

INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

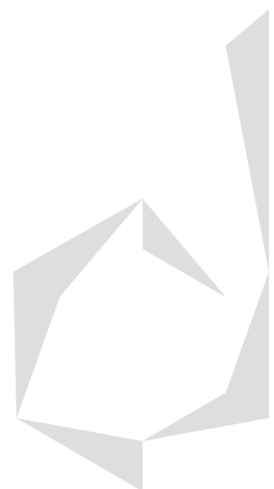
ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Encoder Rotativo y
Joystick Analógico

Contenido

Encoder Rotatorio: Control Digital	2
Módulo KY-040.....	5
Código Arduino Interrupciones Externas – Encoder Rotativo: Control Digital:	7
Máquina de Estados: Identificación de Giro en Encoder Rotativo	9
Divisor de Reloj: Módulo DIV	9
Retardo o Delay Antirrebotes (Delay, Millis y Shift Register)	10
Código VHDL:	11
Diagrama de Estados:	11
Máquina de Estados Tipo Moore:	12
Código UCF:.....	14
Joystick: Control Analógico	14
Módulo KY-023.....	15
Código Arduino – Joystick: Control Analógico:	16
Referencias.....	18



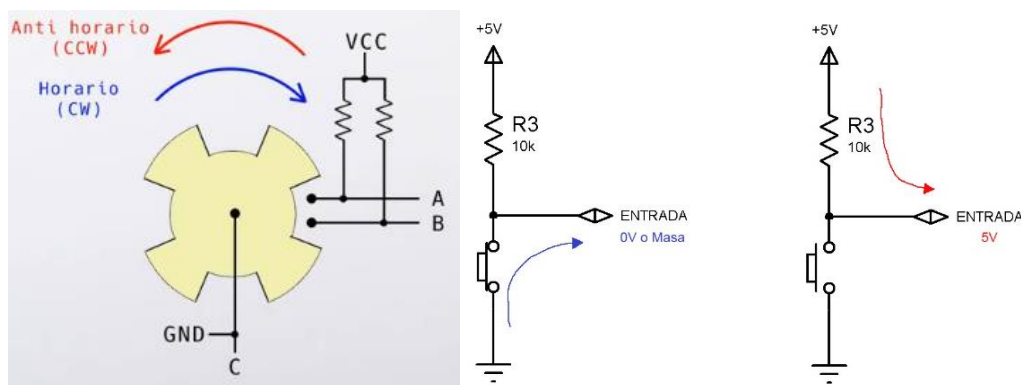
Encoder Rotatorio: Control Digital

El encoder rotativo es un **dispositivo de control digital** que **convierte el movimiento mecánico en una señal binaria que puede ser leída por un sistema de control**, como un microcontrolador, CPU, FPGA, etc. Físicamente parece un potenciómetro, ya que tiene un eje que se puede girar en **sentido horario** o **antihorario** y al hacerlo **genera señales digitales de salida**, además, cuando su eje gira, se siente que el mismo da pequeños pasos o clics donde se detiene, en específico **el módulo KY-040** (que es el más usado) **requiere 30 de estos pasos para dar un giro completo de 360°** y no tiene un límite de giro hacia ninguno de los dos lados, por lo que **no tiene una posición absoluta, la posición siempre será relativa al valor anterior**.

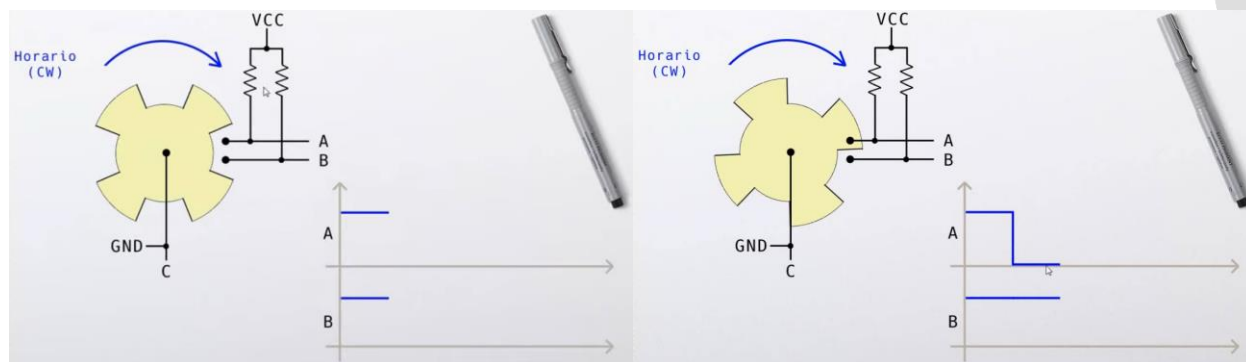
El giro del encoder se percibe si es a la **derecha (sentido horario CW)** o a la **izquierda (sentido antihorario CCW)** porque uno de sus **pines de señal A o B** entra en contacto con la placa dentada de su base (pin **GND = C**) antes que el otro y como están conectadas a **resistencias pull up**, cambian su valor binario de 1 a 0 y 0 a 1 en una secuencia específica, así es como se percibe la dirección del giro.

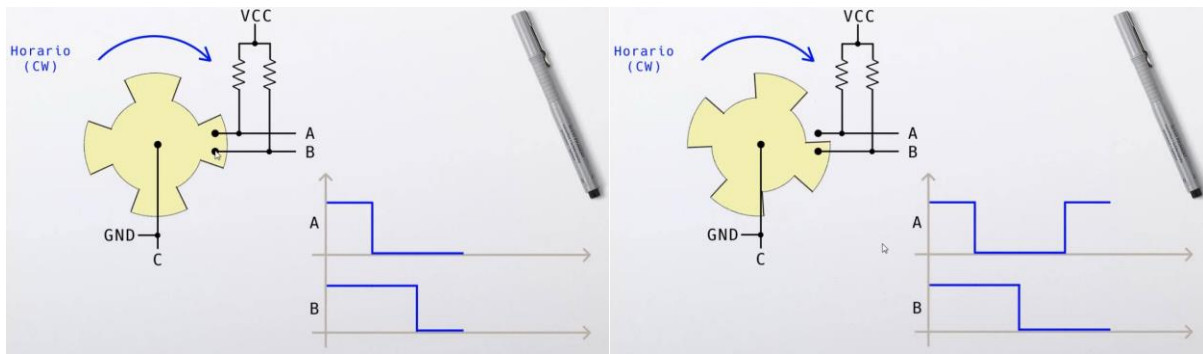
Como las resistencias internas son de tipo **pull up**, los **pines de señal A o B** entregarán:

- **Un 1 lógico**: Cuando el switch esté abierto, osea **cuando el pin de señal no esté haciendo contacto con la placa dentada GND del eje**.
- **Un 0 lógico**: Cuando el switch esté cerrado, osea **cuando el pin de señal sí esté haciendo contacto con la placa dentada GND del eje**.

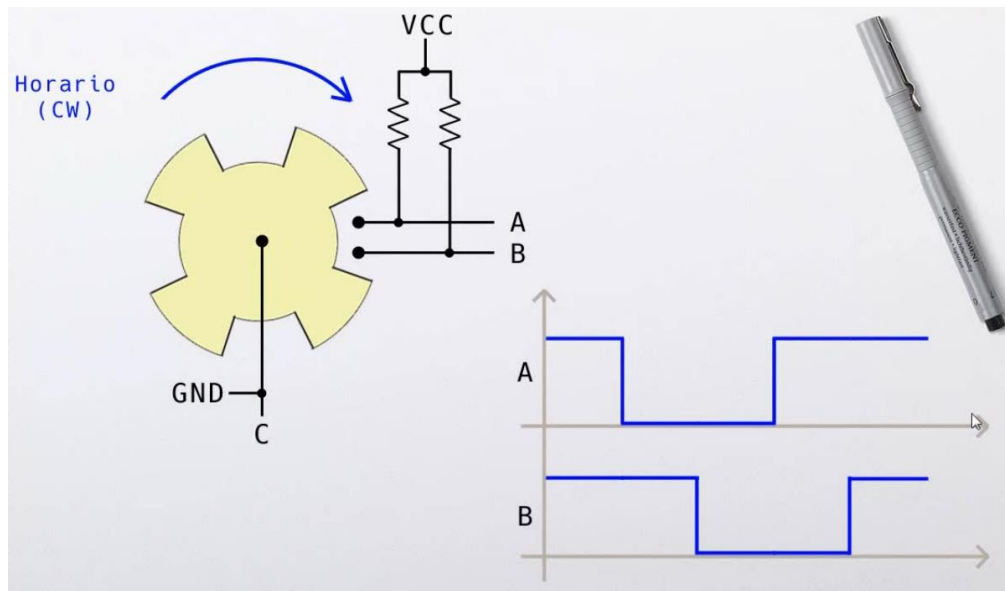


Las señales producidas en los **pines A y B** cuando se gira el encoder **de un paso a otro en sentido horario (CW o hacia la derecha)** son las siguientes, tomando en cuenta que **para que el encoder dé una vuelta completa dará 30 de estos pasos**:





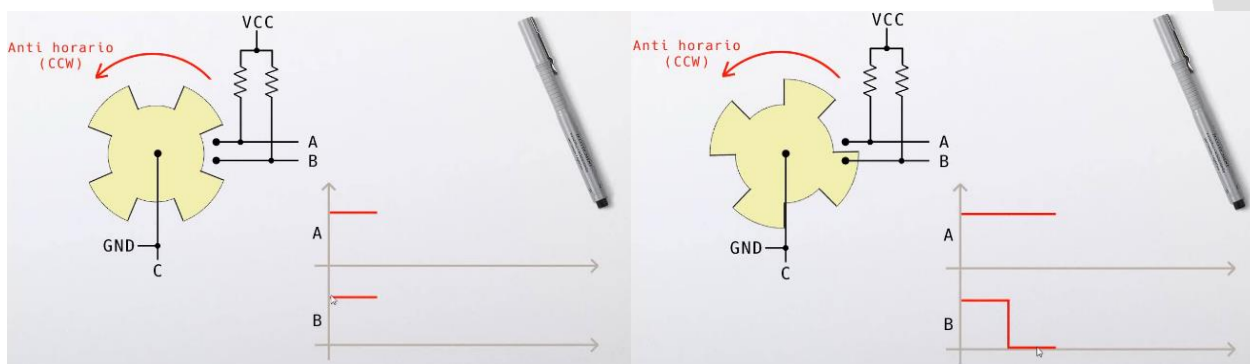
Las señales del encoder terminan en su estado inicial (1 lógico en ambas), por lo que se concluye que **el giro CW** se puede identificar cuando **el pin de señal A pasa de ser 1 a 0 y viceversa antes que el B**.

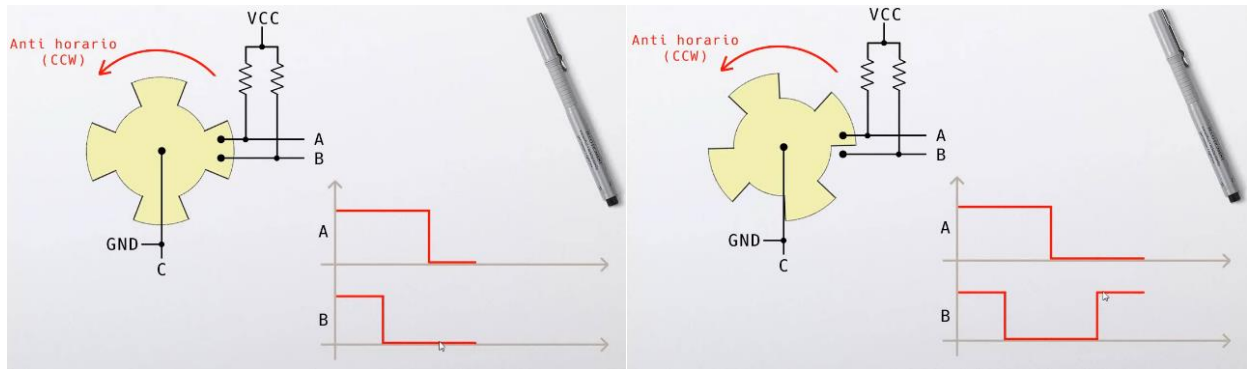


Cuando giramos el codificador en **sentido horario** el cambio de valor se da primero en el **pin A** y más tarde en el **pin B**, estas señales aparecen durante el movimiento de **1 de los 30 pasos del eje** y se representa de la siguiente manera: $(\text{PinA}, \text{PinB}) = (1, 1) \rightarrow (0, 1) \rightarrow (0, 0) \rightarrow (1, 0)$.

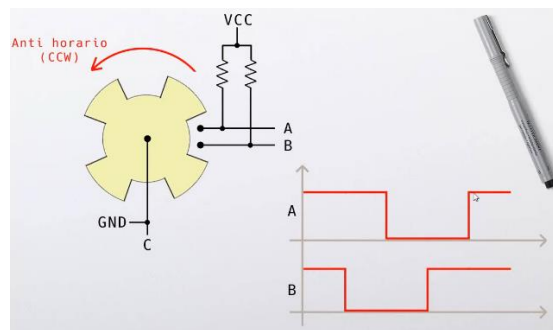
A → B

Los pulsos digitales causados en los **pines de señal A y B** cuando se gira el encoder **de un paso a otro** en **sentido antihorario (CCW)** son las siguientes:





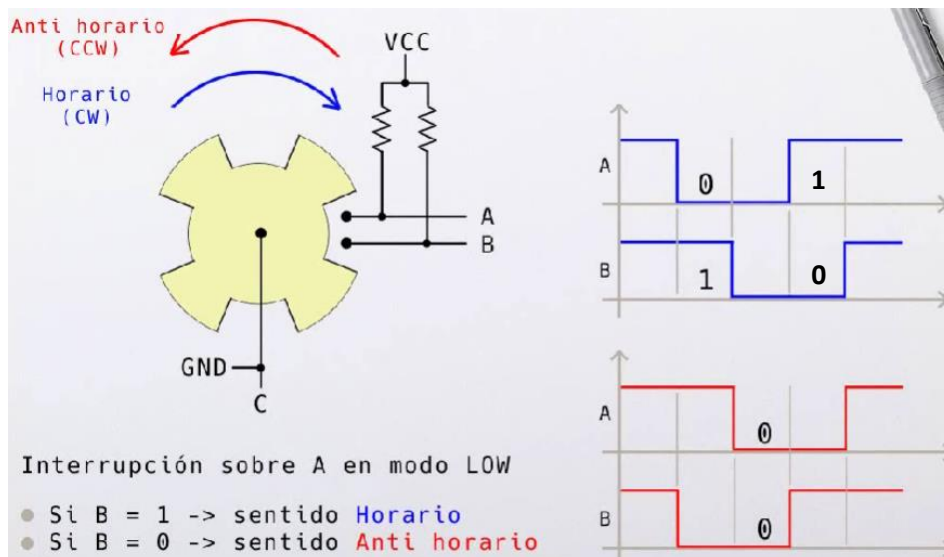
Al terminar ambos pines **A** y **B** en su estado inicial (1 lógico), se concluye **el giro CCW** del encoder y se puede identificar porque **el pin de señal B se convierte de 1 a 0 y viceversa antes que el A**.



Cuando giramos el codificador en **sentido antihorario** el cambio de valor se da primero en el **pin B** y luego en el **pin A**, estas señales digitales ocurren durante el transcurso de **uno de los 30 pasos del encoder** y se representa de la siguiente manera: **(PinA, PinB) = (1, 1) → (1, 0) → (0, 0) → (0, 1)**.

B → A

Esa es la forma de determinar el sentido de giro, **si primero ocurre un flanco de bajada (de 1 a 0 lógico) en el pin A, entonces el giro es en sentido horario (CW)**, pero **si el flanco de bajada ocurre primero en el pin B, el giro es en sentido antihorario (CCW)**.



En conclusión, estas señales digitales creadas por la conmutación de **los pines de señal A y B** con el pin **GND = C** del plato dentado perteneciente al encoder rotativo **pueden ser analizadas en un microcontrolador o FPGA, para así controlar la posición, velocidad o dirección de un dispositivo eléctrico**, como lo puede ser un motor, pantalla LCD, un display de 7 segmentos, etc.

Módulo KY-040

El módulo **KY-040** del codificador rotatorio debe rotar **30 pasos para dar una vuelta de 360°** y cuenta con los siguientes 5 pines de conexión:

- **VCC (+) y GND:** Son los pines de alimentación del encoder que reciben de 3.3 a 5V.
- **A (CLK) y B (DT):** Son los denominados **pines de señal**, cada uno está conectado a una de las 2 resistencias pull up del módulo, conectadas a su vez al pin VCC de la placa.
 - Con las señales digitales generadas en estos dos pines se determina el sentido de giro del eje del encoder y posteriormente se podrá llevar a cabo el control digital de otro dispositivo electrónico o eléctrico.
 - **Giro en sentido horario CW (derecha) del encoder:** Ocurre un **flanco de bajada en el pin A antes que en el B.**
 $(PinA, PinB) = (1, 1) \rightarrow (0, 1) \rightarrow (0, 0) \rightarrow (1, 0)$
 - **Giro en sentido antihorario CCW (izquierda) del encoder:** Ocurre un **flanco de bajada en el pin B antes que en el A.**
 $(PinA, PinB) = (1, 1) \rightarrow (1, 0) \rightarrow (0, 0) \rightarrow (0, 1)$
- **C, Pulsador, Switch o SW:** El pin C está conectado a la placa dentada que está en la base del eje rotatorio y funciona como interruptor cuando se presiona, ya que internamente se encuentra conectado a la tierra común del **pin GND**.

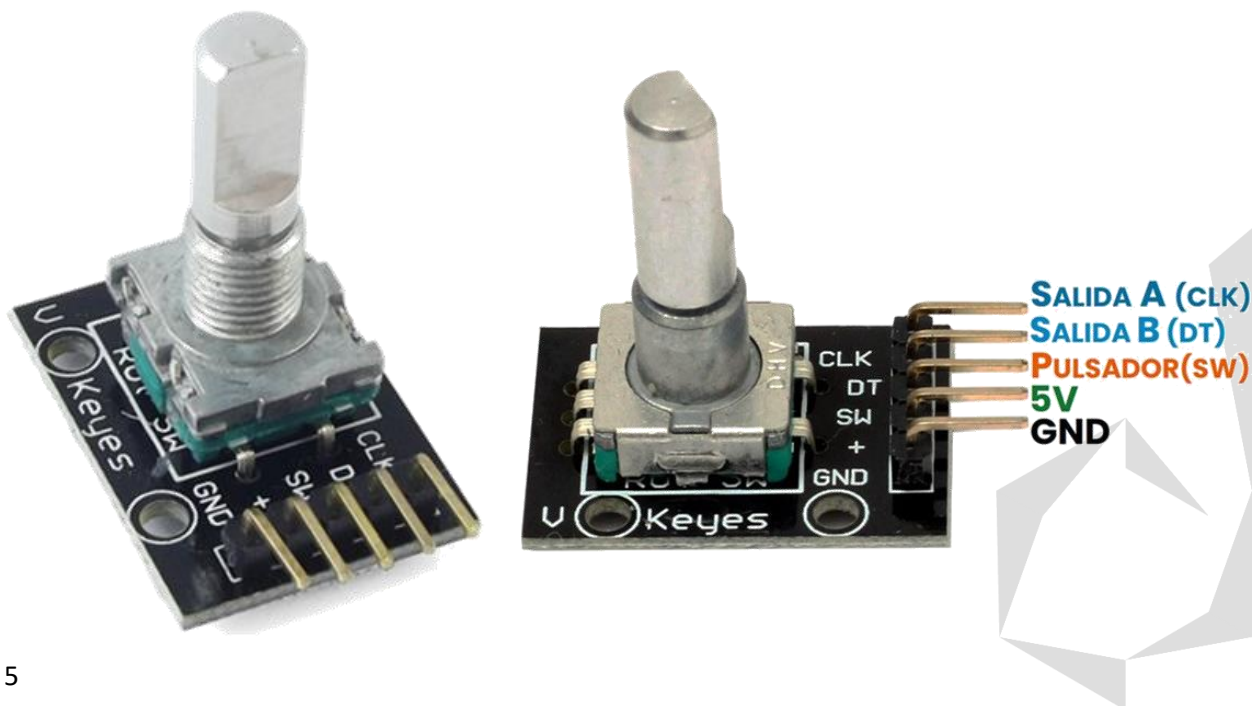
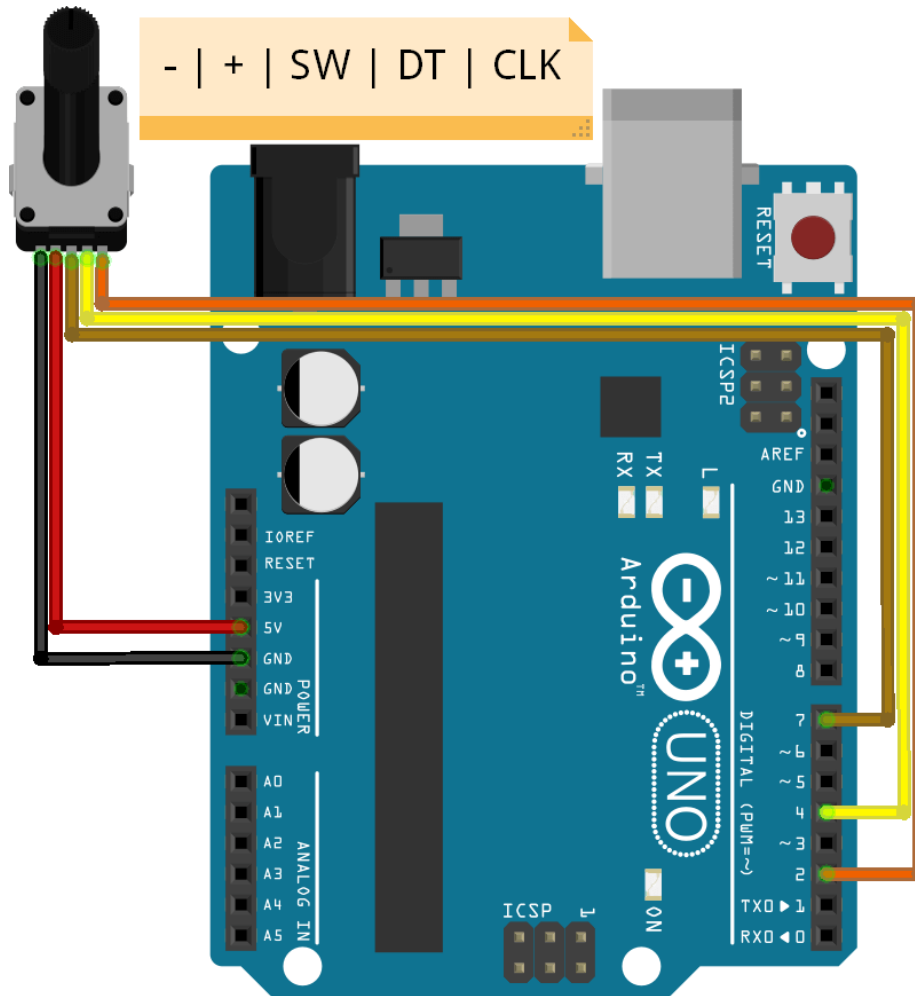


Diagrama de conexión del encoder rotativo KY-040 con la placa de desarrollo Arduino



Siempre que se vaya a efectuar un control con un Encoder Rotativo en Arduino, es necesario conectar alguno de sus pines de señal a uno de los puertos de interrupciones de la placa de desarrollo, para que así no importando que es lo que esté haciendo el código en ese momento, ponga en pausa su instrucción actual y haga caso al control indicado con el encoder.

MODELO	PINES INTERRUPCIONES
Uno, Nano, Mini y otras basadas en 328	2, 3
Mega, Mega2560 y MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo y otras basadas en 32u4	0, 1, 2, 3, 7
Zero	todos los pines digitales excepto el 4
Due	todos los pines digitales

Código Arduino Interrupciones Externas – Encoder Rotativo: Control Digital:

```
/*22.1.-Encoder Rotativo: Este es un dispositivo digital que físicamente se parece a un
potenciómetro, con el cual se pueden controlar otros dispositivos electrónicos o eléctricos,
cuenta con 3 pines para efectuar dicho control: Pin A (CLK), Pin B (DT) y Pin C (SW).
Los pines A y B entregan dos señales digitales con las cuales se puede identificar el sentido
de giro del encoder, que consta de 30 pasos para dar una vuelta de 360°, cabe mencionar que el giro
del encoder no tiene tope, solo se basa en su última posición para saber dónde se encuentra:
- Giro en sentido horario (derecha o CW) del encoder: Ocurre un flanco de bajada en el pin A antes
que en el B. (PinA,PinB) = (1,1)→(0,1)→(0,0)→(1,0)
- Giro en sentido antihorario (izquierda o CCW) del encoder: Ocurre un flanco de bajada en el pin B
antes que en el A. (PinA,PinB) = (1,1)→(1,0)→(0,0)→(0,1)
Y el tercer Pin C es un pulsador, que al ser presionado se conecta a GND, por lo que su valor se
vuelve en 0 lógico.
Se elige que la interrupción se ejecute cuando la señal del PinA esté en modo LOW (0 lógico), porque
al checar el nivel digital del PinB, si este tiene un valor de 1 (2da posición del giro CW = (0,1)),
el giro será hacia la derecha y si tiene un valor de 0 (3era posición del giro CCW = (0,0)), el giro
será hacia la izquierda.*/
//DECLARACIÓN DE VARIABLES Y CONSTANTES:
/*#define: Instrucción utilizada para crear con una directiva una constante global en el programa de
Arduino siguiendo la sintaxis descrita a continuación:
#define nombreConstante valorConstante*/
#define PinA 2 //El PinA se conectan a un pin que active interrupciones externas en el Arduino.
#define PinB 4 //El PinB se puede conectar a un pin digital cualquiera del Arduino.
#define PinC 7 //Pin C conectado al pin digital 3 del Arduino.
/*volatile: Modificador de acceso de alcance global, cuya función es indicar que una variable puede
cambiar su valor en cualquier momento, incluso si no se modifica explícitamente en el código. Se utiliza
en programas de sistemas embebidos, donde las variables pueden ser modificadas por hardware externo como
interrupciones o procesos en paralelo.*/
int ANTERIOR = 50; //En esta variable se almacena el último valor leído de la variable posición.
volatile int POSICION = 50; //Cuando se ejecute la interrupción, se guardará la posición actual.
int POSICION_MAXIMA = 100; //El encoder podrá girar de 0 a 100, empezando en el valor 50.

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL:
void setup() {
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
  - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
  - segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
  INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable.*/
  pinMode(PinA, INPUT); //La constante pinA asignada al pin de interrupción 2 es una entrada.
  pinMode(PinB, INPUT); //La constante pinB asignada al pin digital 4 es una entrada.
  /*Serial.begin(baudRate): Este método inicializa la comunicación serial entre la placa Arduino
  y la computadora, además de que configura su velocidad de transmisión dada en unidad de baudios
  (bit transmitido por segundo) que recibe como su único parámetro:
  - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es
  compatible con la mayoría de los dispositivos y programas.
  - Sin embargo, si se necesita una transferencia de datos más rápida y el hardware/software
  lo admiten, se puede optar por velocidades más altas como 115200 o 57600 baudios.
  Es importante asegurarse de que la velocidad de transmisión especificada coincida con la
  velocidad de comunicación del otro dispositivo al que se conecta el Arduino. Si la velocidad de
  transmisión no coincide, los datos pueden no transmitirse o recibirse correctamente.*/
  Serial.begin(9600); //Comunicación serial de 9600 baudios, para recibir los pasos del encoder.
  /*Serial.println(): Método que escribe un mensaje en la consola serial de Arduino, a la cual se puede
  acceder en la esquina superior derecha donde se encuentra una lupa.*/
  Serial.println("Control Encoder");
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(20);
  /*attachInterrupt(): Método que se define en el setup del código y sirve para indicar el índice
  o pin a donde está conectada la interrupción, la función que se ejecutará cuando la interrupción
  sea detectada y el momento del pulso que acciona la interrupción (flanco de subida, bajada, etc.):
  - primer parámetro: Indica el índice de la interrupción, que no es lo mismo al pin de Arduino
  que será asignado para recibir la interrupción, ya que tienen números distintos, donde por
  ejemplo el pin 2 del Arduino corresponde al índice 0 de la interrupción y el pin 3 al 1, para
  ello se utiliza el siguiente método:
  - digitalPinToInterrupt(pin): Este método no importando el tipo de placa que se esté
  usando transforma el número del pin digital en su equivalente índice de interrupción.
  - segundo parámetro: Indica el nombre de la función ISR (Interrupt Service Routine) que se manda
  a llamar cuando la interrupción sea detectada, indicando la acción que se ejecuta.
  - tercer parámetro: Indica el momento del pulso que activa la interrupción, ya sea:
  - LOW: La interrupción es activada en el nivel bajo del pulso (antes o después de los flancos).
  - RISING: La interrupción es activada en el flanco de subida del pulso.
  - CHANGE: La interrupción es activada cuando ocurra un flanco de bajada o subida en el pulso,
  osea cuando haya un cambio de nivel.
  - FALLING: La interrupción es activada en el flanco de bajada del pulso.
  - HIGH: La interrupción es activada en el nivel alto del pulso (después del flanco de subida).
  Se elige que la interrupción se ejecute cuando la señal del PinA esté en modo LOW (0 lógico), porque
  al checar el nivel digital del PinB, si este tiene un valor de 1 lógico (Giro CW = (PinA,PinB) = (0,1)),
  el giro será hacia la derecha y si tiene un valor de 0 lógico (Giro CCW = (PinA,PinB) = (0,0)), el giro
  será hacia la izquierda.
  Si se usara el PinB para ejecutar la interrupción en vez del PinA, se haría lo mismo, utilizando el modo
  de interrupción LOW, pero detectando el giro CW cuando en PinA haya un 1 y CCW cuando haya un 0.*/
  attachInterrupt(digitalPinToInterrupt(2), POS_ENCODER, LOW);
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO: Interrupciones que identifican el sentido de giro
//del encoder y si su pulsador ha sido presionado.
void loop() {
  /*Serial.println(): Método que escribe un mensaje en la consola serial de Arduino, a la cual se puede
  acceder en la esquina superior derecha donde se encuentra una lupa. Si queremos concatenar esto con
  un valor se debe usar el método String() para convertir su contenido en una cadena de caracteres.*/
  if(POSICION != ANTERIOR){
    Serial.println("Posición = " + String(POSICION));
  }
}
```




```

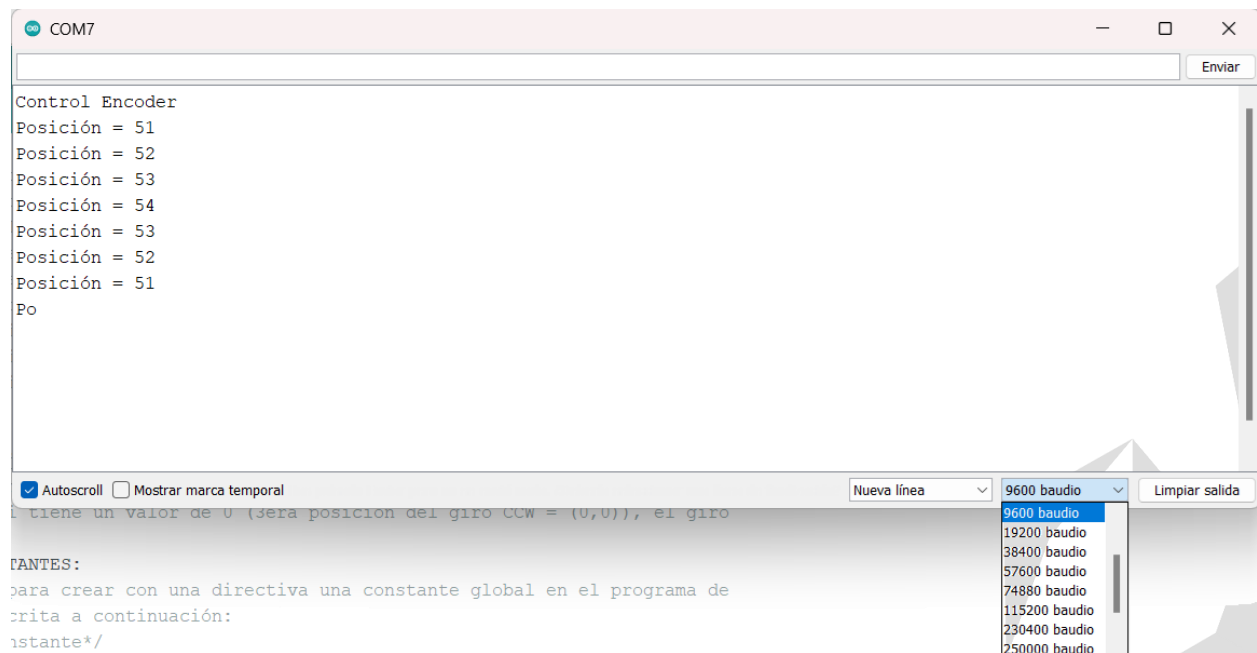
    ANTERIOR = POSICION;
}
}

//FUNCIÓN ISR QUE SE EJECUTA CUANDO SE ACTIVA UNA INTERRUPTIÓN:
/*Se elige que la interrupción se ejecute cuando la señal del PinA esté en modo LOW, porque en ese caso
se checará el nivel digital del PinB y si este tiene un valor de 1, el giro será hacia la derecha (CW),
por lo que se aumentará su posición y si tiene un valor de 0, el giro será hacia la izquierda (CCW), por
lo que se reducirá su posición.
Además, se añadirá un código de antirrebote, ya que como los pasos se realizan de forma mecánica,
conectando los pines A y B con la base dentada de la placa C, se debe dejar pasar cierto tiempo entre
muestreos para no captar lecturas falsas.*/
void POS_ENCODER() {
    //ANTIRREBOTES CON DELAY DE 5 MILLISEGUNDOS: Para ello se debe medir el tiempo entre interrupciones.
    /*static: Modificador de acceso de alcance local y duración de almacenamiento estática, cuya función es
que una variable conserve su valor incluso después de que la función donde fue declarada haya terminado
su ejecución, esto se usa en interrupciones porque cuando este tipo de funciones dejan de correr, se
borran los valores de todas sus variables.
unsigned long: Variable sin signo de 32 bits, puede abarcar números muy grandes de 0 a 4,294,967,295.*/
    static unsigned long ultimaInterrupcion = 0;
    /*millis(): Método que devuelve el tiempo transcurrido en milisegundos, empezando a contar desde que se
enciende la placa Arduino y no deteniéndose hasta que esta se apague o llegue a su límite, que es el
mismo dado por el tipo de dato unsigned long: De 0 a 4,294,967,295 milisegundos = 1,193.0464 horas
49.7102 días = 49 días y 17 horas.
La función se puede utilizar para detener el programa, sustituyendo al método delay(), la razón de ello
es que el método delay() detiene completamente la ejecución del programa, mientras que el método millis()
permite que otros procesos se ejecuten mientras solo una parte del código se pausa.*/
    unsigned long tiempoInterrupcion = millis();

    if(tiempoInterrupcion - ultimaInterrupcion > 5){
        //LECTURA DE LOS PINES DE SEÑAL A Y B:
        /*digitalRead(Pin): Lo que hace este método es leer una entrada digital en un pin en específico que
se indica como su primer parámetro, se puede ver si este valor está en nivel alto con la constante
HIGH o si está en su nivel bajo con la constante LOW.*/
        if(digitalRead(PinB) == HIGH){ //Giro CW
            POSICION++;
        }
        else{ //Giro CCW
            POSICION--;
        }
        /*Actualización del tiempo guardado en la variable ultimaInterrupcion para así ejecutar de nuevo el
código antirrebotes la próxima vez que sea activada la interrupción.*/
        ultimaInterrupcion = tiempoInterrupcion;
        /*min(num1, num2): Método que compara dos valores num1 y num2, devolviendo el más pequeño (mínimo).*/
        /*max(num1, num2): Método que compara dos valores num1 y num2, devolviendo el más grande (máximo).*/
        /*Como el encoder rotativo no tiene un límite mecánico por ninguno de sus dos lados, por medio del
código se fija dicho límite usando los métodos min() y max():
min(limMax, pos): Se utiliza para indicar la posición máxima que puede alcanzar el encoder.
max(limMin, pos): Se utiliza para indicar la posición mínima que puede alcanzar el encoder.*/
        POSICION = min(POSICION_MAXIMA, max(0, POSICION)); //limMin = max(limMin, pos) = 0; limMax = min(limMax, pos) = 100.
    }
}
}

```

Al observar el monitor serie, este debe tener la misma velocidad de transmisión indicada en el método **Serial.begin()**;



Máquina de Estados: Identificación de Giro en Encoder Rotativo

Divisor de Reloj: Módulo DIV

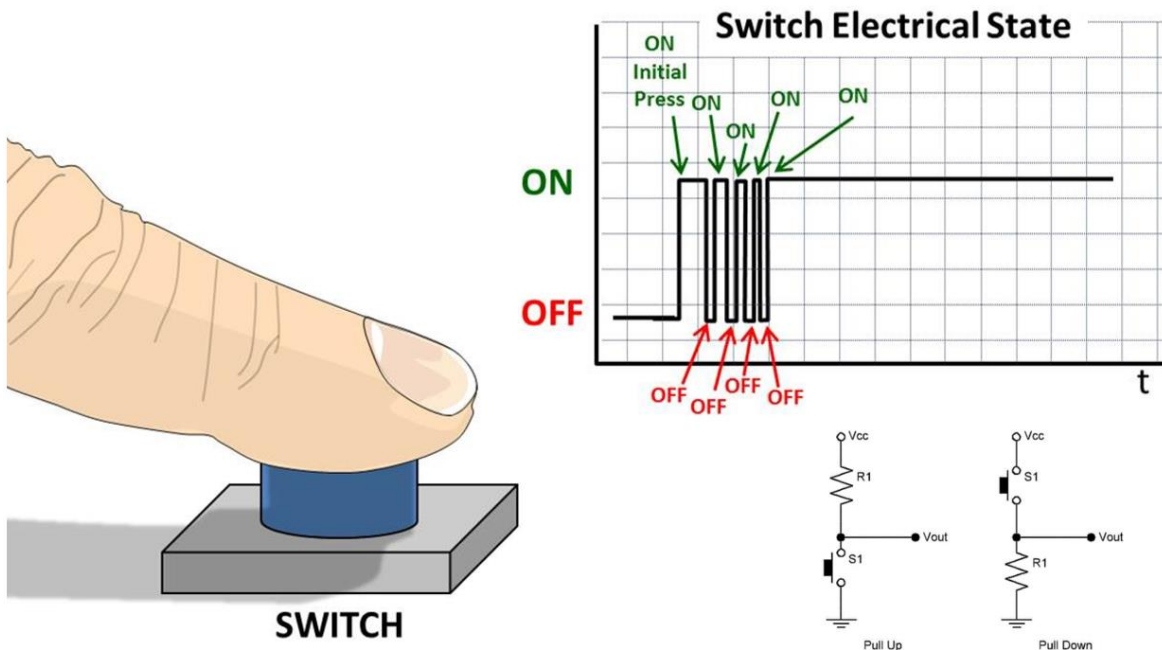
Del divisor de reloj se elegirá la frecuencia de 762.94 Hz, que se encuentra en la coordenada 15 del vector que divide al reloj.

q(i)	BASYS 2 y NEXYS 2		NEXYS 3 y NEXYS 4	
	Frecuencia (Hz)	Periodo (s)	Frecuencia (Hz)	Periodo (s)
i	50,000,000.00	0.00000002	100,000,000.00	0.00000001
0	25,000,000.00	0.00000004	50,000,000.00	0.00000002
1	12,500,000.00	0.00000008	25,000,000.00	0.00000004
2	6,250,000.00	0.00000016	12,500,000.00	0.00000008
3	3,125,000.00	0.00000032	6,250,000.00	0.00000016
4	1,562,500.00	0.00000064	3,125,000.00	0.00000032
5	781,250.00	0.00000128	1,562,500.00	0.00000064
6	390,625.00	0.00000256	781,250.00	0.00000128
7	195,312.50	0.00000512	390,625.00	0.00000256
8	97,656.25	0.00001024	195,312.50	0.00000512
9	48,828.13	0.00002048	97,656.25	0.00001024
10	24,414.06	0.00004096	48,828.13	0.00002048
11	12,207.03	0.00008192	24,414.06	0.00004096
12	6,103.52	0.00016384	12,207.03	0.00008192
13	3,051.76	0.00032768	6,103.52	0.00016384
14	1,525.88	0.00065536	3,051.76	0.00032768
15	762.94	0.00131072	1,525.88	0.00065536
16	381.47	0.00262144	762.94	0.00131072
17	190.73	0.00524288	381.47	0.00262144
18	95.37	0.01048576	190.73	0.00524288
19	47.68	0.02097152	95.37	0.01048576
20	23.84	0.04194304	47.68	0.02097152
21	11.92	0.08388608	23.84	0.04194304
22	5.96	0.16777216	11.92	0.08388608
23	2.98	0.33554432	5.96	0.16777216
24	1.49	0.67108864	2.98	0.33554432

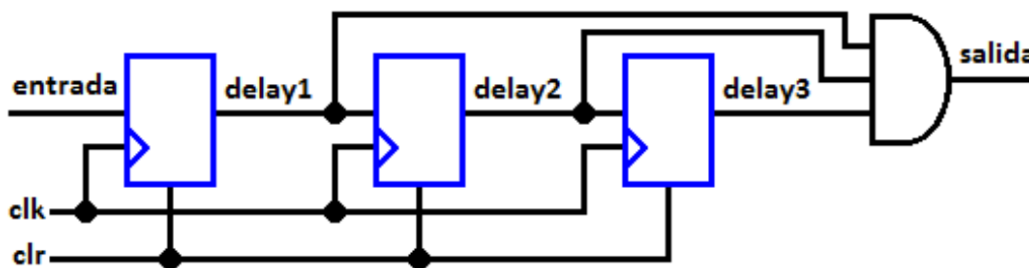
$$f_i = \frac{f}{2^{i+1}}$$

Retardo o Delay Antirrebotes (Delay, Millis y Shift Register)

Los rebotes son falsas pulsaciones que se producen al presionar un botón o interruptor. El código de antirrebotes se utiliza para evitar que sean detectadas las señales no deseadas causadas por los rebotes mecánicos al accionar un switch, donde al accionarse tienen un efecto similar al de cuando se suelta una pelota y esta va rebotando contra el piso hasta que se detiene, el rebote siempre se dará no importando si se coloca una resistencia pull up o pull down para detectar el cambio de tensión.



En un código de Arduino esto se haría a través de un temporizador que detenga la ejecución del programa por unos milisegundos, utilizando el método `millis()` o `delay()`, para así permitir que pasen las falsas pulsaciones, pero en el código de VHDL y Verilog esto se debe realizar a través de un **shift register** que retrase la señal, cabe mencionar que en este código se agrega una entrada llamada **clear** que será **asíncrona**, esto se refiere a que **no se acciona siguiendo el paso de los flancos de subida de la señal de reloj**, sino que se ejecuta en cualquier momento.



El tiempo de retraso en el antirrebotes ocasionado por el **shift register** se calcula de la siguiente manera:

$$t_{delay} = T_{CLK} \left(\#delays - \frac{1}{2} \right) = \frac{1}{f_{CLK}} \left(\#delays - \frac{1}{2} \right)$$

$$f_{CLK} = \frac{1}{T_{CLK}} = \frac{1}{t_{delay}} \left(\#delays - \frac{1}{2} \right)$$

El **Pin A (CLK)** y **Pin B (DT)** se conectarán a los puertos **JA1** y **JA2** de la Nexys 2 y su alimentación podrá ser conectada a los puertos **+JA6 o +JA12** y **-JA5 o -JA11**.

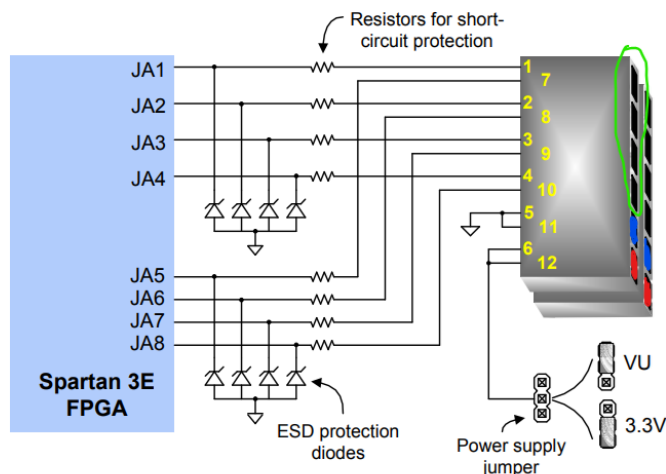


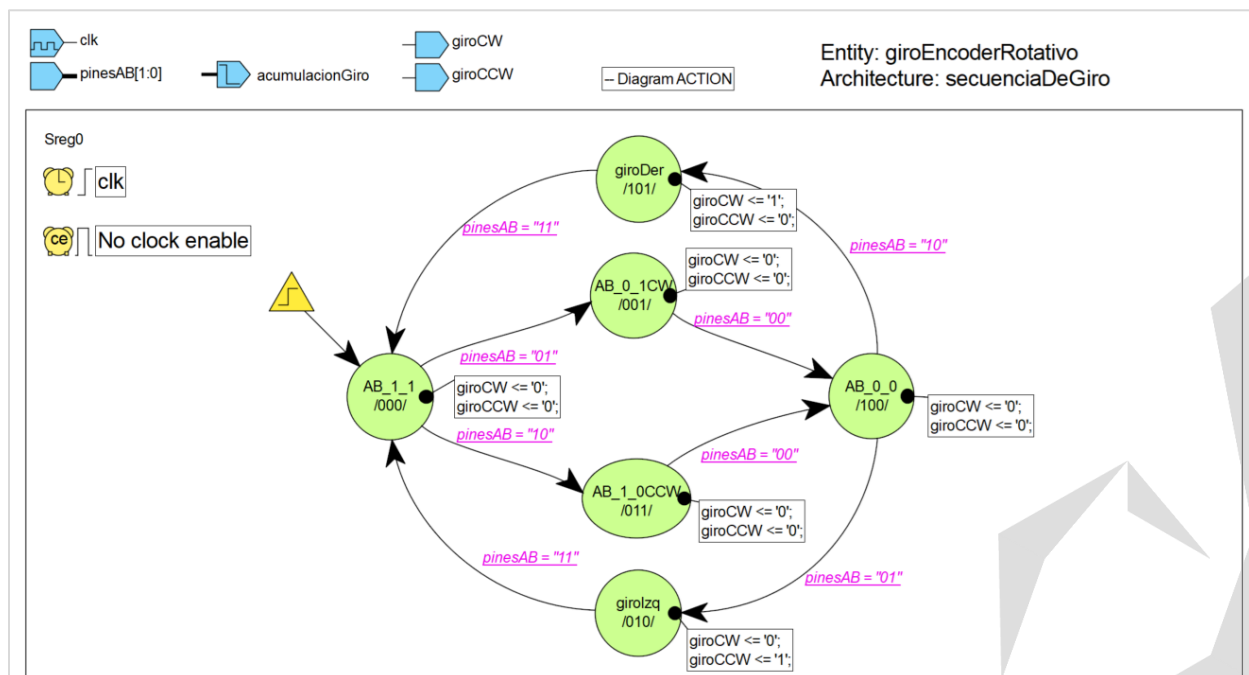
Figure 23: Nexys2 Pmod connector circuits

Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14 ¹
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15 ²
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15 ³
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4: P18	JD10: J14 ⁴

Notes: ¹ shared with LD3 ² shared with LD3 ³ shared with LD3 ⁴ shared with LD3

Código VHDL:

Diagrama de Estados:



Máquina de Estados Tipo Moore:

```
--2.-MAQUINA DE ESTADO FINITO TIPO MOORE: Las FSM (Finite State Machines) están conformadas de
--entradas, transiciones y estados, donde cada estado tiene asignada una o varias salidas. Este tipo
--de programación es excelente para detectar o ejecutar secuencias, todo siguiendo el paso de la
--señal de reloj CLK. En la máquina de Moore sus salidas se generan solo cuando se alcanza un estado
--específico, por lo que su velocidad es más lenta, eso es bueno cuando la secuencia es ingresada
--manualmente, pero no cuando se ingresa por medios digitales, ya que la velocidad de la entrada
--supera la de la detección.
--El estado se refiere a un concepto abstracto que indica como una FSM reacciona automáticamente
--a una secuencia de entradas según un flujo preestablecido. Donde un mismo sistema, si se
--encuentra en diferente estado, podrá reaccionar de forma distinta a una misma entrada.
--En este caso se usará la FSM para detectar el sentido de giro de un encoder rotativo, identificando
--las secuencias dadas por sus pines A y B para saber si el giro es CW (hacia la derecha) o CCW (hacia
--la izquierda).
--Secuencia giro CW (Clock Wise), osea hacia la derecha: (PinA,PinB) = (1,1)(0,1)(0,0)(1,0).
--Secuencia giro CCW (Counter Clock Wise), osea hacia la izquierda: (PinA,PinB) = (1,1)(1,0)(0,0)(0,1).
library IEEE;
use IEEE.std_logic_1164.all;
--Librerías declaradas para poder usar el lenguaje de programación VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Librería declarada para poder hacer operaciones matemáticas sin considerar el signo.
entity giroEncoderRotativo is
    port ( clk : in STD_LOGIC;          --Reloj de 50MHz proporcionado por la NEXYS 2 en el puerto B8.
          pinesAB: in STD_LOGIC_VECTOR (1 downto 0); --Pines para detectar el sentido de giro del encoder.
          giroCW : out STD_LOGIC;      --Led que indica el sentido de giro hacia la derecha del encoder.
          giroCCW : out STD_LOGIC);    --Led que indica el sentido de giro hacia la izquierda del encoder.
end giroEncoderRotativo;

architecture secuenciaDeGiro of giroEncoderRotativo is
    --TYPE: Con esta instrucción se define un tipo de dato que tiene un nombre en específico y clasifica
    --elementos parecidos, más o menos como una clase en POO, a través de la siguiente nomenclatura:
    --type nombreTipoDeDato is (declaracionDeElementos);
    --En VHDL con esta instrucción se declaran los estados de la FSM.
    type Estados is (AB_1_1, AB_0_1CW, giroIzq, AB_0_0CCW, AB_0_0, giroDer);

    --SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la
    --NEXYS 2, solo existe durante la ejecución del código y sirve para poder almacenar algún valor de
    --forma temporal, se debe declarar dentro de la arquitectura y antes de su begin a través de la
    --siguiente nomenclatura:
    --signal nombreVariable : tipoDeDato := valor;
    --En VHDL con esta instrucción se guarda el estadoPresente y estadoFuturo de la máquina de estados,
    --cabe mencionar que dentro del código proporcionado por el diagrama de estados de Aldec, el estado
    --presente es nombrado como Sreg0 y el futuro como NextState_Sreg0.
    signal estadoPresente, estadoFuturo: Estados;

    --Divisor de Reloj: Esta signal sirve para que podamos obtener una gran gama de frecuencias indicadas
    --en la tabla del divisor de reloj, dependiendo de la coordenada que elijamos tomar del vector.
    signal clkdiv: std_logic_vector (25 downto 0); --Declaración del divisor de reloj.
    --Antirrebotes: Una de las aplicaciones del registro de corrimiento, que es resultado de una
    --conexión en serie de varios flip flops (usualmente de Tipo D), donde simplemente se conecta
    --la entrada de un FF tipo D a la salida de otro siguiendo el paso marcado de una misma señal
    --de reloj, es la del código antirrebotes, que sirve para crear un pequeño retraso de tiempo en
    --el programa, que deje pasar las falsas pulsaciones que se producen al presionar un botón.
    --El retraso en tiempo del antirrebotes estará sujeto a la frecuencia de la señal de reloj, ya que
    --cada que se perciba un flanco de subida, con el diferente número de delays, se estará esperando el
    --tiempo calculado con la siguiente ecuación:
    --t_delay = periodoReloj * (#signals_delay - (1/2))
    --t_delay = 1/frecuenciaReloj * (#signals_delay - (1/2))
    --frecuenciaReloj = 1/t_delay * (#signals_delay - (1/2))
    signal delay_1 : STD_LOGIC_VECTOR (1 downto 0);
    signal delay_2 : STD_LOGIC_VECTOR (1 downto 0);
    signal pinesDebounce : STD_LOGIC_VECTOR (1 downto 0); --Entrada ya con el delay antirrebotes añadido.
    --Los type y signal se declaran dentro de la arquitectura, pero antes de su begin.
begin
    --Primero que nada, se añade el proceso del divisor de reloj, este no viene incluido en el código
    --de Aldec. Se puede nombrar los process de la siguiente manera:
    --nombreProcess : process(entradas y signals separadas por comas) begin
    divisorDeReloj: process(clk, clkdiv) begin
        --La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco
        --de subida en la señal de reloj clk proveniente de la NEXYS 2, se elige distintas frecuencias
        --para ver cuál es la que mejor funciona para detectar las entradas de la máquina de estados.
        if(rising_edge(clk)) then
            clkdiv <= clkdiv + '1'; --Esto crea al divisor de reloj.
        end if;
    end process;

    --Luego se añade el proceso del antirrebotes, este tampoco viene incluido en el código de Aldec.
    antirrebotesBoton: process(clkdiv(17), pinesAB, delay_1, delay_2) begin
        --La instrucción rising_edge() hace que este condicional solo se ejecute cuando ocurra un flanco
        --de subida en la señal del divisor de reloj clkdiv, en este caso esa acción es que se trasladen
        --los datos de la entrada a la salida del registro, pero como pasa a través de varias signal, esto
        --es lo que ocasionara el retraso en tiempo, que genera el delay del antirrebotes, para obtener un
        --delay de antirrebotes de 8 microsegundos, que es lo suficientemente rápido para identificar la
        --secuencia introducida por el encoder rotativo; para ello la frecuencia de la señal de reloj
        --debe ser de 187.5 kHz.
        --frecuenciaReloj = 1/t_delay * (#signals_delay - (1/2)) = 1/8e-6 * (2 - (1/2)) = 187,500 Hz.
        --La que mas se le acerca es la posición 7 del divisor de reloj cuya frecuencia es f = 195,312.5 Hz.
        if(rising_edge(clkdiv(17))) then
            delay_1 <= pinesAB;
            delay_2 <= delay_1;
        end if;
        pinesDebounce <= delay_1 and delay_2;
    end process;
end architecture;
```

```

--Las partes necesarias para implementar el código de una FSM son las siguientes:
--Actualización del estado presente y futuro: En esta parte del código que se ejecuta cada que se
--perciba un flanco de subida en la señal de reloj, continuamente se asigna el valor del estado
--futuro al estado presente, para de esa forma actualizarlo.
--La frecuencia ideal para identificar la secuencia en una FSM tipo Moore está en la coordenada
--15 = 762.94Hz = 1.3ms.
--Esta parte a veces viene incluida hasta abajo del código proporcionado por Aldec, pero debe ser
--modificado para funcionar a través de la señal del divisor de reloj.
actualizacionDeEstado: process(clkdiv(15)) begin
    if rising_edge(clkdiv(15)) then
        estadoPresente <= estadoFuturo;
    end if;
end process;

--Lógica de la máquina de estados: A través de esta parte se lleva a cabo la función de la
--FSM, donde dependiendo de su tipo, ya sea Mealy o Moore, sus salidas se asignan a las transiciones
--de un estado a otro o al estado en sí. Cabe mencionar que siempre que se cree una nueva FSM, esta
--es la parte que deberá ser reemplazada en el código, lo demás se puede quedar igual y funcionaria de
--maravilla, pero hay que tener en cuenta cambiar el nombre de la entrada por la señal del antirrebotes
--y cambiar el nombre de Sreg0 por estadoPresente y nextState_Sreg0 por la señal estadoFuturo.
--Además, deberemos cambiar el nombre de los nuevos estados en la parte de arriba, en los type.
--Aquí a través de condicionales if y case se declaran los estados, entradas y transiciones.
maquinaDeEstados: process(pinesDebounce, estadoPresente) begin
    --Condiciones iniciales de la máquina de estados:
    estadoFuturo <= estadoPresente;
    --Secuencia de giro CW, hacia la derecha: (PinA,PinB) = (1,1),(0,1),(0,0),(1,0).
    --Secuencia de giro CCW, hacia la izquierda: (PinA,PinB) = (1,1),(1,0),(0,0),(0,1).
    giroCW <= '0'; --Giro derecha = 0.
    giroCCW <= '0'; --Giro izquierda = 0.
    --FSM tipo Moore: LAS SALIDAS se declaran en LOS ESTADOS que describen los case.
    case(estadoPresente) is
        when AB_1_1 => --Estado: AB_1_1 = (1,1).
            --SALIDAS DEL ESTADO AB_1_1:
            giroCW <= '0';
            giroCCW <= '0';
            --TRANSICIONES DEL ESTADO AB_1_1:
            case pinesDebounce is
                when "01" => --Transición al siguiente paso del giro hacia la derecha.
                    estadoFuturo <= AB_0_1CW;
                when "10" => --Transición al siguiente paso del giro hacia la izquierda.
                    estadoFuturo <= AB_1_0CCW;
                when others =>
                    null;
            end case;
        when AB_0_1CW => --Estado: AB_0_1CW = (1,1),(0,1) = Giro CW.
            --SALIDAS DEL ESTADO AB_0_1CW:
            giroCW <= '0';
            giroCCW <= '0';
            case pinesDebounce is
                when "00" => --Transición al siguiente paso del giro hacia la derecha.
                    estadoFuturo <= AB_0_0;
                when others =>
                    null;
            end case;
        when giroIzq => --Estado: giroIzq = (1,1),(1,0),(0,0),(0,1) = Giro CCW.
            --SALIDAS DEL ESTADO giroIzq:
            giroCW <= '0';
            giroCCW <= '1';
            case pinesDebounce is
                when "11" => --Transición al reinicio de la secuencia de ambos giros.
                    estadoFuturo <= AB_1_1;
                when others =>
                    null;
            end case;
        when AB_1_0CCW => --Estado: AB_1_0CCW = (1,1),(1,0) = Giro CCW.
            --SALIDAS DEL ESTADO AB_1_0CCW:
            giroCW <= '0';
            giroCCW <= '0';
            case pinesDebounce is
                when "00" => --Transición hacia al sig. paso del giro hacia la derecha o izq.
                    estadoFuturo <= AB_0_0;
                when others =>
                    null;
            end case;
        when AB_0_0 => --Estado: AB_0_0 con giro CCW = (1,1),(1,0),(0,0) o giro CW = (1,1),(0,1),(0,0).
            --SALIDAS DEL ESTADO AB_0_0:
            giroCW <= '0';
            giroCCW <= '0';
            case pinesDebounce is
                when "10" => --Transición que termina la secuencia del giro CW (der).
                    estadoFuturo <= giroDer;
                when "01" => --Transición que termina la secuencia del giro CCW (izq).
                    estadoFuturo <= giroIzq;
                when others =>
                    null;
            end case;
        when giroDer => --Estado: giroDer = (1,1),(0,1),(0,0),(1,0) = Giro CW.
            --SALIDAS DEL ESTADO giroDer:
            giroCW <= '1';
            giroCCW <= '0';
            case pinesDebounce is
                when "11" => --Transición al reinicio de la secuencia de ambos giros.
                    estadoFuturo <= AB_1_1;
                when others =>
                    null;
            end case;
    end case;
end process;

```

```

                                end case;
                                when others =>
                                    null;
                                end case;
                            end process;
                            --Fin de la lógica de la máquina de estados.
                        end secuenciadeGiro;

```

Código UCF:

```

//ENTRADAS Y SALIDAS de la Maquina de Estados
//ENTRADAS:
net "clk" loc = "B8";           //Entrada clk asignada al reloj en el puerto B8.
net "pinesAB[1]" loc = "K12";   //Pin A del encoder asignado al puerto JA2 en el puerto K12.
net "pinesAB[0]" loc = "L15";   //Pin B del encoder asignado al puerto JA1 en el puerto L15.

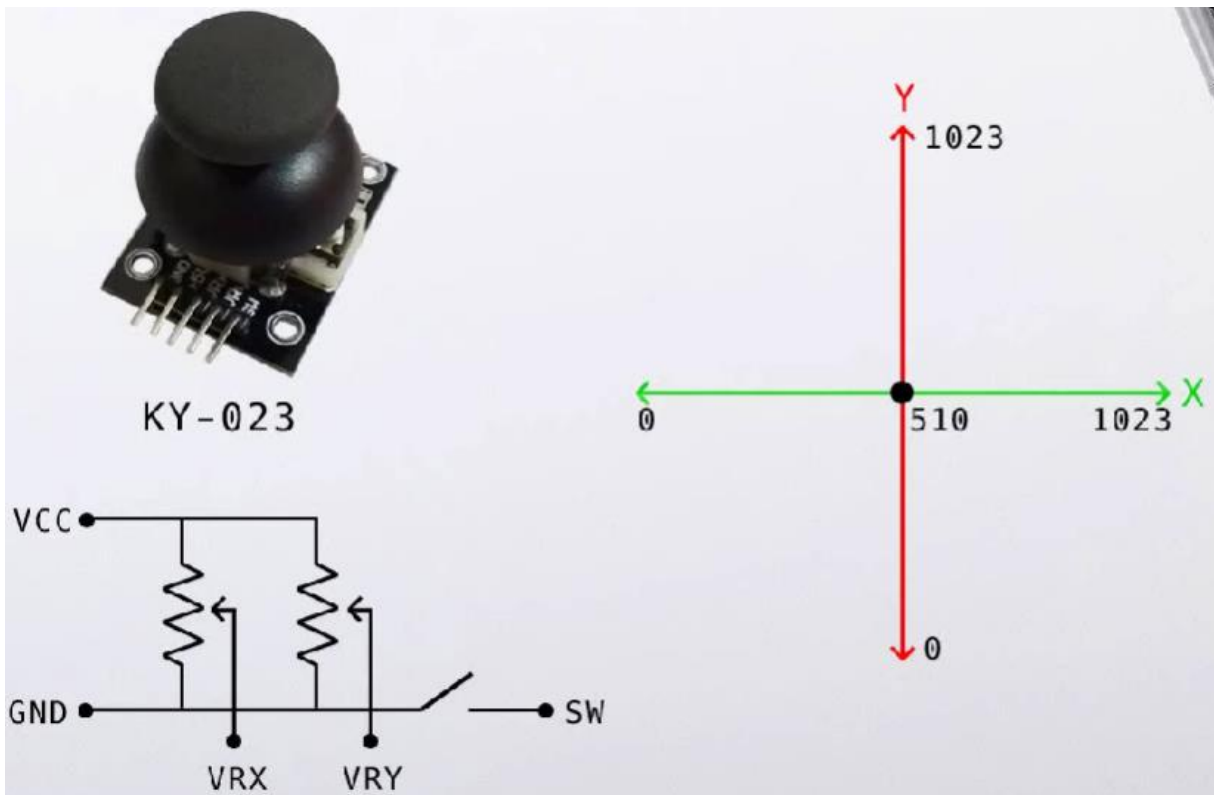
//SALIDAS:
net "giroCW" loc = "J14";       //LD0 que se enciende al identificar el giro del encoder en sentido CW.
net "giroCCW" loc = "R4";       //LD7 que se enciende al identificar el giro del encoder en sentido CCW.

```

Joystick: Control Analógico

El joystick es un dispositivo de control analógico que **convierte el movimiento mecánico en una señal analógica que se mueve en todo el plano XY, causada por una resistencia variable**, la cual puede ser leída por un sistema de control que tenga pines de entrada analógicos, como un microcontrolador, CPU, etc.

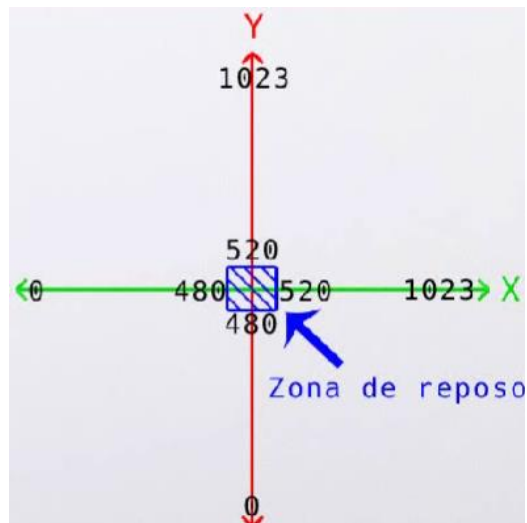
El control tiene una palanca que se mueve a través de todo su eje horizontal (**VRX**) y vertical (**VRY**). En su posición inicial se encuentra justo en el valor intermedio de sus dos rangos de resistencia, donde, **después de haber pasado por el ADC del Arduino, puede adoptar valores de entre 0 y 1023**, por lo que inicialmente se encuentra en la coordenada: $Posicion_{inicial} = (X, Y) = \left(\frac{1023}{2}, \frac{1023}{2}\right) = (511.5, 511.5)$.



Aunque idealmente la coordenada inicial se encuentra en el punto: $Posicion_{inicial} = (511.5, 511.5)$, este no siempre es el caso, por lo cual se declarará un **cuadrado que considere el posible error, llamado zona de reposo**; a partir de este punto la palanca del control analógico podrá ser movida a cualquier posición del plano XY, y al hacerlo adoptará los siguientes valores de tensión de **0 a 5V**, osea de **0 a 1023**:

- **Si la palanca se mueve hacia la izquierda en el eje horizontal (coordenadas -X)**: La señal analógica del Joystick **reduce su tensión**, adoptando valores de **0 a 480**.
- **Si la palanca se mueve hacia la derecha en el eje horizontal (coordenadas +X)**: La señal analógica del Joystick **aumenta su tensión**, adoptando valores de **520 a 1023**.
- **Si la palanca se mueve hacia abajo en el eje horizontal (coordenadas -Y)**: La señal analógica del Joystick **reduce su tensión**, adoptando valores de **0 a 480**.
- **Si la palanca se mueve hacia arriba en el eje horizontal (coordenadas +Y)**: La señal analógica del Joystick **aumenta su tensión**, adoptando valores de **520 a 1023**.

Cuando se suelte la palanca, está por sí sola regresará a su punto inicial. También cabe mencionar que, **si esta es presionada, se cerrará el interruptor NA (Normalmente Abierto) del pin SW**, adoptando el valor de **0 lógico**. Adicionalmente, la **zona de reposo** recomendada es de **480 a 520** en la zona central.



En conclusión, las señales analógicas creadas por el movimiento de la palanca en los pines **VRX y VRY** sobre todo el plano XY o el interruptor de la palanca del Joystick que puede ser presionado y está unido al pin **GND = SW** podrán ser interpretados, para así **controlar la posición, velocidad o dirección de un dispositivo eléctrico**, como lo puede ser un motor, el letrero que se muestra en una pantalla LCD, un display de 7 segmentos, etc.

Módulo KY-023

El módulo **KY-023** del joystick cuenta con los siguientes 5 pines de conexión:

- **VCC (+5V) y GND**: Son los pines de alimentación del encoder que reciben de 3.3 a 5V.
- **VRX y VRY**: Se encuentran conectados a las direcciones horizontal (**VRX**) y vertical (**VRY**) de la palanca perteneciente al joystick, generando a través de sus resistencias variables dos señales analógicas que **después de haber pasado por el ADC del Arduino, pueden adoptar valores de 0 a 1023**, abarcando completamente el plano XY.

- **SW:** El pin SW es un interruptor normalmente abierto, que al ser presionado se conecta al pin **GND** y adopta el valor de 0 lógico, para ello este pin siempre se debe conectar a una resistencia pull-up de 10kΩ, osea que esté conectada entre el pin y la alimentación **VCC**.

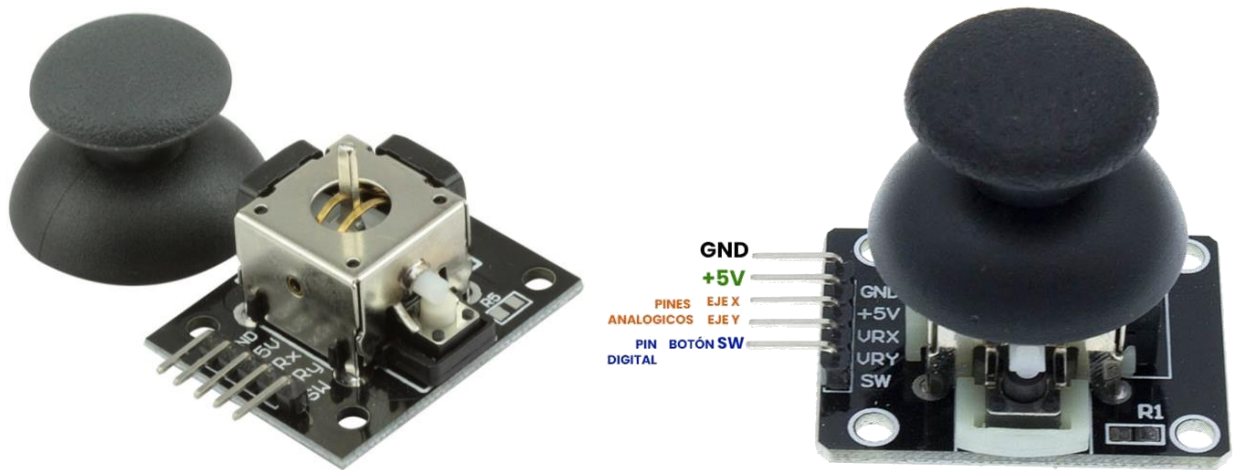
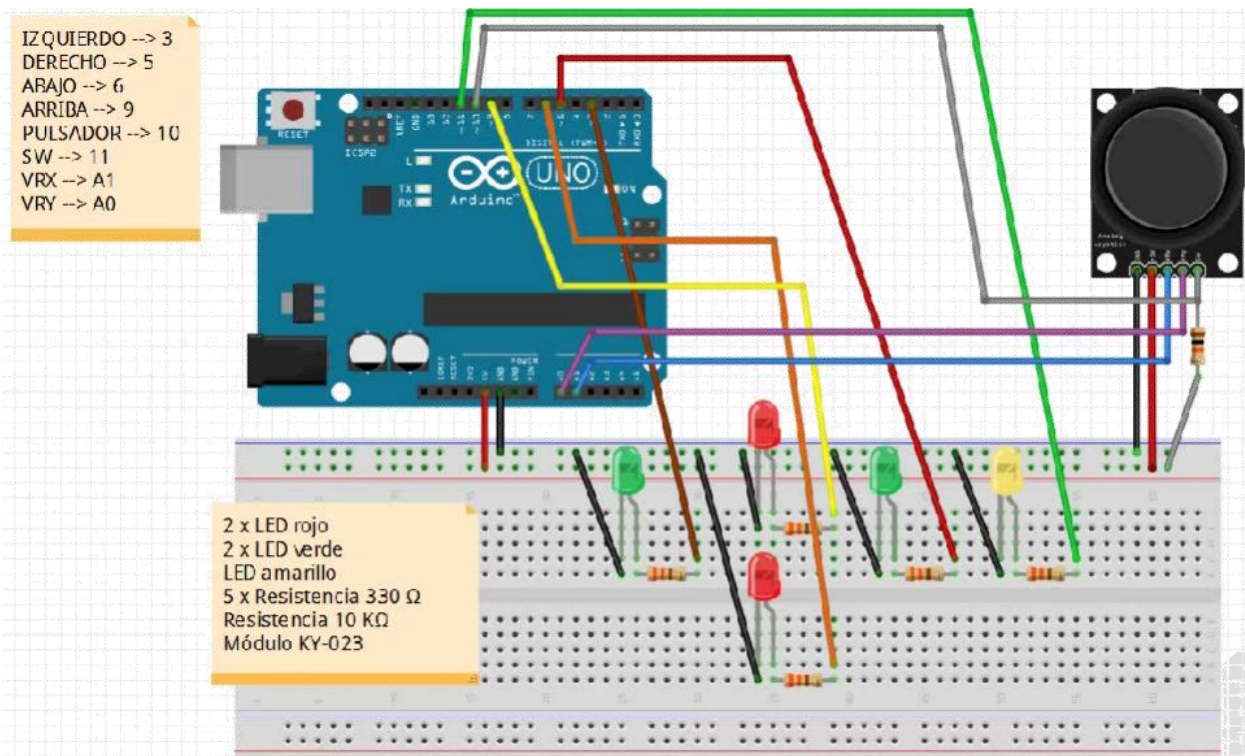


Diagrama de conexión del encoder rotativo KY-040 con la placa de desarrollo Arduino



Código Arduino – Joystick: Control Analógico:

```
/*22.2.-Joystick: Este es un dispositivo analógico con el cual se pueden controlar otros dispositivos electrónicos o eléctricos, cuenta con 3 pines para efectuar dicho control: VRX, BRY y SW. Los pines VRX y VRY entregan dos señales analógicas que cambian su valor dependiendo del punto XY donde se encuentre su eje, tan sencillo como eso.*/
//DECLARACIÓN DE VARIABLES Y CONSTANTES:
#define VRX A1 //Pin VRX que da las coordenadas horizontales X del Joystick.
#define VRY A0 //Pin VRY que da las coordenadas verticales Y del Joystick.
#define SW 11 //Pin SW del interruptor normalmente abierto del Arduino.
```

```

int X; //Coordenada X que adopta valores de 0 a 1023 por el ADC de 10 bits.
int Y; //Coordenada Y que adopta valores de 0 a 1023 por el ADC de 10 bits.

int CoordenadaX; //Coordenada X después de la transformación de 0 a 1023 -> 0 a 5V.
int CoordenadaY; //Coordenada Y después de la transformación de 0 a 1023 -> 0 a 5V.

int intervaloMuestreo = 50; //Intervalo de muestreo dado en milisegundos.

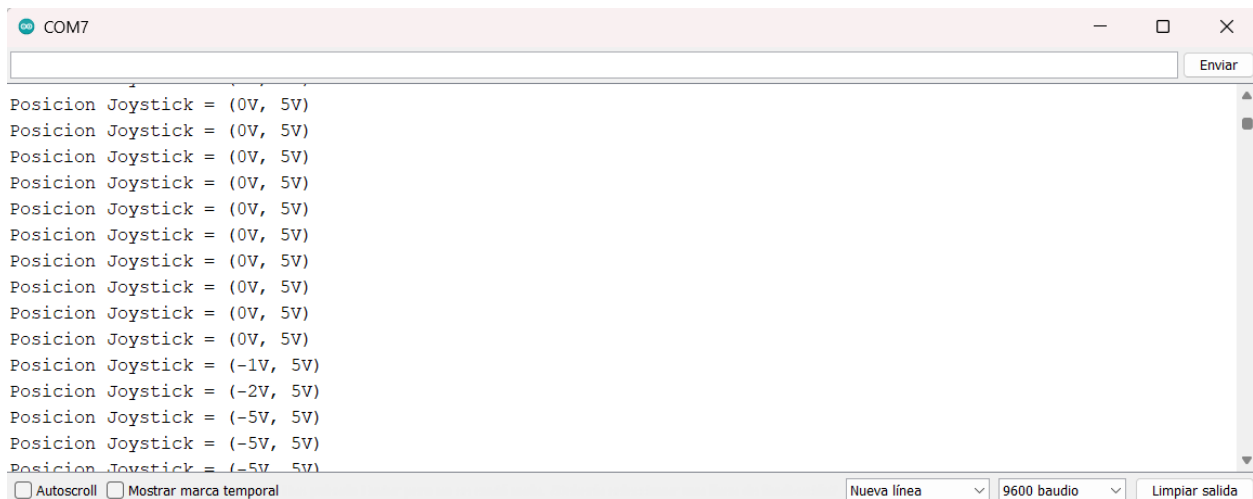
//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL:
void setup() {
  /*pinMode(): Método que indica cuales pines del Arduino son entradas y cuales son salidas:
   - primer parámetro: Indica el pin de Arduino que será asignado como salida o entrada.
   - segundo parámetro: Usa la instrucción OUTPUT para indicar que el pin es una salida o
   INPUT para indicar que el pin es una entrada.
  El número del pin que recibe este método como primer parámetro se puede declarar directamente
  como un número o se puede declarar al inicio del programa como una variable.*/
  pinMode(VRX, INPUT); //Constante VRX asignada al pin A1.
  pinMode(VRY, INPUT); //Constante VRY asignada al pin A0.
  pinMode(SW, INPUT); //Constante SW asignada al pin digital 11.
  /*Serial.begin(baudRate): Este método inicializa la comunicación serial entre la placa Arduino
  y la computadora, además de que configura su velocidad de transmisión dada en unidad de baudios
  (bit transmitido por segundo) que recibe como su único parámetro:
   - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es
   compatible con la mayoría de los dispositivos y programas.
   - Sin embargo, si se necesita una transferencia de datos más rápida y el hardware/software
   lo admiten, se puede optar por velocidades más altas como 115200 o 57600 baudios.
  Es importante asegurarse de que la velocidad de transmisión especificada coincida con la
  velocidad de comunicación del otro dispositivo al que se conecta el Arduino. Si la velocidad de
  transmisión no coincide, los datos pueden no transmitirse o recibirse correctamente.*/
  Serial.begin(9600); //Comunicación serial de 9600 baudios, para recibir los pasos del encoder.
  /*Serial.println(): Método que escribe un mensaje en la consola serial de Arduino, a la cual se puede
  acceder en la esquina superior derecha donde se encuentra una lupa.*/
  Serial.println("Control Joystick");
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO:
void loop() {
  /*analogRead(): Método que lee un valor analógico por medio del ADC del Arduino que consta de 10
  bits, por lo que convertirá los valores de tensión de 0 a 5V que reciba a un equivalente de 0 a
  2^10 - 1 = 1023.*/
  X = analogRead(VRX);
  Y = analogRead(VRY);
  /*digitalRead(Pin): Lo que hace este método es leer una entrada digital en un pin en específico que
  se indica como su primer parámetro, se puede ver si este valor está en nivel alto con la constante
  HIGH o si está en su nivel bajo con la constante LOW.*/
  int JoystickButton = digitalRead(SW);

  /*La zona de reposo es un cuadrado que se encuentra alrededor del punto inicial del joystick, que
  como abarca valores de 0 a 5V, osea de 0 a 1023 después de ser procesador por el ADC de 10 bits del
  Arduino, idealmente se debería encontrar en la coordenada:
  Posicion_inicial = (X,Y) = (1023/2,1023/2) = (511.5,511.5)
  Pero por seguridad se declara la zona de reposo que considera el error que se podría tener, abarcando
  posiciones de 480 a 520.*/
  if(X >= 0 && X < 480){ //Obtención de coordenadas -X.
    /*map(): Método que realiza una regla de 3, convirtiendo un número de un rango a otro que sea
    equivalente al primero, en sus parámetros recibe lo siguiente:
     - Primer parámetro: Recibe la variable de donde proviene el número a convertir.
     - Segundo parámetro: Recibe el valor mínimo del primer rango de valores.
     - Tercer parámetro: Recibe el valor máximo del primer rango de valores.
     - Cuarto parámetro: Recibe el valor mínimo del segundo rango de valores, al que queremos llegar.
     - Quinto parámetro: Recibe el valor máximo del segundo rango de valores, al que queremos llegar.
    De esta manera se convierten los valores de de 0 a 480 entregados por el método analogRead() y la zona
    de reposo de la dirección -X a valores de tensión de -5 a 0V.*/
    CoordenadaX = map(X, 0, 480, -5, 0);
  }else if(X >= 520 && X <= 1023){ //Obtención de coordenadas +X.
    CoordenadaX = map(X, 520, 1023, 0, 5);
  }else{
    CoordenadaX = 0;
  }
  if(Y >= 0 && Y < 480){ //Obtención de coordenadas -Y.
    CoordenadaY = map(Y, 0, 480, -5, 0);
  }else if(Y >= 520 && Y <= 1023){ //Obtención de coordenadas +Y.
    CoordenadaY = map(Y, 520, 1023, 0, 5);
  }else{
    CoordenadaY = 0;
  }
  /*Serial.println(): Método que escribe un mensaje en la consola serial de Arduino, a la cual se puede
  acceder en la esquina superior derecha donde se encuentra una lupa. Si queremos concatenar esto con
  un valor se debe usar el método String() para convertir su contenido en una cadena de caracteres.*/
  Serial.println("Posicion Joystick = " + String(CoordenadaX) + "V, " + String(CoordenadaY) + "V");
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(intervaloMuestreo);
}

```





```
COM7
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (0V, 5V)
Posicion Joystick = (-1V, 5V)
Posicion Joystick = (-2V, 5V)
Posicion Joystick = (-5V, 5V)
Posicion Joystick = (-5V, 5V)
Posicion Joystick = (-5V, 5V)
Autoscroll  Mostrar marca temporal  Nueva línea  9600 baudio  Limpiar salida
```

Para poder utilizar el Joystick con un FPGA se necesitaría utilizar un ADC externo, ya que este solo recibe y trabaja con señales digitales.

Referencias

Bitwise Ar, “Arduino desde cero en Español - Capítulo 21 - Interrupciones externas 🖐️ (pruebas con KY-010)”, 2017 [Online], Available: https://www.youtube.com/watch?v=up-goNfJ_EY

Bitwise Ar, “Arduino desde cero en Español - Capítulo 22 - Codificador rotatorio KY-040 (rotary encoder)”, 2017 [Online], Available: <https://www.youtube.com/watch?v=fnDHQ6YcxTs>

Bitwise Ar, “Arduino desde cero en Español - Capítulo 17 - Módulo Joystick analógico KY-023 🖱️”, 2017 [Online], Available: <https://www.youtube.com/watch?v=okvUaG2BRBo>

