Arduino Tcl/Tk
Osciloscopio

José María Sirvent Ichaso

Índice general

Capítulo 1

Antes de empezar

Este libro forma parte de una serie que trata sobre Arduino y sobre cómo expandir su funcionalidad utilizando Tcl/Tk en un ordenador o teléfono móvil. La idea es aprovechar al máximo las posibilidades gráficas y de conexión de estos últimos, para hacer brillar esos proyectos que tenemos en mente en los que Arduino está involucrado.

Este libro asume que tienes un mínimo de experiencia integrando Arduino con Tcl/Tk. Si no es así, te recomiendo los libros anteriores de la misma serie 'Arduino Tcl/Tk':

- Primeros pasos
- Radio FM

1.1. ¿Qué vas a encontrar en este libro?

Vamos a desarrollar un 'osciloscopio' mediante Arduino y Tcl/Tk. Evidentemente, éste será un 'poor man's oscilloscope' porque las frecuencias de muestreo estarán sobre los 38 kHz, pero veremos que desarrollaremos una heramienta realmente útil.

Podremos visualizar señales DC, AC, $V_{pp}, V_{rms}, V_{max}, V_{min}$, calcular su frecuencia, visualizar su espectrograma, etc.

Como nunca se tiene suficiente, también vamos a desarrollar un osciloscopio que será capaz de 2Msps [1] mediante el ADC CA3306, lo que nos permitirá analizar un rango más aceptable de señales.

1.2. Estructura de la serie

Este libro es la continuación natural a mi libro Arduino Tcl/Tk: Radio FM en el que mediante la documentación del TEA5767 realizamos una radio FM funcional con diferentes interfaces. Aquí por tanto ya doy por supuesto una cierta soltura utilizando el puerto serie, Tcl/Tk, data sheets y demás.

[1] Megasamples per second

Los proyectos que tengo en mente para la serie son estos:

- radio FM
- osciloscopio
- lanzamisiles
- chat vía radio
- alarma
- ...

La verdad que hay tantas cosas divertidas para crear, que la lista puede ser interminable. Esta es una de las razones por las que no he creado un libro sobre Arduino-Tcl/Tk, sino una serie. Además, evitamos un libro de grandes dimensiones y vamos cerrando ítems de la lista.

1.3. Código fuente

TODO Todo el código fuente de los programas del libro lo podrás encontrar en Arduino Tcl/Tk Osciloscopio.

Capítulo 2

Maqueta

Todo este libro parte de una de las funcionalidades integradas en el microcontrolador de Arduino: su conversor analógico-digital (ADC). Será precisamente este periférico el encargado de transformar la señal analógica que queremos observar en una secuencia de valores que posteriormente podremos representar y procesar en el ordenador.

Las características del ADC determinan, en buena medida, las prestaciones que podremos obtener de nuestro osciloscopio. Arduino nos proporciona un excelente punto de partida para experimentar con adquisición de señales sin necesidad de añadir hardware especializado. Si posteriormente queremos aumentar considerablemente la velocidad

de muestreo, podemos recurrir a conversores externos específicamente diseñados para esta tarea, como el CA3306, utilizado en numerosos proyectos con Arduino para alcanzar velocidades de varios millones de muestras por segundo.

Sin embargo, las prestaciones de un osciloscopio no dependen únicamente de su velocidad de muestreo. Para que resulte realmente útil también son importantes las herramientas que nos permiten visualizar, analizar y medir las señales adquiridas. A lo largo del libro iremos incorporando algunas de ellas.

Cuando comencé a desarrollar este prototipo, mi objetivo inicial era bastante sencillo: visualizar la forma de onda de una señal y realizar medidas manuales mediante cursores. Una vez montado el primer circuito en la breadboard, comprobé que el software nos permitía ir mucho más allá sin complicar excesivamente el hardware. De esta forma fueron apareciendo funciones como el zoom, el cálculo automático de la tensión pico a pico y de la tensión eficaz, o la representación del espectro de la señal.

Además, mediante algunas ampliaciones relativamente sencillas en la etapa de entrada, podremos trabajar con señales de hasta 30 Vpp, extendiendo considerablemente el rango de tensiones que podemos estudiar respecto a los 0–5 V que admite directamente una entrada de Arduino.

2.1. Materiales

La lista de materiales puede ser mínima si queremos:

- Arduino
- 2 cables

Aunque en este caso solo podremos medir y visualizar señades DC. De todas formas, veremos que lo que hará útil a nuestro osciloscopio es la versatilidad y todas las funcionalidades que vayamos añadiendo.

Aun siendo modestos, podemos aumentar las posibilidades de este osciloscopio añadiendo unos cuantos componentes como amplificadores operacionales, relés y algún que otro componente discreto.

Un aspecto importante de este proyecto es que el apartado hardware puede crecer tanto como queramos y, con él, también las prestaciones físicas de nuestro osciloscopio. Podríamos incorporar desde el principio acoplamiento AC, distintos rangos de entrada, amplificación, varios canales y otras mejoras. Sin embargo, para evitar caer en la típica madriguera de conejo y mantener el desarrollo manejable, comenzaremos con una versión deliberadamente espartana: un osciloscopio de un solo canal capaz de medir señales continuas entre 0 y 5 V.

La elección de un único canal responde a esa misma filosofía de simplicidad, pero también tiene una ventaja

práctica: nos permite dedicar los recursos disponibles a conseguir la mayor velocidad de muestreo posible. Una vez tengamos esta primera versión funcionando correctamente, podremos plantearnos ampliar progresivamente sus capacidades, incorporando medida de señales AC, rangos de tensión mayores, amplificación de la señal o incluso canales adicionales.

2.2. Idea inicial

Todo surgió hace ya muchos años leyendo una entrada del magnífico blog Jeelabs [1], donde encontré la entrada A mini scope. En ella, con solo unas decenas de líneas, podíamos ver señales en la pantalla de nuestro ordenador.

La idea quedó rondándome durante algún tiempo, hasta que más adelante leí *Programming Arduino Next Steps*, de Simon Monk. En uno de sus ejemplos se llevaba el ADC de Arduino mucho más allá de su uso habitual, aprovechando al máximo sus posibilidades. Aquello terminó de convencerme: si con un hardware tan sencillo era posible obtener resultados interesantes, había llegado el momento de dejar de darle vueltas y empezar a hacer mis propias pruebas.

[1] Anteriormente usaba otro dominio

2.3. ¿Qué nos ofrece Arduino?

Lo primero que vamos a hacer es ver qué nos ofrece Arduino desde un principio y comprender de dónde vienen los datos que vamos a obtener, porque comprendiéndolo, seremos capaces de modificar y ampliar las capacidades de nuestro osciloscopio.

Ya sabemos que podemos hacer lecturas analógicas en 6 de sus pines: A0-A5. Veamos con un pequeño programa qué podemos obtener.

```
/**
  Vamos a ver cuantas lecturas por segundo(aprox)
  podemos hacer con un Arduino tal cual.
*/
void setup()
{
  Serial.begin(115200);
  delay(100);
  Serial.println("Inicio test");
  long start = millis();
  long i;

  for(i = 0; i < 1000000; i++)
    analogRead(A0);

  long length = (millis() - start) / 1000;
  Serial.println("Fin test");
  Serial.print("Segundos: ");
  Serial.println(length);
```

```
20 }
21
22 void loop()
23 {
24 }
```

Lo que arroja en mi caso, un resultado de 8928 lecturas/s o aproximadamente 9ksps. [2]

Curioso número ¿verdad? ¿de dónde habrá salido? La respuesta viene en la hoja de especificaciones del ATmega328P y de cómo está implementado 'analogRead'. Te recomiendo visites la página de Arduino UNO para cualquier consulta.

En la sección *Analog-to-Digital Converter* de las especificaciones tenemos las respuestas, y además vemos en el resumen de las características que en lugar de nuestros aproximadamente 9ksps, podemos llegar a 76.9ksps. Recomiendo leer la sección entera pero si vas al grano con la *Prescaling and Conversion Timing* sacamos las ideas básicas:

- Necesitamos una frecuencia de reloj de 50kHz-200kHz si queremos los 10bits de resolución... pero podemos aumentar la frecuencia de reloj si no necesitamos tanta resolución

[2] kilo samples per second

- Controlamos el preescalado con los registros ADPS0-2

- Una conversión normal tarda unos 13 ciclos de reloj

Si buscamos en el código de Arduino por los registros ADPSx, vemos que se configura en el fichero wiring.c y el preescalado es de 128 (si quieres ver cómo se hace la lectura del valor, es decir qué hace la función 'analogRead' examina wiring_analog.c). Por tanto:

- 16MHz / 128 = 125kHz / 13 ciclos/conversión = 9615 lecturas/s

Vaya, este valor teórico se parece bastante a nuestro valor real. ¡Ya sabemos de dónde sale nuestro valor empírico y saber el porqué es lo verdaderamente interesante ya que nos va a permitir tener la capacidad de manipularlo!

En principio 9ksps nos puede parecer suficiente (y seguramente lo será en muchos casos) pero tras haber leído el documento del ATmega nos viene a la cabeza como un flash utilizar 8bits y aumentar la frecuencia de lecturas/s y así tener algo parecido a ¡un osciloscopio!

Con 8bits de resolución podemos aumentar la frecuencia de reloj a 1MHz, lo que nos dará una frecuencia de muestreo aproximada de 77kHz.

Bien, pues leeremos los 8bits más significativos, así que el bit ADLAR del registro ADMUX deberá estar a 1 y bastará con leer el registro ADCH, como indica en la sección *ADCL and ADCH – The ADC Data Register*, además, usaremos el ADC en Free Running para obtener conversiones tan pronto se pueda.

Por tanto, hemos de realizar un programa que nos permita configurar el preescalado del ADC sin tener que estar reprogramando Arduino. Una posible solución sería la siguiente:

```
/**
  Arduino - Tcl/Tk - Osciloscopio

  Muestras/s aproximadas:
  -Prescaler 128: 16MHz / 128 = 125000 / (13
    ciclos/lectura) = 10ksps
  -Prescaler 64: 20ksps
  -Prescaler 32: 40ksps
  -Prescaler 16: 77ksps

  No iremos más allá...
*/
// Diferentes preescalados
const byte PS_128 = B00000111;
const byte PS_64 = B00000110;
const byte PS_32 = B00000101;
const byte PS_16 = B00000100;

// Configuracion base del ADCSRA:
```

```
19 // -habilitar el ADC sin empezar la conversión
20 // -usar free running
21 // -habilitar interrupciones
22 // -sin preescalado (PS_2)
23 const byte ADCSRA_BASE = B10101000;
24
25 volatile byte ADCSRA_OLD = 0x00;
26 // Buffer
27 #define BUFFER_SIZE 512
28
29 volatile byte buffer[BUFFER_SIZE];
30 volatile int bufferIndex;
31 volatile byte bufferFull;
32
33 // Comandos GUI --> Arduino
34 char command = ' ';
35 /**
36   Cuando llenamos el buffer paramos las lecturas
      para enviar los datos
37 */
38 ISR(ADC_vect)
39 {
40   buffer[bufferIndex] = ADCH;
41   bufferIndex = (bufferIndex + 1) % BUFFER_SIZE;
42
43   if(bufferIndex == 0)
44   {
45     ADCSRA_OLD = ADCSRA;
46     ADCSRA &= ~_BV(ADEN);
47     bufferFull = true;
48   }
```

```
49 }
50
51 /**
52   Inicializar el buffer de lecturas
53 */
54 void init_buffer()
55 {
56   bufferIndex = 0;
57   bufferFull = false;
58   memset((byte*)buffer, 0, BUFFER_SIZE);
59 }
60
61 /**
62   Configurar el ADC para usarlo como "
      osciloscopio"
63   Lectura en A0
64 */
65 void init_adc()
66 {
67   // Eliminar posibles ruidos deshabilitando el
      digital input buffer
68   DIDR0 = B00111111;
69   // Voltaje de referencia AVcc con condensador
      en AREF,
70   // usando 8bits (valor ADCH), ADC0 (A0)
71   ADMUX = B01100000;
72   // Inicialmente tenemos la menor frecuencia de
      muestreo
73   ADCSRA = ADCSRA_BASE | PS_128;
74   // Trigger: free running
75   ADCSRB = B00000000;
```

```
76    // Empezar las conversiones
77    ADCSRA |= _BV(ADSC);
78  }
79
80  void setup()
81  {
82    // Queremos los datos lo antes posible,
83    // así que utilizamos la máxima velocidad
84    Serial.begin(115200);
85    delay(100);
86
87    init_buffer();
88    init_adc();
89  }
90
91  void loop()
92  {
93    if(bufferFull)
94    {
95      // Enviar datos y reanudar ADC
96      Serial.write((byte*)buffer, BUFFER_SIZE);
97      //Serial.println(int(buffer[128]));
98      Serial.flush();
99      init_buffer();
100     ADCSRA = ADCSRA_OLD;
101   }
102
103   // Configurar el osciloscopio desde la GUI
104   if(Serial.available())
105   {
106     command = Serial.read();
```

```
107
108     switch(command)
109     {
110       case 'A':
111         ADCSRA = ADCSRA_BASE | PS_128;
112         break;
113       case 'B':
114         ADCSRA = ADCSRA_BASE | PS_64;
115         break;
116       case 'C':
117         ADCSRA = ADCSRA_BASE | PS_32;
118         break;
119       case 'D':
120         ADCSRA = ADCSRA_BASE | PS_16;
121         break;
122     }
123   }
124 }
```

2.3.1. Programa por partes

Como ves, el programa es muy sencillo. Básicamente tenemos una configuración básica del ADC en la que lo habilitamos, sin empezar las conversiones, en modo free running y unas cuantas constantes para configurar el preescalado y cambiar así las muestras/s que tomamos.

Tenemos un buffer en RAM de 512 muestras de capacidad que enviaremos al puerto serie para que nuestra aplicación haga con ellos lo que considere. Este buffer es mane-

jado en la rutina de servicio de interrupción, por lo que las variables que allí se manejan son volátiles. *ISR(ADC_vect)* significa: "interrumpe lo que estés haciendo y ejecuta esto cuando tengas un valor del ADC convertido".

Una vez tenemos lleno el buffer, lo indicamos, guardamos el valor de cómo estaba configurado el ADC y deshabilitamos las interrupciones hasta nueva orden porque no queremos seguir leyendo valores mientras estamos enviando los datos.

Para facilitarnos el manejo de bits sueltos, utilizo la macro _BV, que está definida en la documentación de avr y libc. Te recomiendo echar un ojo. La web 'oficial' es avr-libc pero el proyecto se ha movido a GitHub. También puedes consultar la web de Microchip avr-libc.

En la documentación, encontrarás que _BV es una macro equivalente a:

$$\_BV(x) == 1 \ll x$$

Por tanto, con:

$$ADCSRA\& = \tilde{}\_BV(ADEN)$$

estoy diciendo: "en el registro ADCSRA pon a cero (fíjate en la ~ que indica NOT) el bit ADEN".

2.4. Comprobación

Para realizar una primera comprobación, tendrás que hacer un pequeño cambio en el programa, ya que vamos a utilizar el serial plotter. Simplemente queremos ver un valor del buffer:

```
//Serial.write((byte*)buffer, BUFFER_SIZE);
Serial.println(byte(buffer[128]));
```

Ya podemos probar el prototipo utilizando un cable conectado al pin A0. Haremos unas cuantas mediciones. Como tenemos una precisión de 8bits, tenemos 256 valores. Con lo cual:

- 0V será 0
- 5V será 255
- 3.3V será 168 / 169

Aquí, si tienes el Arduino alimentado por un puerto USB habrás notado dos pequeñas cosas: [3]

- el valor para 3.3V es más 172
- pero es que además baila entre 171 y 172

[3] Estos valores pueden diferir en tu equipo.

La primera es debida a que en realidad tus 5V de Vref del USB son más bien 4.9V, [4] y la segunda, que visualizamos en la Fig. 2.1

Figura 2.1: Valor de 3.3V sin condensador adicional

podrias pensar que según pone en *Analog-to-Digital Converter, Overview* tal vez el pin AREF no esté desacoplado, pero un vistazo al esquemático de Arduino te confirmará que sí. Para mejorar la estabilidad en la lectura puedes poner un condensador de 16V y 220 μF entre AREF y GND, obteniendo unas lecturas más limpias como puede verse en la Fig. 2.2

Finalmente, si dejamos el cable al aire, podremos ver como portadora la frecuencia de alimentación de red con sus 50Hz. Y si enviamos los comandos: A, B, C o D; com-

[4] Con lo cual ya sabemos que si queremos medidas más precesisas de tensión deberemos alimentar Arduino mediante el conector de barril y usar el USB solo para la comunicación.

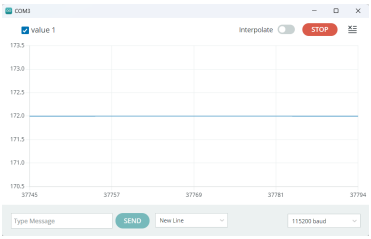

Figura 2.2: Valor de 3.3V con condensador adicional

probamos que se cambia la frecuencia de muestreo obteniendo una imagen diferente.

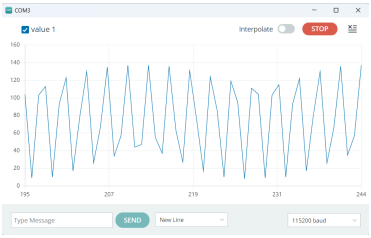

Figura 2.3: Cable al aire con preesacaldo 128

Esta sencilla prueba ya nos sirve para comprobar si nuestra configuración está funcionando. Si dispones de un generador de señales puedes hacer más experimentos, pero recuerda que la señal ha de estar entre 0-5V.

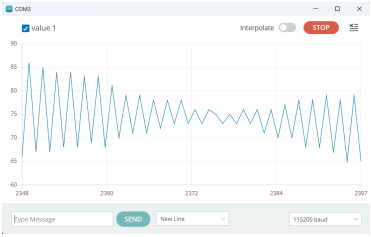

Figura 2.4: Cable al aire con preesacaldo 16

En el siguiente capítulo vamos ahora a añadir funcionalidad, ya que lo que tenemos hasta el momento no nos sirve más que para visualizar formas de onda.

Apéndice A

CA3306

El principal problema de nuestro primer prototipo es la frecuencia de muestreo, que es insuficiente para la mayoría de situaciones donde podamos usar nuestro osciloscopio, por ello vamos a tratar el ADC CA3306, con el que se puede llegar a frecuencias de muestreo de 15Msps, aunque esta vez la resolución del ADC es de solo 6bits, y por tanto únicamente tendremos 64 valores.

Aunque las características de este ADC han sido ampliamente superadas por ADCs más modernos, he escogido este integrado porque es barato, sencillo de utilizar y porque hay muchos proyectos e información interesante. Por ejemplo en la web de Bob Davis podemos ver ejemplos de

5Msps [1] o la web de Paul Sullivan [2] donde recoge el uso de este ADC en la revista Everyday Electronics allá por el 1981.

Bibliografía

[1] Bob Davis. Arduino powered 5 million samples per second oscope with ca3306. https://bobdavis321.blogspot.com/2013/06/arduino-powered-3-million-samples-per.html.

[2] Paul Sullivan. Using the raspberry pi as an oscilloscope. https://paul.sullivan.za.org/raspberry-pi-oscilloscope/.