dUQx

v 2.1

Alexander López Parrado
Ingeniería Electrónica
Universidad del Quindío.

Este documento describe la tarjeta de adquisición de datos y control USB dUQx (pronunciado como "diuks"). La tarjeta está orientada a la construcción de sistemas de control e instrumentación sobre los sistemas operativos: Linux y Windows. En los dos sistemas operativos dUQx dispone de una API multiplataforma en lenguaje C y MATLAB. Adicionalmente, esta nueva versión (dUQx-2.1) es basada en la tarjeta Arduino UNO R3 [1], así que solo es necesario programar la memoria flash del microcontrolador con el firmware dUQx-2.1.

Los archivos para el desarrollo de aplicaciones con dUQx incluyendo este manual de usuario se encuentran en el archivo comprimido dUQx-2.1.zip, de ahora en adelante la ruta <raíz elegida>\dUQx-2.1 hace referencia al directorio donde fue descomprimido dUQx2-0.zip.

1. Características de dUQx

dUQx es una tarjeta de adquisición de datos USB de muy bajo costo, sin embargo cuenta con todos los recursos para realizar lectura como generación de señales analógicas y digitales.

dUQx dispone de un puerto digital de 10 líneas, cada bit del puerto digital puede ser configurado independientemente como entrada o salida. La tabla muestra la correspondencia entre los bits del puerto y los pines de la tarjeta Arduino UNO.

Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PB5	PB4	PB3	PB0	PD7	PD6	PD5	PD4	PD3	PD2
pin 13	pin 12	Pin 11	pin 8	pin 7	pin 6	pin 5	pin 4	pin 3	pin 2

dUQx no cuenta con un DAC, sin embargo se dispone de dos salidas analógicas emuladas mediante dos canales PWM saliendo por los pines PB1 y PB2 (pines 9 y 10 de la tarjeta Arduino UNO). La Figura 1 muestra una señal PWM tal y como se genera por dUQx.

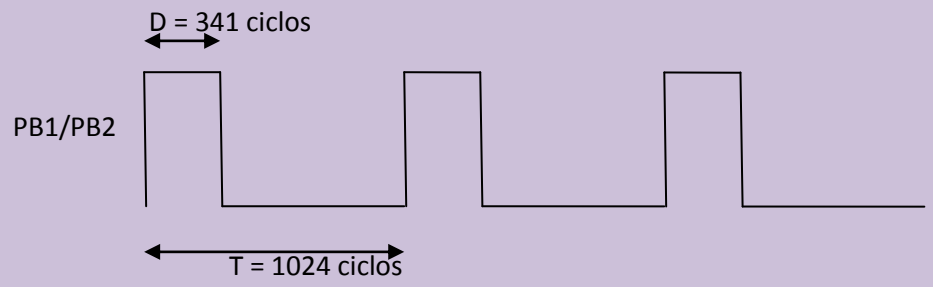

Figura 1. Señales PWM complementarias.

La frecuencia de las señales PWM es de 15.625 kHZ, lo que implica que cada periodo se compone de 16 MHz / 15625 = 1024 ciclos, esto equivale a tener dos DAC de $\log_2(1024) = 10$ bits.

Si se desea obtener una señal de pseudo-DC sólo es necesario adicionar un filtro pasa-bajo con una frecuencia de corte inferior a 15.625 kHZ.

dUQx dispone de 6 entradas analógicas unipolares (Pines A0 a A5 de la tarjeta Arduino UNO), hay que tener en cuenta que se usa el voltaje de referencia del pin AVCC, por lo que dUQx puede leer voltajes entre 0 V y 5.0 V. El ADC de dUQx puede usar una resolución de 10 bits u 8 bits, en el primer caso la resolución es de 5.0 v/1024 = 4.88 mV y en el segundo de 5.0 v/256 = 19.53 mV.

dUQx puede realizar capturas de dos modos: única muestra y bloques. En el modo única muestra la tasa de muestreo máxima es de 250 Hz aproximadamente y para el modo bloque es de 7 kHz, estos valores dependen del sistema operativo y el lenguaje utilizado; para el modo de 8 bits el tamaño máximo del bloque es de 255 muestras y para el modo de 10 bits el tamaño máximo del bloque es de 127 muestras.

3.1 Acceso usando lenguaje C

El acceso usando lenguaje C ha sido probado con el entorno de desarrollo Code::Blocks tanto en Linux como en Windows. Por lo que se asume que el computador tiene instalado Code::Blocks y las

herramientas de compilación de lenguaje C (GCC en Linux y MINGW o CYGWIN en Windows).

Los archivos con el código fuente para el acceso a dUQx usando lenguaje C se encuentran en la ruta <raíz elegida>\dUQx-2.1\portable\c\src\, estos deben ser adicionados al proyecto del usuario. Los archivos dUQx.c y dUQx.h contienen la implementación y los prototipos, respectivamente, de las funciones que permiten acceder desde un PC con Linux o Windows a todas las funcionalidades de dUQx.

Adicional a los archivos de código fuente para el acceso a dUQx, se incluye un programa de ejemplo en <raíz elegida>\dUQx-2.1\portable\c\test\ así como un proyecto de prueba ubicado en <raíz elegida> \dUQx-2.1\portable\c\gcc\codeblocks\ para Code::Blocks. Nótese que el código fuente es multiplataforma y se puede compilar tanto en Linux como en Windows. Al abrir el proyecto de prueba, asegúrese de elegir la configuración adecuada (Debug o Release) en el sistema operativo que esté utilizando, esto se muestra en la Figura 2.

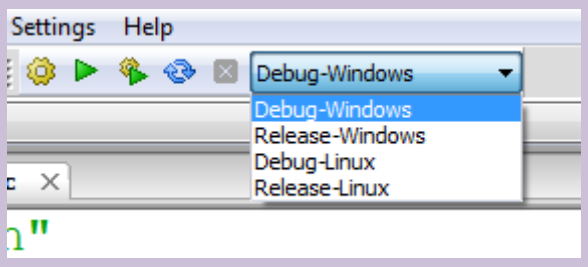

Figura 2. Configuración a usar dependiendo del sistema operativo.

Para compilar cualquier programa que use dUQx usando el entorno de desarrollo Code::Blocks se debe crear un proyecto en C y adicionar al proyecto los archivos con extensión .c almacenados en <raíz elegida>\dUQx-2.1\portable\c\src\. Este directorio también se debe adicionar a las rutas de búsqueda de archivos de cabecera de Code::Blocks.

En el caso de Windows es necesario instalar el controlador suministrado con el entorno de desarrollo de Arduino (https://www.arduino.cc/download_handler.php).

5.1 Acceso usando MATLAB

Para acceder a dUQx desde MATLAB se usan las funciones para acceso a puertos seriales. En la carpeta <raíz elegida>\dUQx-2.1\portable\MATLAB\ se encuentran los archivos .m para acceder a

dUQx desde MATLAB. Para utilizar los *scripts* sólo es necesario que la carpeta <raíz elegida>\dUQx-2.1\portable\MATLAB\ sea adicionada a las rutas de búsqueda de MATLAB. Se incluyen dos *scripts* de ejemplo que prueba dUQx y su desempeño en MATLAB.

2. Sobre los programas de prueba

Para todos los programas de prueba suministrados en los dos sistemas operativos y lenguajes de programación, incluyendo MATLAB; se mide el tiempo para realizar una escritura tanto al puerto digital como el análogo así como el tiempo para realizar una lectura del puerto digital y de uno de los canales analógicos.

Tomar nota de estos tiempos ya que estos limitan los sistemas que se pueden controlar y/o instrumentar con dUQx.

3. Anexos

A. API de dUQx en lenguaje C

A continuación se describen las funciones para el acceso a dUQx usando lenguaje C. Esta API es aplicable para el acceso desde Linux y Windows.

```
uint8_t ch dUQx_Init(int serport);
```

Esta función inicia dUQx. Esta función debe ser llamada antes de iniciar el uso de dUQx, **`dUQx_Init`** retorna 0 si dUQx se inició correctamente o 1 si hubo algún error.

El argumento `serport` es un número entero que identifica el puerto serial asociado a la tarjeta Arduino UNO, éste debe ser elegido a partir de los valores en la siguiente tabla,

	Linux	windows
0	ttyS0	COM1
1	ttyS1	COM2
2	ttyS2	COM3
3	ttyS3	COM4
4	ttyS4	COM5
5	ttyS5	COM6
6	ttyS6	COM7

7	ttyS7	COM8
8	ttyS8	COM9
9	ttyS9	COM10
10	ttyS10	COM11
11	ttyS11	COM12
12	ttyS12	COM13
13	ttyS13	COM14
14	ttyS14	COM15
15	ttyS15	COM16
16	ttyUSB0	n.a.
17	ttyUSB1	n.a.
18	ttyUSB2	n.a.
19	ttyUSB3	n.a.
20	ttyUSB4	n.a.
21	ttyUSB5	n.a.
22	ttyAMA0	n.a.
23	ttyAMA1	n.a.
24	ttyACM0	n.a.
25	ttyACM1	n.a.
26	rfcomm0	n.a.
27	rfcomm1	n.a.
28	ircomm0	n.a.
29	ircomm1	n.a.

```
void dUQx_End(void);
```

Esta función finaliza dUQx. Debe ser llamada al finalizar el uso de dUQx.

```
void dUQx_ConfigureDigital(uint16_t ddr);
```

Configura el puerto digital de dUQx, un 1 en determinado bit de `ddr` configura la línea correspondiente del puerto digital como salida. Por ejemplo:

```
dUQx_ConfigureDigital(0x3ff);
```

Configura todos los pines del puerto digital como salida. Debe tenerse en cuenta que sólo se consideran los 10 bits de menor peso de `ddr`.

```
void dUQx_WriteDigital(uint16_t port);
```

Escribe el dato `port` en el puerto digital. Si el pin está configurado como salida el valor correspondiente aparecerá en el pin, si el pin está configurado como entrada y el bit correspondiente en `port` es 1 se activará una resistencia de pull-up. Debe tenerse en cuenta que sólo se consideran los 10 bits de menor peso de `port`.

```
void dUQx_ReadDigital(uint16_t *pin);
```

Lee un dato del puerto digital, el dato es almacenado en la dirección apuntada por `pin`. Si el pin está configurado como entrada se retorna su estado 1 ó 0, si el pin está configurado como salida se retorna el último valor escrito en él. Debe tenerse en cuenta que sólo se debe considerar los 10 bits de menor peso del número almacenado en la dirección dada por `pin`.

```
void dUQx_CalibrateAnalog(double *vref);
```

Calibra dUQx realizando una estimación del voltaje de referencia. La estimación del voltaje de referencia es almacenada en la dirección apuntada por `vref`. Esta función debería ser llamada una vez antes de utilizar las funciones `dUQx_WriteAnalog` y `dUQx_ReadAnalog`.

```
void dUQx_WriteAnalog(double vo, double vref,uint8_t ch);
```

Escribe el voltaje `vo` en la salida analógica indicada por **`ch`**, si **`ch=0`** el voltaje se obtendrá por el pin PB1 y si **`ch=1`** el voltaje se obtendrá por el pin PB2. En todo caso este voltaje corresponde al nivel de DC de la señal PWM del pin PB1/PB2. El voltaje de referencia `vref` debería ser el valor retornado por `dUQx_CalibrateAnalog`.

```
void dUQx_ReadAnalogSingle(uint8_t ch,double vref,double *
v)
```

Lee una única muestra del voltaje del canal análogo `ch` y lo retorna en la dirección apuntada `vi`. El voltaje de referencia `vref` debería ser el valor retornado por `dUQx_CalibrateAnalog`.

`uint8_t dUQx_ReadAnalogBuffer(uint8_t ch,double vref,double ** v,uint8_t n)`

Lee **`n`** muestras del canal analógico `ch`. **`dUQx_ReadAnalogBuffer`** retorna la cantidad de muestras capturadas y **`v`** queda apuntando al lugar donde estas fueron almacenadas. El voltaje de referencia `vref` debería ser el valor retornado por `dUQx_CalibrateAnalog`. La cantidad de muestras solicitadas, **`n`**, debe ser menor o igual a 255 en el modo de 8 bits y menor o igual a 127 en el modo de 10 bits. Ver función **`dUQX_SetResolution`**.

El siguiente ejemplo muestra como leer 100 muestras del canal analógico 0.

```
double *vp,vref;
uint8_t len;

...

dUQx_CalibrateAnalog(&vref);
len=dUQx_ReadAnalogBuffer(0,vref,&vp,100);
```

`void dUQX_SetResolution(uint8_t r)`

Fija la resolución de captura del ADC a 8 bits si **`r`** vale cero o a 10 bits si **`r`** vale uno.

`void dUQx_ADCPreescalerSet(uint8_t ps)`

Fija el valor del preescalador para el reloj del ADC de la siguiente forma.

`ps` == 0, reloj del ADC es 16000000/2

`ps` == 1, reloj del ADC es 16000000/2

`ps` == 2, reloj del ADC es 16000000/4

`ps` == 3, reloj del ADC es 16000000/8

ps == 4, reloj del ADC es 16000000/16

ps == 5, reloj del ADC es 16000000/32

ps == 5, reloj del ADC es 16000000/64

ps == 7, reloj del ADC es 16000000/128

```
void dUQx_ADCEnabledSet(uint8_t e)
```

Habilita el ADC cuando **e** vale uno o lo inhabilita cuando **e** vale cero.

B. API de dUQx usando MATLAB

A continuación se describen las funciones para el acceso a dUQx desde MATLAB. Revisar el *script* <raíz elegida> \dUQx2.1\portable\MATLAB\test.m para un ejemplo de utilización de dUQx con MATLAB.

```
function duqx_init(serport)
%Inicia la tarjeta de adquisición de datos.
%serport: índice del puerto serial
%Retorna 1 si está conectada
%Retorna 0 si está desconectada
%
%Alexander López Parrado(2017)


function duqx_configure_digital(ddr)
%Configura el puerto digital usando como registro de
dirección de datos a
%ddr. Un bit en 1 en ddr configura como salida el pin
respectivo, de lo
%contrario lo configura como entrada.
%
%Alexander López Parrado(2017)



function duqx_write_digital(port)
%Escribe el valor de port en el puerto digital
%
%Alexander López Parrado(2017)


function pin=duqx_read_digital()
```

```
%Lee el puerto digital y retorna su estado en pin
%
%Alexander López Parrado(2017)


function duqx_write_analog(v,vref,ch)
%Escribe el voltaje v en la salida analógica indicada por
ch usando a vref como voltaje de
%referencia.
%
%%Alexander López Parrado(2017)


function v=duqx_read_analog_single(ch,vref)
%Lee una única muestra del canal análogo ch usando como
voltaje de
%referencia a vref. El voltaje leído es retornado en v.
%
%Alexander López Parrado(2017)

function v=duqx_read_analog_buffer(ch,vref,n)
%Leer hasta n muestras del canal analógico ch usando vref
como
%voltaje de referencia.
%
%n debe ser menor o igual a 255 para el modo de 8 bits por
muestra.
%n debe ser menor o igual a 127 para el modo de 10 bits por
muestra.
%
%Las muestras capturadas son retornadas en el vector v.
%
%%Alexander López Parrado(2017)


function vref=duqx_calibrate_analog()
%Retorna en vref el voltaje referencia para los canales
analógicoos.
%
%Alexander López Parrado(2017)


function duqx_set_resolution(r)
%Fija el tamaño de bits por muestra capturada en el canal
análogo en 8 bits (r=0) o 10 bits (r=1).
%
%El tamaño por defecto usado por dUQx es de 10 bits
%
%Alexander López Parrado(2017)

function duqx_adc_preescaler_set(ps)
```

```
%Fija a ps el preescalador del reloj del ADC
%
%ps == 0, reloj del ADC es 16e6/2
%ps == 1, reloj del ADC es 16e6/2
%ps == 2, reloj del ADC es 16e6/4
%ps == 3, reloj del ADC es 16e6/8
%ps == 4, reloj del ADC es 16e6/16
%ps == 5, reloj del ADC es 16e6/32
%ps == 5, reloj del ADC es 16e6/64
%ps == 7, reloj del ADC es 16e6/128
%
%por defecto el preescalador del ADC es de 128
%
%Alexander López Parrado (2017)

function duqx_adc_enabled_set(e)
%Habilita (e=1) o inhabilita (e=0)  el ADC
%
%%Alexander López Parrado (2017)
```

4. Referencias

[1] ARDUINO UNO, Arduino AG, Disponible en https://www.arduino.cc/en/Main/ArduinoBoardUno.