, (1<<INT1)
out     EIMSK, R16

rcall   INICIALIZAR_TIMER0

```

```

;TEMPORIZADOR GENERAL (marca los instantes de transmision por puerto serie)

rcall    INICIALIZAR_TIMER2        ;generador de cuadrada para monoestable

sei                          ;habilitación global de todas las interrupciones

rcall    TEST_TX                    ; transmite un mensaje inicial


MAIN:                                          ; Programa principal (bucle infinito)

        tst        eventos          ; Pasó algo?
        breq       MAIN             ; nada

        sbrc       eventos, EVENTO_FILAMENTO
        rjmp       EVENTO5          ; activar con m-enter luego de r-enter

        sbrc       eventos, EVENTO_PROMEDIAR
        rjmp       EVENTO3

        sbrc       eventos, EVENTO_ADC_FIN
        rjmp       EVENTO1

        sbrc       eventos, EVENTO_1SEG
        rjmp       EVENTO2

        sbrc       eventos, EVENTO_RX_SERIE
        rjmp       PROCESO_TRAMA_RX

        rjmp       MAIN


EVENTO1:
        CBR        eventos, (1<<EVENTO_ADC_FIN)
        rcall      LEER_BITS
        rjmp       main


EVENTO2:
        CBR        eventos, (1<<EVENTO_1SEG)
        rcall      LEER_ADC
        rjmp       main


EVENTO3:
        CBR        eventos, (1<<EVENTO_PROMEDIAR)
        rcall      PROMEDIAR_MUESTRAS
        rjmp       main


EVENTO5:
; hace la resta de un valor (promediado) con el obtenido con filamento.
;Si vine acá es porque ya hay un valor CON filamento en ADC_PROMEDIO
        CBR        eventos, (1<<EVENTO_FILAMENTO)
        rcall      MSJ_Y_COMPARACION
        rjmp       main


PROCESO_TRAMA_RX:

```



```

// Los datos recibidos x puerto serie están a partir del dirección RAM RX_BUF
// y terminan siempre con el caracter LF. Además "bytes_recibidos" me dice
// cuántos bytes tiene la trama.
cbr      eventos,(1<<EVENTO_RX_SERIE)      ; Limpio flag del evento
clr      bytes_recibidos                    ; limpio nro. de bytes recibidos
                                              ;porque no lo uso

lds      t0, RX_BUF                        ; miro el 1er caracter de la trama recibida
cpi      t0, 'h'
brne     VER_SI_MEDIR_O_VALOR_O_READY

rcall    HELP_TX
rjmp     main

```

VER\_SI\_MEDIR\_O\_VALOR\_O\_READY:

```

cpi      t0, 'r'
brne     VER_SI_MEDIR_O_VALOR

sbrc     eventos, EVENTO_M
SBR      eventos, (1<<EVENTO_FILAMENTO)
sbrc     eventos, EVENTO_V
rcall    DATO_TX

andi     eventos, ~((1<<EVENTO_M) | (1<<EVENTO_V)) ; limpio esos bits
rjmp     main

```

VER\_SI\_MEDIR\_O\_VALOR:

```

rcall    MEDIR_O_VALOR
cpi      t0, 'm'
brne     VER_SI_VALOR
SBR      eventos, (1<<EVENTO_M)
; no hace nada, vuelve al main, se va a medir y espera el ready
rjmp     main

```

VER\_SI\_VALOR:

```

cpi      t0, 'v'
brne     main
SBR      eventos, (1<<EVENTO_V)
; no hace nada, vuelve al main, se va a medir y espera el ready
rjmp     main

```

LEER\_ADC:

```

;-----
;                                     COMUNICACION SERIE
;-----
#if      F_CPU == 8000000
.equ     BAUD_RATE      = 51      ; 19.2 kbps e=0.2%      @8MHz y U2X=1
#elif F_CPU == 16000000
.equ     BAUD_RATE      = 103     ; 19.2 kbps e=0.2% @16MHz y U2X=1
#else
.error "Falta calcular el baud rate para el F_CPU definido!"
#endif

```

```

;-----
USART_init:
    push    t0
    push    t1
    pushw   X

#ifdef _M328PDEF_INC_
    outi     UBRROH, high(BAUD_RATE)
    outi     UBRROL, low(BAUD_RATE)
    outi     UCSROA, (1<<U2X0)
    ; Trama: 8 bits de datos, sin paridad y 1 bit de stop,
    outi     UCSROC, (0<<UPM01)|(0<<UPM00)|(0<<USBS0)|(0<<UCSZ02)|(1<<UCSZ01)|(1<<UCSZ00)
    ; Configura los terminales de TX y RX; y habilita
    ; únicamente la int. de recepción
    outi     UCSROB, (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0)|(0<<UDRIE0)
#else
    outi     UBRRH, high(BAUD_RATE)    ; Velocidad de transmisión
    outi     UBRRL, low(BAUD_RATE)
    outi     UCSRA, (1<<U2X)            ; Modo asinc., doble velocidad
    outi     UCSRC, (1<<URSEL)|(0<<UPM1)|(0<<UPM0)|(0<<USBS)|(1<<UCSZ1)|(1<<UCSZ0)
    outi     UCSRB, (1<<RXCIE)|(1<<RXEN)|(1<<TXEN)|(0<<UDRIE)
#endif

    movi     ptr_tx_L, LOW(TX_BUF)      ; inicializa puntero al
    movi     ptr_tx_H, HIGH(TX_BUF)     ; buffer de transmisión.

    ldiw     X, TX_BUF                  ; limpia BUF_SIZE posiciones
    ldi       t1, BUF_SIZE              ; del buffer de transmisión
    clr      t0

loop_limpiar:
    st       X+, t0
    dec      t1
    brne     loop_limpiar

    clr      bytes_a_tx                  ; nada pendiente de transmisión
    clr      bytes_recibidos             ; nada recibido aún.

    popw     X
    pop      t1
    pop      t0
    ret

```

```

;-----
; RECEPCION: Interrumpe cada vez que se recibe un byte x RS232.
;
; Recibe: UDR (byte de dato)
; Devuelve: nada
;-----

```

```

ISR_RX_USART_COMPLETA:
; EL registro UDR tiene un dato y debería ser procesado
    push     t0
    pushi    SREG
    pushw    Y

    ldiw     Y, RX_BUF

```

```

        add        YL, bytes_recibidos
        clr        t0
        adc        YH, t0

#ifdef _M328PDEF_INC_
        input      t0,    UDRO
#else
        input      t0, UDR
#endif

        st         Y, t0
        inc        bytes_recibidos
        ldi        t0, BUF_SIZE
        cp         bytes_recibidos, t0
        brlo       BUF_RX_CON_ESPACIO

        clr        bytes_recibidos           ; error, se sobrepasó el espacio disponible
                                              ; para mensajes recibidos x puerto serie.
                                              ; Debería informar al main (pero no lo hago).

BUF_RX_CON_ESPACIO:
        ld         t0, Y

ACA:
        cpi        t0, LF
        brne       FIN_ISR_RX_USART

        ldi        t0, (1<<EVENTO_RX_SERIE)
        or         eventos, t0

FIN_ISR_RX_USART:
        popw       Y
        popi       SREG
        pop        t0
        reti

;-----
; TRANSMISION: interrumpe cada vez que puede transmitir un byte.
; Se transmiten "bytes_a_tx" comenzando desde la posición TX_BUF del
; buffer. Si "bytes_a_tx" llega a cero, se deshabilita la interrupción.
;
; Recibe: bytes_a_tx.
; Devuelve: ptr_tx_H:ptr_tx_L, y bytes_a_tx.
;-----
ISR_REG_USART_VACIO:           ; UDR está vacío

        push       t0
        push       t1
        pushi      SREG
        pushw      X

        tst        bytes_a_tx           ; hay datos pendientes de transmisión?
        breq       FIN_TRANSMISION

        movw       XL, ptr_tx_L         ; Recupera puntero al próximo byte a tx.
        ld         t0, X+               ; lee byte del buffer y apunta al

#ifdef _M328PDEF_INC_
        output      UDRO, t0

```

```

#else
        output    UDR, t0                ; sgte. dato a transmitir (en la próxima int.)
#endif

```

```

        cpi        XL,LOW(TX_BUF+BUF_SIZE)
        brlo       SALVA_PTR_TX
        cpi        XH,HIGH(TX_BUF+BUF_SIZE)
        brlo       SALVA_PTR_TX
        ldiw       X,TX_BUF                ; ptr_tx=ptr_tx+1, (módulo BUF_SIZE)

```

SALVA\_PTR\_TX:

```

        movw       ptr_tx_L,XL            ; preserva puntero a sgte. dato

        dec        bytes_a_tx              ; Descuenta el nro. de bytes a tx. en 1
        brne       SIGUE_TX                ; si quedan datos que transmitir
                                           ;vuelve en la próxima int.

```

FIN\_TRANSMISION: ; si no hay nada que enviar,

#ifdef \_M328PDEF\_INC\_

```

        lds        t0,UCSR0B
        andi       t0, ~(1<<UDRIE0)
        sts        UCSR0B, t0
        ;cbix      UCSR0B, UDRIE0

```

#else

```

        cbix       UCSRB, UDRIE            ; se deshabilita la interrupción.

```

#endif

sigue\_tx:

```

        popw       X
        popi       SREG
        pop        t1
        pop        t0
        reti

```

```

;-----
; NOMBRE_TX: transmite el mensaje almacenado en memoria flash a partir
; de la dirección NOMBRE_TX que termina con 0x00 (el 0 no se transmite).
; Recibe: nada
; Devuelve: nada
;-----

```

INTRODUCIR\_FIL\_TX:

```

        pushw      Z
        push       t0
        PUSH       R0
        PUSH       R1

        ldiw       Z,(MSJ_INTR*2)
        rcall      TX_MSJ

        POP        R1
        POP        R0
        pop        t0
        popw       Z
        ret

```

TEST\_TX:

```

pushw Z
push t0
PUSH R0
PUSH R1

ldiw Z,(MSJ_TEST_TX*2)
rcall TX_MSJ

POP R1
POP R0
pop t0
popw Z
ret

```

HUMEDO\_TX:

```

pushw Z
push t0
PUSH R0
PUSH R1

ldiw Z,(MSJ_HUMEDO*2)
rcall TX_MSJ

POP R1
POP R0
pop t0
popw Z
ret

```

SECO\_TX:

```

pushw Z
push t0
PUSH R0
PUSH R1

ldiw Z,(MSJ_SECO*2)
rcall TX_MSJ

POP R1
POP R0
pop t0
popw Z
ret

```

DATO\_TX:

```

pushw Z
push t0
PUSH R0
PUSH R1

ldiw Z,(MSJ_DATO*2)
rcall TX_MSJ

POP R1

```

```

POP      R0
pop      t0
popw     Z
ret

```

HELP\_TX:

```

pushw    Z
push     t0
PUSH     R0
PUSH     R1

ldiw     Z,(HELP*2)
rcall    TX_MSJ

POP      R1
POP      R0
pop      t0
popw     Z
ret

```

HELP:

```

.db "h: ayuda; v: valor de medicion; m: humedad"
.db      '\r','\n',0,0

```

MSJ\_DAT0:

```

.db "ADC= 0x%", low(ADC_PROMEDIO), high(ADC_PROMEDIO), " %", low(ADC_PROMEDIO+1),
high(ADC_PROMEDIO+1)
.db " %", low(ADC_PROMEDIO+2), high(ADC_PROMEDIO+2)
.db " (, " %, low(ADC_VACIO), high(ADC_VACIO)
.db " %, low(ADC_VACIO+1), high(ADC_VACIO+1), " ) "
.db      '\r','\n',0,0

```

MSJ\_TEST\_TX:

```

.db      "Cargando"
.db      '\r','\n',0,0

```

MSJ\_INTR:

```

.db      "Introduzca el filamento."
.db      '\r','\n',0,0

```

MSJ\_SECO:

```

.db      "El filamento esta seco. Puede ser usado."
.db      '\r','\n', 0,0

```

MSJ\_HUMEDO:

```

.db      "El filamento esta humedo. Secar con horno."
.db      '\r','\n', 0,0

```

```

;-----
; TX_MSJ: transmite el mensaje almacenado en memoria flash a partir
; de la dirección que se pase en el puntero Z. El mensaje debe termina
; con 0x00 (el 0 no se transmite).
;
; Recibe: Z (=r31|r30)
; Devuelve: bytes_a_tx > 0
; Habilita la int. de transmisión serie con ISR en ISR_REG_USART_VACIO().

```

;-----

TX\_MSJ:

```
push    t0
pushi   SREG
pushw   X
```

```
movw    XL, ptr_tx_L    ; toma el último valor del puntero
```

COPIA\_A\_TX\_BUF:

```
lpm          t0,Z+    ; y copia de flash a ram
tst          t0        ; si encuentra un 0x00 en el mensaje, termina
breq        ACTIVA_TX_MSJ ;de cargar el buffer en RAM e inicia la transmisión.
```

; Si el carácter es '%' seguido de un nro. hexadecimal de 16 bits, toma el byte  
; de esa dirección de RAM, lo convierte a ASCII y lo pone en el buffer de transmisión.

```
cpi          t0, '%'
brne        NO_HAY_VARIABLES
```

```
pushw   Y
lpm          YL, Z+
lpm          YH, Z+
ld         t0, Y
rcall      BYTE_2_ASCII    ; devuelve en r1|r0 el ascii del byte
popw       Y
```

```
st          X+, r1
inc          bytes_a_tx
```

```
mov          t0, r0
cpi          XL, low(TX_BUF+BUF_SIZE)
brlo        NO_HAY_VARIABLES
cpi          XH, high(TX_BUF+BUF_SIZE)
brlo        NO_HAY_VARIABLES
ldiw         X, TX_BUF      ; ptr_tx++ módulo BUF_SIZE
```

NO\_HAY\_VARIABLES:

```
st          X+,t0
inc          bytes_a_tx
```

```
cpi          XL, low(TX_BUF+BUF_SIZE)
brlo        COPIA_A_TX_BUF
cpi          XH, high(TX_BUF+BUF_SIZE)
brlo        COPIA_A_TX_BUF
ldiw         X, TX_BUF      ; ptr_tx++ módulo BUF_SIZE
```

```
rjmp        COPIA_A_TX_BUF
```

ACTIVA\_TX\_MSJ: ; habilita la int. de tx

#ifdef \_M328PDEF\_INC\_

```
sbix        UCSROB, UDRIE0
```

#else

```
sbix        UCSRB, UDRIE
```

#endif

```

        popw    X
        popi    SREG
        pop     t0
        ret

;-----
; Recibe t0 y devuelve el ascii en r1|r0
; Por ejemplo, si t0=0xA3 devuelve r1='A'=0x41 r0='3'=0x33
;-----
BYTE_2_ASCII:
        mov     r0,    t0
        andi    t0, 0xF0
        swap    t0
        cpi     t0, 0x0A
        brlo    sumo_30h
        subi    t0,    -0x07

sumo_30h:
        subi    t0,    -0x30
        mov     r1, t0

        mov     t0, r0
        andi    t0, 0x0F
        cpi     t0, 0x0A
        brlo    sumo_30h_bajo
        subi    t0,    -0x07                ; sino 0x30+0x07

sumo_30h_bajo:
        subi    t0, -0x30                ; if r1<=9, sumo 30
        mov     r0, t0
        ret

ISR_MUESTRA_ADC:
        PUSH    R16
        PUSH    R17
        IN      R16,    SREG
        PUSH    R16

        IN      R16,    PIN_ADC
        LDI     R17,    5

DT_BAJ0:
        SBRC    R16,    ADC_DT
        RJMP    FIN_ISR

        DEC     R17
        BRNE    DT_BAJ0
        SBR     eventos, (1<<EVENTO_ADC_FIN)

        in      R16,    EIMSK
        andi    R16,    ~(1<<INT1)
        out     EIMSK, R16

FIN_ISR:
        POP     R16
        OUT     SREG, R16
        POP     R17
        POP     R16

```



```

                                RETI
;-----
; LEER_BITS: extrae datos del conversor AD y los guarda en el buffer (RAM)
; Luego de ocupar 8*3 bytes cambia el bit de eventos para que se promedie
;-----
LEER_BITS:

                                PUSH    R19
                                PUSH    R18
                                PUSH    R17
                                push     r16

                                IN        R16,          SREG
                                PUSH    R16

                                PUSH    R12
                                PUSH    R11
                                PUSH    R10
                                PUSHW   Y

                                push     contador2

                                ;leo bits del puerto D bit 3 porque es donde esta conectado DT del AD

                                SBI        DDRC, 0
                                LDI        contador2,    24
para sacar 24 bits. DT NO VUELVE A 1                                ;hay que mandar 24 pulsos

LOOP:
                                SBI        PORT_ADC,    ADC_SCK                                ;mando 1 al sck

                                LDI        R16,          16                                ;espero 16 ciclos
ESPERO_1MICRO:
                                DEC        R16
                                BRNE     ESPERO_1MICRO

                                CBI        PORT_ADC,    ADC_SCK                                ;mando 0 al sck

                                CLC
                                SBIC     PIN_ADC,      ADC_DT                                ;leo info del DT
                                SEC

                                BRCC     PORTC_0
                                CBI        PORTC, 0
                                RJMP     ROLEO

PORTC_0:
                                SBI        PORTC, 0

ROLEO:
                                ROL        R10
                                ROL        R11
                                ROL        R12

                                LDI        R16,          8                                ;espero 8 ciclos

```

```

ESPERO_EN_BAJO_SCK:
    DEC        R16
    BRNE       ESPERO_EN_BAJO_SCK

    DEC        contador2
    BRNE       LOOP

    LDI        YH,        HIGH(ADC_BUFFER)
    LDI        YL,        LOW(ADC_BUFFER)
    MOV        R16,      nro_veces
    DEC        R16
    LDI        R17,      3
    MUL        R16,      R17

    ADD        YL,        R0
    adc        yh,        R1

    ST         Y+,        R12
    ST         Y+,        R11
    ST         Y+,        R10

    DEC        nro_veces
    BRNE       NO_AGOTO_BUFFER
    LDI        nro_veces,      N_BUFFER

NO_AGOTO_BUFFER:

    LDI        contador2,      2

SIGUIENTE_CONVERSION:
    SBI        PORT_ADC,      ADC_SCK           ;mando 1 al sck

    LDI        R16,          16                 ;espero 16 ciclos
ESPERO_CLK_ALTO_SC:
    DEC        R16
    BRNE       ESPERO_CLK_ALTO_SC

    CBI        PORT_ADC,      ADC_SCK           ;mando 0 al sck

    LDI        R16,          16                 ;espero 16 ciclos
ESPERO_EN_BAJO_SC:
    DEC        R16
    BRNE       ESPERO_EN_BAJO_SC

    DEC        contador2
    BRNE       SIGUIENTE_CONVERSION

    in         R16,      EIFR
    ori        R16,      (1<<INTF1)
    out        EIFR,      R16

    in         R16,      EIMSK
    ori        R16,      (1<<INT1)
    out        EIMSK,      R16

```

```

SBR                                eventos, (1<<EVENTO_PROMEDIAR)    ;se promedia cada vez que
                                   ;hay una nueva muestra

POP                                contador2
POPW    Y
POP                                R10
POP                                R11
POP                                R12
POP    R16

OUT                                SREG,          R16
POP                                R16
POP                                R17
POP                                R18
POP                                R19

RET

;-----
; PROMEDIAR_MUESTRAS: promedia muestras del buffer y las guarda en la
; dirección de RAM ADC_PROMEDIO
;-----

PROMEDIAR_MUESTRAS:
    ; R3 | R2 | R1 | R0

    PUSHW    Y
    PUSH     R0
    PUSH     R1
    PUSH     R2
    PUSH     R3
    PUSH     R16
    IN        R16,          SREG
    PUSH     R16
    PUSH     R17
    PUSH     R18
    PUSH     R19

    LDI        YH,          HIGH(ADC_BUFFER)
                                   ;pongo puntero en ADC_BUFFER que es primer
byte de la suma
    LDI        YL,          LOW(ADC_BUFFER)
    LDI        R16,          N_BUFFER
    CLR        R0
    CLR        R1
    CLR        R2
    CLR        R3

SUMAS_PARCIALES:
    LD         R19,          Y+
    LD         R18,          Y+
    LD         R17,          Y+
    ADD        R0,          R17    ;sumo la parte baja

```



```

CLR          R24          ;inicializo en 0
OUT          TCNT0,       R24

IN           R24,         TCCR0B
ORI          R24,         (1<<CS02) | (1<<CS00)
ANDI        R24,         ~(1<<CS01))
OUT          TCCR0B,      R24

input        r24,         TIMSK0
ORI          R24,         (1<<TOIE0)
output       TIMSK0,      R24

pop          r24

RET

```

ISR\_TOV0: ;para monitorear flag TOV0

```

PUSH        R16
IN           R16,         SREG
PUSH        R16
PUSH        R17

DEC          veces_ov_0
BRNE        FIN_ISR_TOV0

;pasó un segundo
LDI          veces_ov_0,   OV_PARA_1S
SBR          eventos,     (1<<EVENTO_1SEG)

```

FIN\_ISR\_TOV0:

```

POP          R17
POP          R16
OUT          SREG,        R16
POP          R16

RETI

```

;A continuación, timer para generar cuadrada

INICIALIZAR\_TIMER2:

```

SBI          DDRB,        3
LDI          R16,         79
STS          OCR2A,        R16

```

```

CLR          R25          ;inicializo en 0
STS          TCNT2,        R25
;configuro timer sin prescaler y modo normal
LDS          R25,         TCCR2A
ANDI        R25,         ~(1<<WGM20)
ORI          R25,         (1<<WGM21)
STS          TCCR2A,        R25

```

```

LDS      R25, TCCR2B
ORI      R25, (1<<CS20)
ANDI     R25, ~(1<<CS21)
ANDI     R25, ~(1<<CS22)
STS      TCCR2B, R25

LDS      R25, TCCR2A
ANDI     R25, ~(1<<COM2A1)
ORI      R25, (1<<COM2A0)
STS      TCCR2A, R25

RET

```

```

;-----
; MEDIR_O_VALOR: en primer lugar, guarda los dos bytes menos significativos
; de ADC_PROMEDIO (RAM) y los guarda en ADC_VACIO y ADC_VACIO+1. Servirán
; luego para comparar. Además, se manda mensaje para introducir filamento
;-----

```

```

MEDIR_O_VALOR:
    push    R16
    lds     r16, ADC_PROMEDIO+1
    sts     ADC_VACIO, r16
    lds     r16, ADC_PROMEDIO+2
    sts     ADC_VACIO+1, r16
    rcall   INTRODUCIR_FIL_TX
    pop     R16
    ret

```

```

;-----
; MSJ_Y_COMPARACION: con los valores guardados en vacío y los actuales
; (con filamento), se hace una comparación y se envía el mensaje reportando
; si el mismo está húmedo o seco
;-----

```

```

MSJ_Y_COMPARACION:
    push    r16
    in      r16, sreg
    push    r16
    push    r17
    push    r18
    push    r19
    push    r23
    lds     r16, ADC_PROMEDIO+1
    lds     r17, ADC_PROMEDIO+2 ; valores con filamento
    lds     r18, ADC_VACIO
    lds     r19, ADC_VACIO+1 ; valores en vacío
    sub     r17, r19
    sbc     r16, r18
    brcs    esta_seco
    cpi     r16, UMBRAL
    brlo    esta_seco
    rcall   HUMEDO_TX
    rjmp    esta_humedo

```

```

esta_seco:
    rcall    SECO_TX
esta_humedo:
    pop      r23
    pop      r19
    pop      r18
    pop      r17
    pop      r16
    out      sreg,    r16
    pop      r16
    ret

```

```

;-----
; fin del código
;-----;

```

## 8.2. Hojas de Datos

Se adjunta la primera hoja de las hojas de datos de los principales componentes.

# HD74LS221

## Dual Monostable Multivibrators

REJ03D0458-0300  
Rev.3.00  
Jul.15.2005

This multivibrator features a negative-transition-triggered input and a positive-transition-triggered input either of which can be used as an inhibit input. Pulse triggering occurs at a particular voltage level and is not directly related to the transition time of the input pulse. Schmitt-trigger input circuitry (TTL hysteresis) for B input allows jitter-free triggering from inputs with transition rates as slow as 1 V/s, providing the circuit with excellent noise immunity of typically 1.2 V. A high immunity to  $V_{CC}$  noise of typically 1.5 V is also provided by internal latching circuitry. Once fired, the outputs are independent of further transitions of the A and B inputs and are a function of the timing components, or the output pulses can be terminated by the overriding clear. Input pulses may be of any duration relative to the output pulse. Output rise and fall times are TTL compatible and independent of pulse length.

Typical triggering and clearing sequence are illustrated as a part of the switching characteristics waveforms. Pulse width stability is achieved through internal compensation and is virtually independent of  $V_{CC}$  and temperature.

In most applications, pulse stability will only be limited by the accuracy of external timing components. Jitter-free operation is maintained over the full temperature and  $V_{CC}$  range for more than six decades of timing capacitance (10 pF to 10  $\mu$ F) and more than one decade of timing resistance (2 k $\Omega$  to 100 k $\Omega$ ).

Throughout these ranges, pulse width is defined by the relationship:  $t_{w(out)} = C_{ext} \bullet R_{ext} \bullet \ln 2$ .

### Features

- Ordering Information

Part Name	Package Type	Package Code (Previous Code)	Package Abbreviation	Taping Abbreviation (Quantity)
HD74LS221P	DILP-16 pin	PRDP0016AE-B (DP-16FV)	P	—
HD74LS221RPEL	SOP-16 pin (JEDEC)	PRSP0016DG-A (FP-16DNV)	RP	EL (2,500 pcs/reel)

Note: Please consult the sales office for the above package availability.

*Hoja de Datos del Monoestable.*



## 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales

### DESCRIPTION

Based on Avia Semiconductor's patented technology, HX711 is a precision 24-bit analog-to-digital converter (ADC) designed for weigh scales and industrial control applications to interface directly with a bridge sensor.

The input multiplexer selects either Channel A or B differential input to the low-noise programmable gain amplifier (PGA). Channel A can be programmed with a gain of 128 or 64, corresponding to a full-scale differential input voltage of  $\pm 20\text{mV}$  or  $\pm 40\text{mV}$  respectively, when a 5V supply is connected to AVDD analog power supply pin. Channel B has a fixed gain of 32. On-chip power supply regulator eliminates the need for an external supply regulator to provide analog power for the ADC and the sensor. Clock input is flexible. It can be from an external clock source, a crystal, or the on-chip oscillator that does not require any external component. On-chip power-on-reset circuitry simplifies digital interface initialization.

There is no programming needed for the internal registers. All controls to the HX711 are through the pins.

### FEATURES

- Two selectable differential input channels
- On-chip active low noise PGA with selectable gain of 32, 64 and 128
- On-chip power supply regulator for load-cell and ADC analog power supply
- On-chip oscillator requiring no external component with optional external crystal
- On-chip power-on-reset
- Simple digital control and serial interface: pin-driven controls, no programming needed
- Selectable 10SPS or 80SPS output data rate
- Simultaneous 50 and 60Hz supply rejection
- Current consumption including on-chip analog power supply regulator:
  - normal operation  $< 1.5\text{mA}$ , power down  $< 1\mu\text{A}$
- Operation supply voltage range:  $2.6 \sim 5.5\text{V}$
- Operation temperature range:  $-40 \sim +85^\circ\text{C}$
- 16 pin SOP-16 package

### APPLICATIONS

- Weigh Scales
- Industrial Process Control

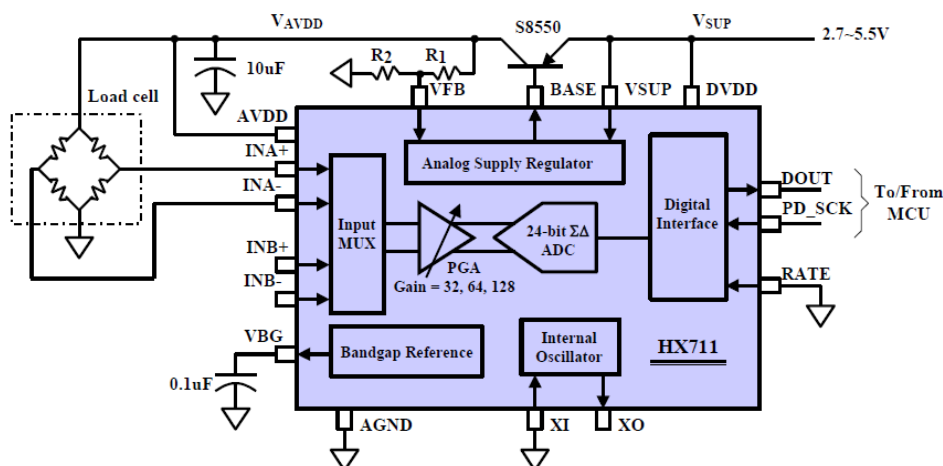


Fig. 1 Typical weigh scale application block diagram



# ATmega48A/PA/88A/PA/168A/PA/328/P

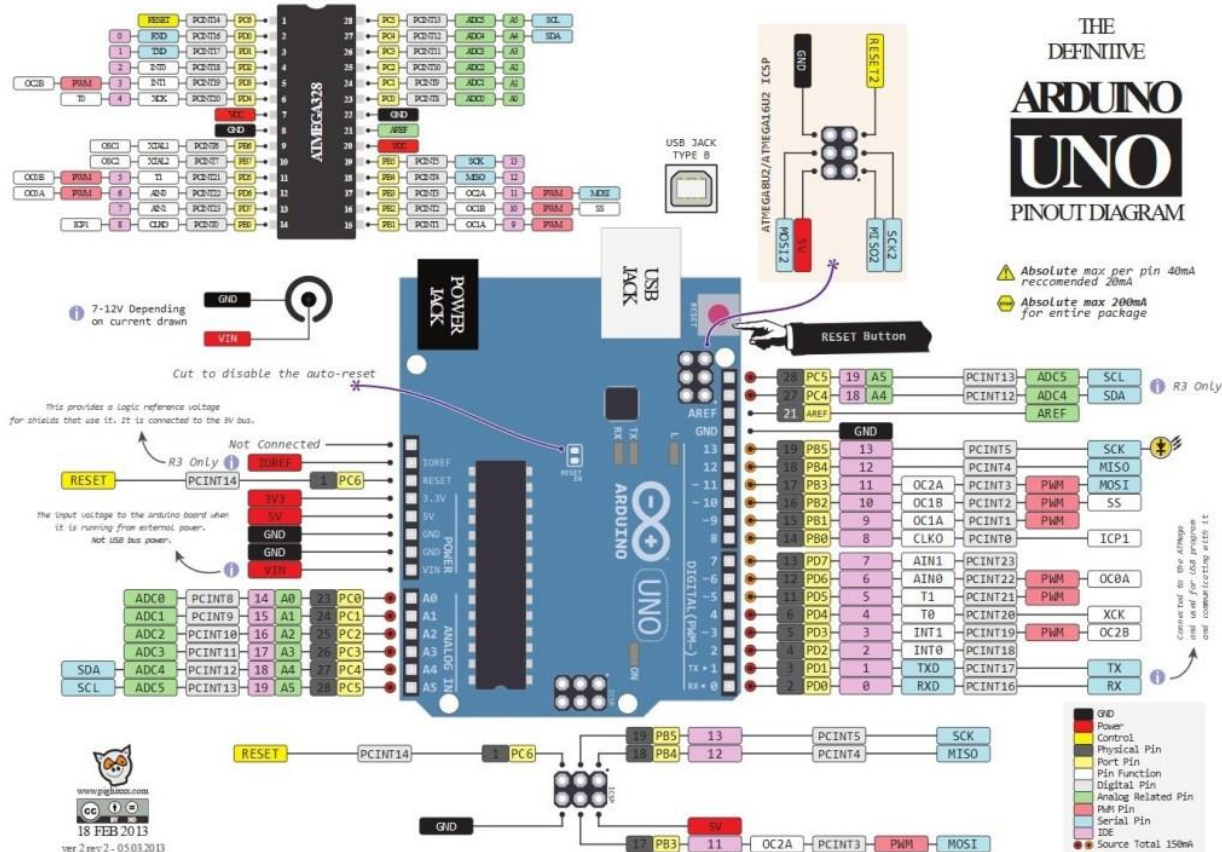
## megaAVR® Data Sheet

### Introduction

The ATmega48A/PA/88A/PA/168A/PA/328/P is a low power, CMOS 8-bit microcontrollers based on the AVR® enhanced RISC architecture. By executing instructions in a single clock cycle, the devices achieve CPU throughput approaching one million instructions per second (MIPS) per megahertz, allowing the system designer to optimize power consumption versus processing speed.

### Features

- High Performance, Low Power AVR® 8-Bit Microcontroller Family
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C<sup>(1)</sup>
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix™ acquisition
  - Up to 64 sense channels
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode



Disposición de Pines en Arduino Uno.

A continuación se muestra una sección de la hoja de datos del ADC, en la que se explica el funcionamiento de la salida de datos del mismo.

### Serial Interface

Pin PD\_SCK and DOUT are used for data retrieval, input selection, gain selection and power down controls.

When output data is not ready for retrieval, digital output pin DOUT is high. Serial clock input PD\_SCK should be low. When DOUT goes to low, it indicates data is ready for retrieval. By applying 25~27 positive clock pulses at the PD\_SCK pin, data is shifted out from the DOUT output pin. Each PD\_SCK pulse shifts out one bit, starting with the MSB bit first, until all 24 bits are shifted out. The 25<sup>th</sup> pulse at PD\_SCK input will pull DOUT pin back to high (Fig.2).

Input and gain selection is controlled by the number of the input PD\_SCK pulses (Table 3). PD\_SCK clock pulses should not be less than 25 or more than 27 within one conversion period, to avoid causing serial communication error.

PD_SCK Pulses	Input channel	Gain
25	A	128
26	B	32
27	A	64

Table 3 Input Channel and Gain Selection

Funcionamiento de DT y SCK del ADC.