

Arduino, Tcl/Tk Osciloscopio

José María Sirvent Ichaso

Contents

Antes de empezar	1
¿Qué vas a encontrar en este libro?	2
Un libro dentro de una serie	5
Código fuente	5
 Parte I - Osciloscopio con Arduino	 7
 Adquisición con el ADC de Arduino	 9
Materiales	11
Idea inicial	12
¿Qué nos ofrece Arduino?	13
Programa por partes	21
Comprobación	23

Tcl / Tk	29
Una falsa limitación durante el desarrollo	29
Zoom	29
Cursores	29
Más medidas	29
Análisis espectral	29
Detección por nivel	29
Ficheros CSV	29
DHT	31
Zero-padding	31
¿Por qué aparecen las oscilaciones?	31
Ejemplo intuitivo	31
Limitando el rango espectral	31
Ventanas	31
Rectangular	31
Hann	31
Comparación general	31
Montaje final: frontend analógico y arquitectura de OSCAR	33
Parte física	34
La navaja suiza de la electrónica	34
Inversor	34
No inversor	34
Precauciones con los AO	34

Cuál era la idea inicial	34
Divisor de tensión	34
Amplificador de Ganancia	34
AC / DC	34
Decisiones de diseño	34
Enrutado de la señal	34
Parte lógica - OSCAR	34
Arquitectura	34
Arquitectura cliente/servidor	34
Arquitectura basada en hilos	34
OscThread	34
Evaluación experimental de OSCAR	35
De la resolución nominal a la resolución efectiva .	36
Mediciones	36
FSM	36
Cálculo del ENOB para una señal de 4 kHz . . .	36
Prescaler: 8	36
Muestras analizadas: 1024	36
Frecuencia: 4000.000000 Hz	36
Frecuencia ajustada: 4001.087625 Hz	36
Componente continua: 126.366945	36
Amplitud de pico: 112.282347	36
Amplitud pico a pico: 224.564694	36
Fase: 135.988 grados	36
RMS de la fundamental: 79.395609	36

RMS del residuo: 1.106038	36
Nivel de entrada: -1.138 dBFS	36
SINAD: 37.121 dB	36
ENOB sin corrección: 5.873 bits	36
ENOB referido a full scale: 6.062 bits	36
¿Qué nos dice realmente este resultado?	36
Resultado final	36
Parte II - Superando los límites de Arduino	37
OSCARET - Adquisición de alta velocidad	39
Replicando otras experiencias	40
Captura de datos v1	40
Captura de datos v2	40
Lectura de los datos del ADC	40
Ensamblador	40
Simplificación del circuito	40
Frontend propio	40
Frontend ya diseñado	40
Entender el frontend	40
Modificar el frontend	40
Resultado	40
Modo experimental	40
Limitaciones	40
Conclusiones	41

Antes de empezar

Este libro forma parte de una serie dedicada a Arduino y a cómo ampliar sus posibilidades mediante Tcl/Tk, utilizando un ordenador o un teléfono móvil como complemento. La idea es aprovechar su capacidad gráfica, de procesamiento y de comunicación para crear proyectos más completos alrededor de Arduino.

Este libro presupone unos conocimientos básicos sobre la integración de Arduino con Tcl/Tk. Si todavía no los tienes, te recomiendo comenzar por los libros anteriores de la serie *Arduino Tcl/Tk*:

- *Primeros pasos*
- *Radio FM*

¿Qué vas a encontrar en este libro?

El objetivo de este libro no es únicamente construir un osciloscopio, sino utilizar su desarrollo como hilo conductor para estudiar diferentes aspectos relacionados con la adquisición y el procesamiento de señales, la programación de microcontroladores y la creación de interfaces gráficas mediante Tcl/Tk.

La primera implementación utilizará exclusivamente los recursos disponibles en un Arduino UNO. El conversor analógico-digital integrado en el ATmega328P se encargará de adquirir las muestras, que posteriormente serán enviadas al ordenador y representadas mediante una aplicación desarrollada en Tcl/Tk.

Esta primera versión será deliberadamente sencilla. Su objetivo será comprender los elementos fundamentales que intervienen en cualquier sistema de adquisición: la frecuencia de muestreo, la resolución del conversor, el almacenamiento temporal de las muestras, su transmisión y, finalmente, su representación en pantalla.

Evidentemente, este primer osciloscopio tendrá unas prestaciones modestas. Trabajaremos con una resolución de 8 bits y una frecuencia de muestreo de aproximadamente 150 ksps, lo que en la práctica nos permitirá

visualizar razonablemente bien señales senoidales de hasta unos 19 kHz ¹. Aun con estas limitaciones, veremos que es posible construir una herramienta realmente útil.

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

Para superar esta limitación en la frecuencia de muestreo, incorporaremos en una segunda versión un conversor externo CA3306 de 6 bits. Aunque su resolución es inferior a la del ADC del ATmega328P, su mayor velocidad permite aumentar considerablemente la frecuencia de muestreo. Esta modificación permitirá comprobar una idea que aparecerá repetidamente a lo largo del libro: en el diseño de un instrumento no existe una única característica que determine sus prestaciones, sino que es necesario alcanzar un compromiso entre resolución, velocidad, memoria disponible y capacidad de procesamiento.

Con este último prototipo obtendremos una f_s de 2 Msps de forma estable y 5 Msps de forma experimental, lo que nos permitirá analizar un rango más aceptable de señales.

La tabla siguiente resume las principales diferencias entre las dos etapas del proyecto.

¹Y señales cuadradas sobre 3 kHz

Característica	Versión inicial	Versión CA3306
Conversor A/D	ATmega328P	CA3306
Resolución	8 bits	6 bits
Frecuencia de muestreo	≈ 150 ksp/s	≈ 2 Msp/s
Límite de Nyquist	≈ 75 kHz	≈ 1 MHz
Senoidal máxima	19 kHz	200 kHz
Canales	1	1
Rango de entrada	± 15 V	± 20 V
Visualización T _{cl} /T _{tk}	Sí	Sí
Detección por nivel	Sí	Sí
Medidas automáticas	Sí	Sí
Análisis espectral	Sí	Sí

La frecuencia de muestreo, es decir, el número de muestras adquiridas por segundo, nos permite estimar qué frecuencia máxima podremos representar con una calidad razonable. Por ejemplo, si queremos disponer de entre 8 y 10 muestras por ciclo, podemos obtener una primera aproximación dividiendo la frecuencia de muestreo entre esos valores.

Esta estimación es válida para señales senoidales. En una señal cuadrada también debemos capturar sus armónicos si queremos reproducir correctamente su forma, por lo que el ancho de banda necesario será considerablemente mayor

2.

Un libro dentro de una serie

Este libro es la continuación natural de [Arduino Tcl/Tk: Radio FM](#), en el que, a partir de la documentación del TEA5767, desarrollamos una radio FM funcional con diferentes interfaces. Aquí, por tanto, doy por supuesta cierta soltura en el uso del puerto serie, Tcl/Tk y las hojas de características de los componentes.

La verdad es que hay tantas cosas interesantes y divertidas por construir que la lista podría ser interminable. Esta es una de las razones por las que no he escrito un único libro sobre Arduino y Tcl/Tk, sino una serie. De esta forma evitamos volúmenes excesivamente grandes y podemos ir cerrando, uno a uno, los distintos proyectos de la lista.

Código fuente

Todo el código fuente de los programas del libro lo podrás encontrar en [Arduino Tcl/Tk Osciloscopio](#).

²El mínimo imprescindible sería el 3er armónico, pero lo deseable sería llegar al 7o o más

Parte I - Oscilloscopio con Arduino

Adquisición con el ADC de Arduino

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 la adquisición de señales sin necesidad de añadir hardware especializado. Si posteriormente queremos aumentar 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.

Materiales

La lista de materiales puede ser mínima si queremos:

- Arduino
- 2 cables

Aunque en este caso solo podremos medir y visualizar señales DC ³. 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

³Son señales DC las que no cambian de polaridad.

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 la medida de señales AC, rangos de tensión mayores, amplificación de la señal o incluso canales adicionales.

Idea inicial

Todo surgió hace ya muchos años leyendo una entrada del magnífico blog JeeLabs ⁴, 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

⁴Anteriormente usaba otro dominio.

era posible obtener resultados interesantes, había llegado el momento de dejar de darle vueltas y empezar a hacer mis propias pruebas.

¿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, Lst. 0.1

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

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 que 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

Listing 0.1 Experimento muestras

```
/**
  Vamos a ver cuántas 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);
}

void loop()
{
}
```

resumen de las características que en lugar de nuestros aproximadamente 9 ksps, podemos llegar a 76.9 ksps ⁵. 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 50 kHz-200 kHz si queremos los 10 bits de resolución... pero podemos aumentar la frecuencia de reloj si no necesitamos tanta resolución
- 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:

- $16 \text{ MHz} / 128 = 125 \text{ kHz} / 13 \text{ ciclos/conversión} = 9615 \text{ lecturas/s}$

Vaya, este valor teórico se parece bastante a nuestro valor real. ¡Ya sabemos de dónde sale nuestro valor empírico y

⁵Como veremos, aquí no hemos hecho caso al fabricante y esto tiene sus consecuencias.

saber el porqué es lo verdaderamente interesante ya que nos va a permitir tener la capacidad de manipularlo!

En principio 9 ksp/s 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 8 bits y aumentar la frecuencia de lecturas/s y así tener algo parecido a ¡un osciloscopio!

Con 8 bits de resolución podemos aumentar la frecuencia de reloj a 1 MHz, lo que nos dará una frecuencia de muestreo aproximada de 77 kHz. Aquí, en cambio, hemos forzado un poco más y con 2 MHz llegamos a las 150 ksp/s.

Bien, pues leeremos los 8 bits 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, Lst. 0.2

Listing 0.2 Preescalado ADC

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

    Muestras/s aproximadas:
    -Prescaler 128: 16 MHz / 128 = 125000 / (13 ciclos/lectura)
                    = 9 ksp/s
    -Prescaler 64: 20 ksp/s
    -Prescaler 32: 40 ksp/s
    -Prescaler 16: 77 ksp/s
    -Prescaler 8: 150 ksp/s
*/

// Diferentes preescalados
const byte PS_128 = B00000111;
const byte PS_64 = B00000110;
const byte PS_32 = B00000101;
const byte PS_16 = B00000100;
const byte PS_8 = B00000011;

// Configuración base del ADCSRA:
// -habilitar el ADC sin empezar la conversión
// -usar free running
// -habilitar interrupciones
// -sin preescalado (PS_2)
const byte ADCSRA_BASE = B10101000;

volatile byte ADCSRA_OLD = 0x00;
// Buffer
#define BUFFER_SIZE 1024

volatile byte buffer[BUFFER_SIZE];
volatile int bufferIndex;
volatile byte bufferFull;
```

Listing 0.3 Preescalado ADC

// Comandos GUI -> Arduino`char command = ' ';`*/****Cuando llenamos el buffer paramos las lecturas para
enviar los datos***/*`ISR(ADC_vect)``{` `buffer[bufferIndex] = ADCH;` `bufferIndex = (bufferIndex + 1) % BUFFER_SIZE;` `if(bufferIndex == 0)` `{` `ADCSRA_OLD = ADCSRA;` `ADCSRA &= ~_BV(ADEN);` `bufferFull = true;` `}``}`*/\*\***Inicializar el buffer de lecturas**\*/*`void init_buffer()``{` `bufferIndex = 0;` `bufferFull = false;` `memset((byte*)buffer, 0, BUFFER_SIZE);``}`

Listing 0.4 Preescalado ADC

```
/**
    Configurar el ADC para usarlo como "osciloscopio"
    Lectura en A0
*/
void init_adc()
{
    // Eliminar posibles ruidos deshabilitando el digital input b
    DIDRO = B00111111;
    // Voltaje de referencia AVcc con condensador en AREF,
    // usando 8 bits (valor ADCH), ADC0 (A0)
    ADMUX = B01100000;
    // Inicialmente tenemos la menor frecuencia de muestreo
    ADCSRA = ADCSRA_BASE | PS_128;
    // Trigger: free running
    ADCSRB = B00000000;
    // Empezar las conversiones
    ADCSRA |= _BV(ADSC);
}

void setup()
{
    // Queremos los datos lo antes posible,
    // así que utilizamos una velocidad alta
    Serial.begin(115200);
    delay(100);

    init_buffer();
    init_adc();
}
```

Listing 0.5 Prescalado ADC

```
void loop()
{
    if(bufferFull)
    {
        // Enviar datos y reanudar ADC
        Serial.write((byte*)buffer, BUFFER_SIZE);
        //Serial.println(int(buffer[128]));
        Serial.flush();
        init_buffer();
        ADCSRA = ADCSRA_OLD;
    }

    // Configurar el osciloscopio desde la GUI
    if(Serial.available())
    {
        command = Serial.read();

        switch(command)
        {
            case 'A':
                ADCSRA = ADCSRA_BASE | PS_128;
                break;
            case 'B':
                ADCSRA = ADCSRA_BASE | PS_64;
                break;
            case 'C':
                ADCSRA = ADCSRA_BASE | PS_32;
                break;
            case 'D':
                ADCSRA = ADCSRA_BASE | PS_16;
                break;
        }
    }
}
```

Listing 0.6 Preescalado ADC

```
    case 'E':  
        ADCSRA = ADCSRA_BASE | PS_8;  
        break;  
    }  
}  
}
```

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 1024 muestras de capacidad que enviaremos al puerto serie para que nuestra aplicación haga con ellos lo que considere. Este buffer es manejado 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”.

Inicialmente, y siendo conservador, utilicé 512 muestras, ya que Arduino UNO dispone únicamente de 2 kB de RAM

y por tanto, si usaba demasiada memoria RAM la pila podría corromper datos y si el programa fallaba, no sabría si era problema del propio programa o de memoria. Una vez terminado el prototipo completo funcionando correctamente con 512 puntos, decidí tomar una posición más agresiva y probar con el doble de puntos. Como todas las pruebas fueron bien, adopté el cambio.

Una vez tenemos lleno el buffer, lo indicamos, guardamos el valor de cómo estaba configurado el ADC y deshabilitamos el ADC 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](#).

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

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

Por tanto, con:

```
ADCSRA &= ~_BV(ADEN);
```

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

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 8 bits, 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: ⁶

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

La primera es debida a que en realidad tus 5V de Vref del

⁶Estos valores pueden diferir en tu equipo.

USB son más bien 4.9V⁷, ya que $N \approx 256 \frac{V_{in}}{V_{ref}} = 256 \frac{3.3}{4.9} = 172$, y la segunda la visualizamos en la Fig. 1

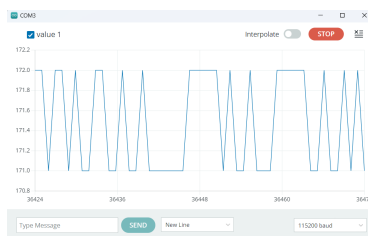


Figure 1: Valor de 3.3V sin condensador adicional

Arduino UNO ya incorpora un condensador de 100 nF entre AREF y GND, según recomienda Atmel en el apartado *Analog-to-Digital Converter, Overview*, como puede comprobarse en su [esquemático de Arduino](#). Sin embargo, en mis pruebas seguía observando una pequeña fluctuación de la lectura entre dos valores consecutivos. Experimentalmente comprobé que añadiendo un condensador electrolítico de 220 μ F/16 V entre AREF y GND (en paralelo con el condensador existente) la lectura se hacía más estable, como puede verse en la Fig. 2.

⁷En mi montaje, alimentar Arduino mediante el conector de barril y utilizar el USB únicamente para la comunicación proporciona una alimentación más estable.

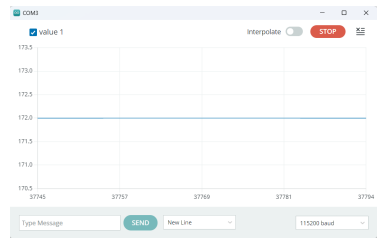


Figure 2: Valor de 3.3V con condensador adicional

Una nota importante es que estabilidad no es lo mismo que exactitud. Que la lectura deje de oscilar entre 171 y 172 significa que se ha mejorado su repetibilidad o estabilidad; no significa necesariamente que el valor medido sea más exacto, Fig. 3

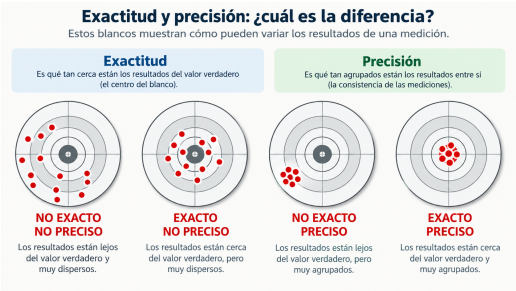


Figure 3: Exactitud vs precisión

Una fluctuación de un solo código en un ADC de 8 bits tampoco es extraordinaria: con una referencia de unos 5 V, 1 LSB representa aproximadamente 19,5 mV. Puede proceder de ruido de la referencia, del propio ADC, de la señal de entrada o simplemente de estar cerca del umbral entre dos códigos.

Finalmente, si dejamos el cable al aire, capta la interferencia de la red eléctrica y aparece una componente dominante de 50 Hz. Y si enviamos los comandos: A, B, C, D o E; comprobamos que se cambia la frecuencia de muestreo obteniendo una imagen diferente, Fig. 4 y Fig. 5.

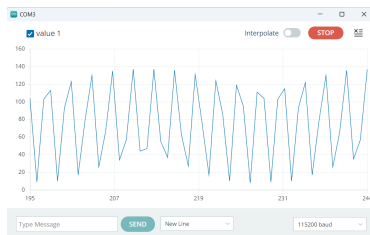


Figure 4: Cable al aire con preescalado 128

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.

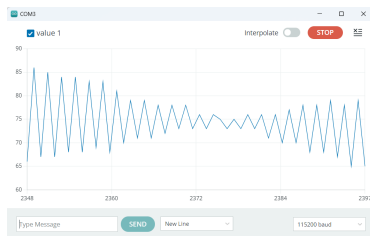


Figure 5: Cable al aire con preescalado 16

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

Tcl / Tk

Una falsa limitación durante el desarrollo

Zoom

Cursores

Más medidas

Análisis espectral

Detección por nivel

Ficheros CSV

DHT

Zero-padding

¿Por qué aparecen las oscilaciones?

Ejemplo intuitivo

Limitando el rango espectral

Ventanas

Rectangular

Hann

Comparación general

Montaje final: frontend analógico y arquitectura de OSCAR

Parte física

La navaja suiza de la electrónica

Inversor

No inversor

Precauciones con los AO

Cuál era la idea inicial

Divisor de tensión

Amplificador de Ganancia

AC / DC

Decisiones de diseño

Enrutado de la señal

Evaluación experimental de OSCAR

De la resolución nominal a la resolución efectiva

Mediciones

FSM

Cálculo del ENOB para una señal de 4 kHz

Prescaler: 8

Muestras analizadas: 1024

Frecuencia: 4000.000000 Hz

Frecuencia ajustada: 4001.087625 Hz

Componente continua: 126.366945

Amplitud de pico: 112.282347

Parte II - Superando los límites de Arduino

OSCARET - Adquisición de alta velocidad

Replicando otras experiencias

Captura de datos v1

Captura de datos v2

Lectura de los datos del ADC

Ensamblador

Simplificación del circuito

Frontend propio

Frontend ya diseñado

Entender el frontend

Modificar el frontend

Resultado

Modo experimental

Conclusiones

Cuando comenzamos este proyecto, el objetivo parecía relativamente sencillo: utilizar el conversor analógico-digital de Arduino para capturar una señal y representarla en la pantalla del ordenador. Sin embargo, a medida que avanzábamos fueron apareciendo muchos de los problemas que encontramos en cualquier sistema real de adquisición de datos.

Partimos de las aproximadamente 9 ksp/s que proporciona `analogRead()` en un Arduino UNO y, modificando directamente la configuración del ADC, conseguimos llevar la frecuencia de muestreo hasta aproximadamente 150 ksp/s. Para ello tuvimos que renunciar a parte de la resolución efectiva del conversor y trabajar con los 8 bits más significativos de cada muestra. Esta primera evolución ya puso

de manifiesto una de las ideas fundamentales del libro: en instrumentación casi siempre estamos trabajando con compromisos. Aumentar la velocidad, la resolución, el número de muestras o el ancho de banda suele tener un coste en alguna otra parte del sistema.

OSCAR terminó siendo mucho más que un programa para dibujar muestras. Incorporamos cursores, medidas automáticas, análisis espectral, detección por nivel, exportación de datos a CSV y distintas posibilidades de visualización. También vimos que disponer de una frecuencia de muestreo elevada no significa automáticamente disponer del mismo ancho de banda, y que el límite de Nyquist no debe confundirse con las prestaciones reales del instrumento.

El análisis espectral nos llevó a estudiar la Transformada Discreta de Hartley, la resolución frecuencial, el *zero-padding* y las funciones de ventana. Comprobamos que muchas veces lo que aparece en una gráfica no es una nueva componente de la señal, sino una consecuencia del intervalo finito de observación, de la fuga espectral o de la forma en la que hemos procesado las muestras.

También descubrimos que la adquisición no termina cuando el ADC entrega un valor. Las muestras deben almacenarse, transmitirse y reconstruirse correctamente.

Los problemas de desalineación de las tramas obligaron a incorporar una cabecera y una máquina de estados para sincronizar la comunicación. Este problema, aparentemente alejado de la electrónica analógica, terminó siendo imprescindible para poder realizar medidas fiables.

La evaluación del ENOB nos permitió comprobar experimentalmente algo que hasta entonces solo habíamos intuitido. Aunque OSCAR maneja muestras de 8 bits, eso no significa que esos 8 bits sean realmente significativos. Para una señal de 4 kHz y utilizando el ADC con prescaler 8 obtuvimos un ENOB normalizado próximo a 6 bits. No se trata de un valor absoluto del ATmega328P, sino del resultado del conjunto completo bajo unas condiciones determinadas: ADC, circuito analógico, alimentación, reloj, generador de señales y método de medida.

Este resultado resume bastante bien la filosofía del proyecto. Llevar el ADC hasta unos 150 ksp/s resulta muy útil para visualizar señales de mayor frecuencia, pero la velocidad adicional tiene un precio en términos de calidad de conversión. No existe, por tanto, una única configuración óptima para todas las situaciones.

El diseño del *frontend* analógico también nos enseñó que los modelos ideales tienen límites. Sobre el papel, un amplificador operacional puede parecer una solución casi per-

fecta, pero al llevar el circuito a la práctica aparecen el ruido, las impedancias, las capacidades parásitas, el *slew rate*, el ancho de banda y los límites de tensión de entrada y salida. Algunas soluciones que parecían razonables inicialmente tuvieron que ser descartadas o modificadas después de comprobar su comportamiento real.

Algo parecido ocurrió en el software. OSCAR pasó de una aplicación monolítica a una arquitectura cliente/servidor y finalmente a una implementación basada en hilos. El objetivo ya no era simplemente hacer funcionar el programa, sino conseguir que el análisis de las señales pudiera realizarse sin bloquear la interfaz gráfica y manteniendo al mismo tiempo una aplicación sencilla de distribuir.

La segunda parte del libro llevó esta misma filosofía un paso más allá. Con OSCARET sustituimos el ADC interno de Arduino por un CA3306 de 6 bits. Perdimos resolución nominal, pero ganamos una frecuencia de muestreo mucho mayor. De esta forma conseguimos trabajar de manera estable alrededor de 2 Msps y alcanzar aproximadamente 5.33 Msps en un modo experimental.

Esta vez el límite dejó de estar principalmente en el ADC. El CA3306 puede funcionar a frecuencias considerablemente mayores, pero el ATmega328P no puede leer y almacenar indefinidamente a esa velocidad. El

cuello de botella se desplazó desde el conversor hasta el microcontrolador. Esta es otra de las conclusiones importantes del proyecto: cuando mejoramos una parte de un sistema, normalmente aparece un nuevo límite en otra.

El aumento de frecuencia obligó además a prestar mucha más atención al *frontend*. Capacidades que a unos pocos kilohertz podían ignorarse, comenzaron a limitar el ancho de banda, y fue necesario estudiar técnicas de compensación similares a las utilizadas en las sondas de osciloscopio. También recurrimos al ensamblador para controlar con precisión el número de ciclos empleados en cada adquisición, algo que no había sido necesario en OSCAR.

Al final hemos construido dos instrumentos con filosofías diferentes. OSCAR ofrece una resolución nominal mayor y explota prácticamente al máximo los recursos internos del Arduino UNO. OSCARET sacrifica resolución para conseguir una velocidad de adquisición muy superior. Ninguno de los dos pretende competir con un osciloscopio comercial; su valor está en que podemos estudiar cada una de sus partes, modificarla y comprobar directamente qué consecuencias tiene cada decisión.

Pero posiblemente el resultado más interesante del proyecto no sean los 150 ksp/s de OSCAR ni los 2 Msps

de OSCARET. Lo realmente importante es todo lo que ha aparecido por el camino: ADC, muestreo, aliasing, cuantificación, ENOB, ancho de banda, amplificadores operacionales, compensación, PWM, temporizadores, ensamblador, procesamiento espectral, comunicaciones, máquinas de estados, hilos y diseño de interfaces.

En muchos momentos hemos tenido que revisar una idea que inicialmente parecía correcta. Hemos encontrado errores, limitaciones inesperadas y resultados que no coincidían con lo previsto. En lugar de ocultarlos, los hemos utilizado como parte del proceso. En ingeniería, que algo no funcione como esperábamos no significa necesariamente que el experimento haya fracasado; muchas veces es precisamente el momento en el que empezamos a entender realmente el problema.

Por eso este libro nunca ha tratado únicamente de construir un osciloscopio. El osciloscopio ha sido el hilo conductor que nos ha permitido estudiar cómo se diseña un sistema completo, desde la señal analógica que llega al conector de entrada hasta los datos que finalmente aparecen representados en la pantalla.

Y, como suele ocurrir con este tipo de proyectos, todavía quedan muchas posibilidades abiertas. Más memoria permitiría realizar capturas mucho más largas; un micro-

controlador más rápido permitiría aprovechar mejor el CA3306; otros ADC podrían aumentar simultáneamente resolución y velocidad; un mejor *frontend* permitiría incrementar el ancho de banda y la impedancia de entrada; y el software podría ampliarse con nuevas medidas, decodificación de protocolos o nuevos módulos de análisis.

Pero esas mejoras pertenecen ya a otro proyecto.

Por ahora hemos llegado suficientemente lejos: empezamos leyendo un pin analógico de Arduino y hemos terminado construyendo, entendiendo y caracterizando dos sistemas de adquisición completamente funcionales.

Y, sobre todo, hemos aprendido por qué funcionan.

