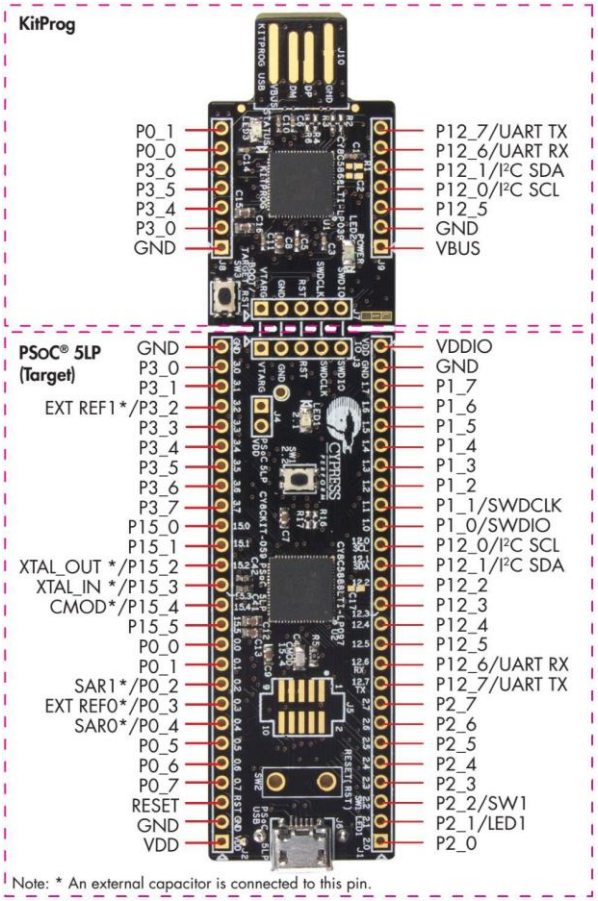
Prácticas de Laboratorio

Diseño de Sistemas Electrónicos



José Manuel Herrera Vera

Robin Costas del Moral

Miguel Ángel Rispal Martínez

José Antonio Padial Molina

ÍNDICE

Práctica 1.3. Iluminación de un LED mediante señales PWM

La práctica consiste en implementar un sistema que haga parpadear 2 leds.

- Uno por la propia señal PWM.
- Otro por una interrupción excitada por la señal del PWM, además este último debe cambiar su frecuencia de parpadeo en función de un botón.

Nosotros para cambiar la frecuencia hemos usado una variable contador para que ciertas iteraciones no se ejecute la función toggle_LED2() y a sí conseguimos regular la frecuencia de parpadeo.

Código:

Cabeceras y funciones

```
#include "project.h"

uint8 intep;
int cont_int = 0;
int flag_5hz = 1;
int flag_05hz = 0;

void toggle_LED1(){
   LED1_Write(LED1_Read() ^ 1U);
}

void toggle_LED2(){
   LED2_Write(LED2_Read() ^ 1U);
}
CY_ISR(isr_1_Interrupt){
      intep=1;
}
```

Main

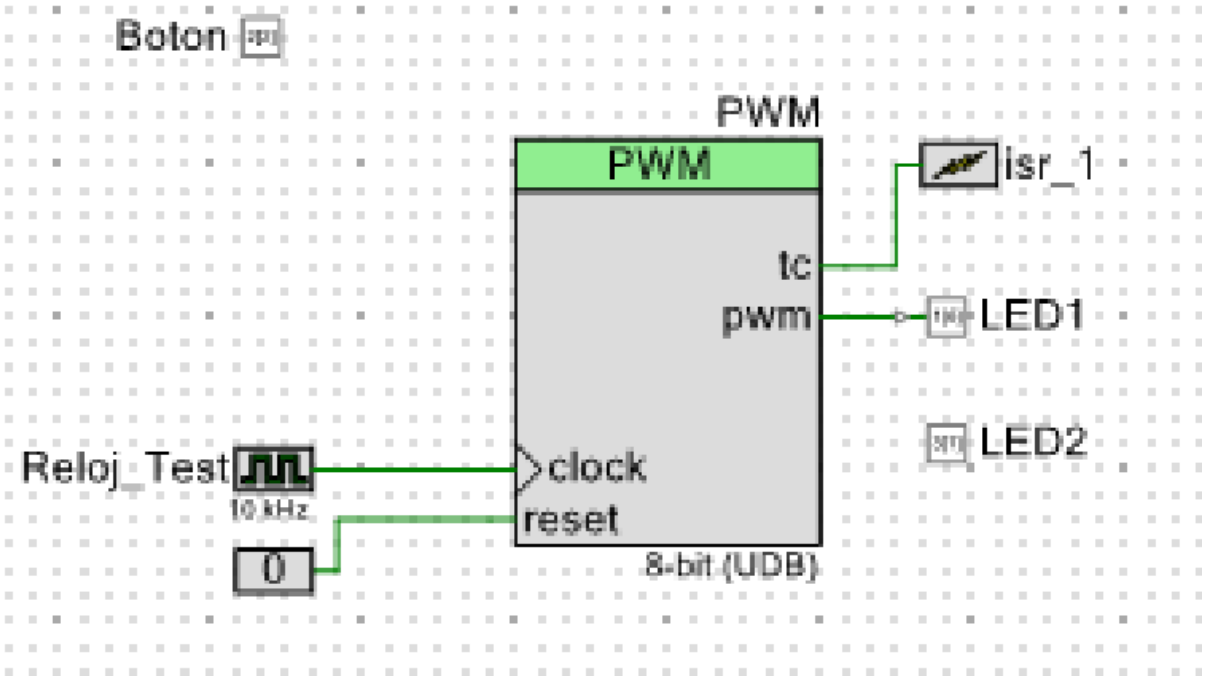
```
int main(void)
{
   isr_1_Start();
   isr_1_SetVector(&isr_1_Interrupt);

   PWM_Start();

   CyGlobalIntEnable;

   intep=0;

   for(;;)
   {
      if(intep==1 && flag_5hz==1){
         toggle_LED2();
         intep=0;
      }
      else if(intep == 1 && flag_05hz ==1){
         cont_int++;
         if(cont_int == 5){
            toggle_LED2();
            cont_int=0;
         }
         intep=0;
      }

      if(!Boton_Read()){
         flag_05hz = 1;
         flag_5hz = 0;
      }
      else{
         flag_05hz = 0;
         flag_5hz = 1;
      }
   }
}
```

Las funciones toggle_LED1() y toggle_LED2() básicamente cambian el estado de un led, es decir, si está apagado lo enciende y si está encendido lo apaga.

CY_ISR(isr_1_Interrupt) es la función propia de la interrupción (isr_1) y lo único que hace es que pone la variable intep a 1.

Main: Primero se inicializan la interrupción y el PWM, se habilitan las interrupciones y se pone intep a 0. Dentro del bucle for(;;), se comprueban los valores de intep y los flags, dependiendo de estos valores y de una variable contador podemos regular la frecuencia con la que se ejecuta la función toggle_LED2(). Al final del bucle comprobamos si está pulsado o no el botón con el cual cambiamos los valores de los flags.

Top Design y explicación:

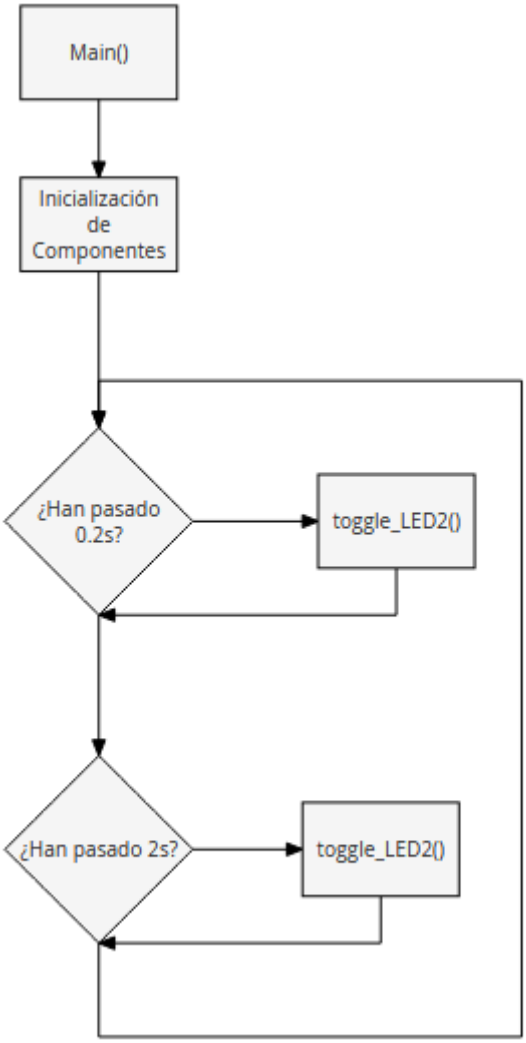


En la parte superior izquierda encontramos el botón, el cual en su configuración tiene puesto pull-up ya que es el botón de la PSOC.

Podemos ver un PWM como parte principal, el cual tiene conectado como entrada un reloj de 10KHz y un reset que siempre está a 0, y como salida tiene los propios pulsos del PWM conectados al LED1 y la salida tc (terminal count, da señal cuando el PWM ha llegado a su último valor) conectada a la interrupción isr_1.

Al lado del PWM podemos ver el LED2 suelto, este led no se conecta ya que funciona a través de la interrupción.

Diagrama de flujo:


Main()
Inicialización de Componentes
¿Han pasado 0.2s?
toggle_LED2()
¿Han pasado 2s?
toggle_LED2()

Práctica 2.1. Manejo del LCD
Práctica 2.2. Puertas Lógicas en el PSoC

La práctica 2.1 consiste en familiarizarse con la pantalla LCD.

- Una LCD para mostrar los resultados de un contador

La práctica 2.2 consiste en familiarizarse con las puertas lógicas.

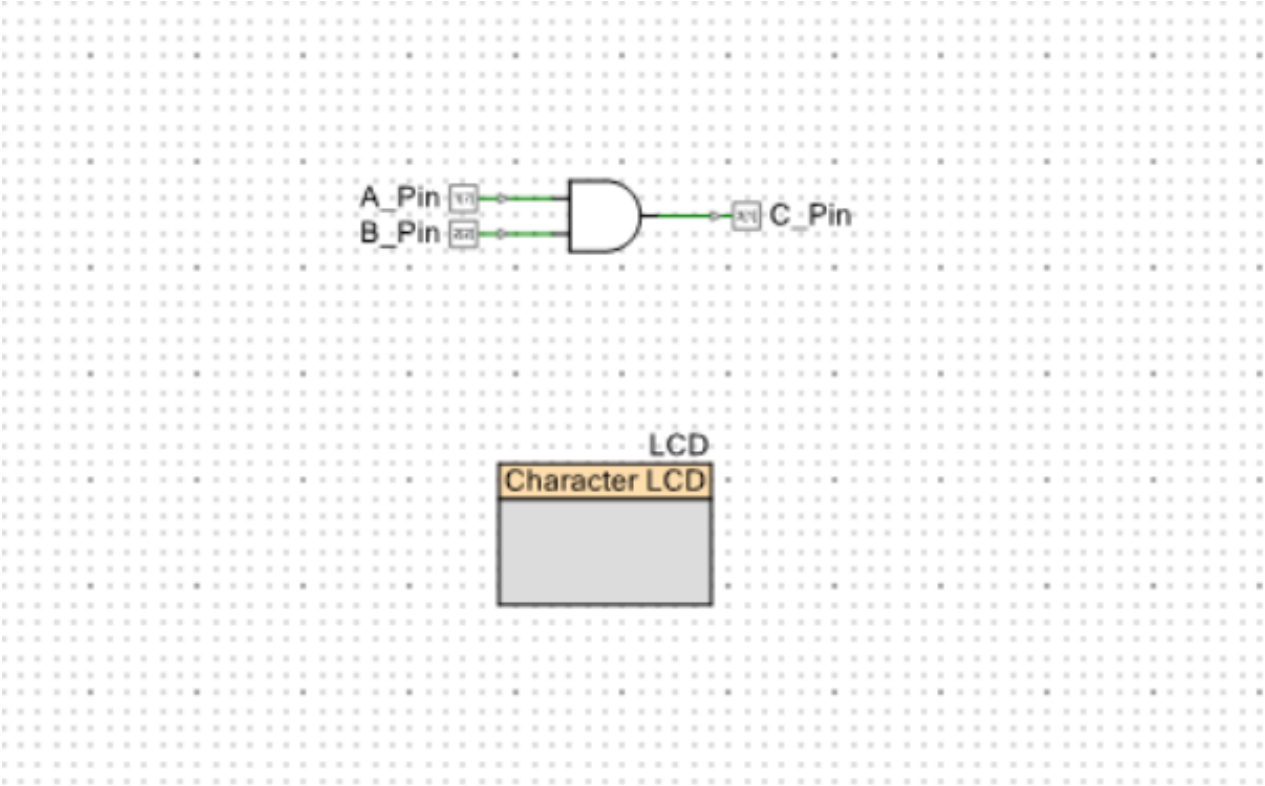
- Uso de una puerta AND para encender un led.

Código

```
#include "project.h"
#include "display.h"

int main(void)
{
    int contador = 0;
    LCD_Start();
    DisplayWelcome();
    DisplayTitle();

    for(;;)
    {
        DisplayCount(contador);
        contador++;

        if(contador>99)
            contador=0;

        CyDelay(500);
    }
}
```

Éste programa lo que hace es mostrarnos por la pantalla LCD un simple contador, que va desde el 0 hasta el 99.
Utilizamos para imprimir: `DisplayCount(contador),` y la variable contador es la que va aumentando cada iteración hasta llegar al 99, y empezamos de 0

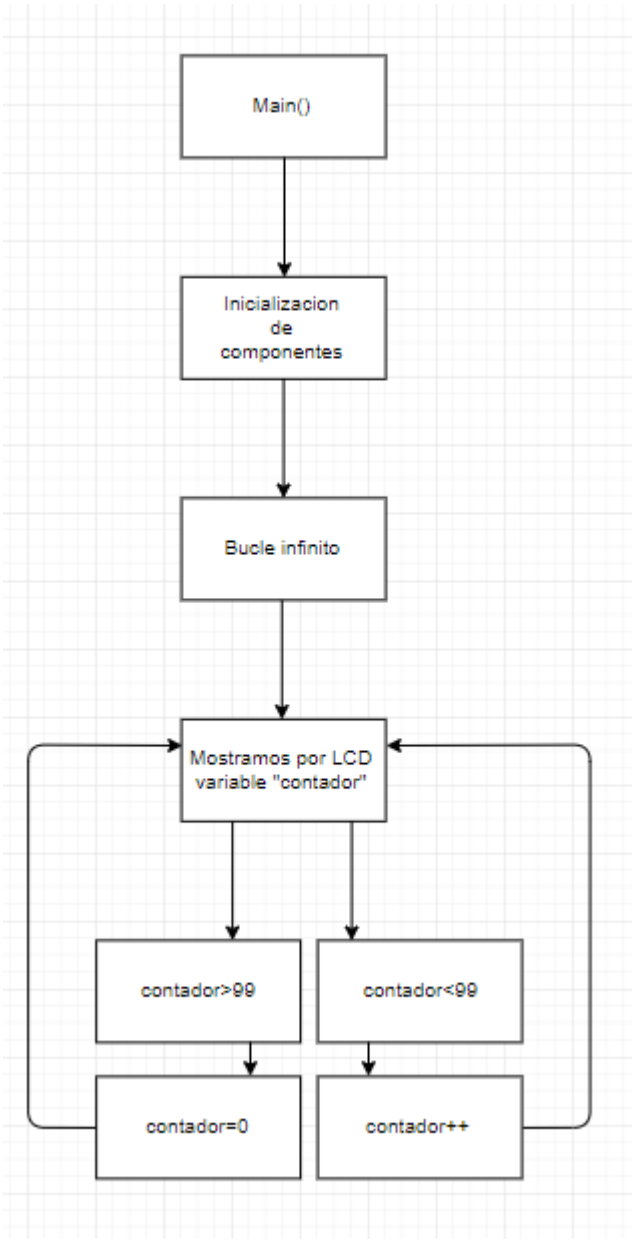
Top Design y explicación:

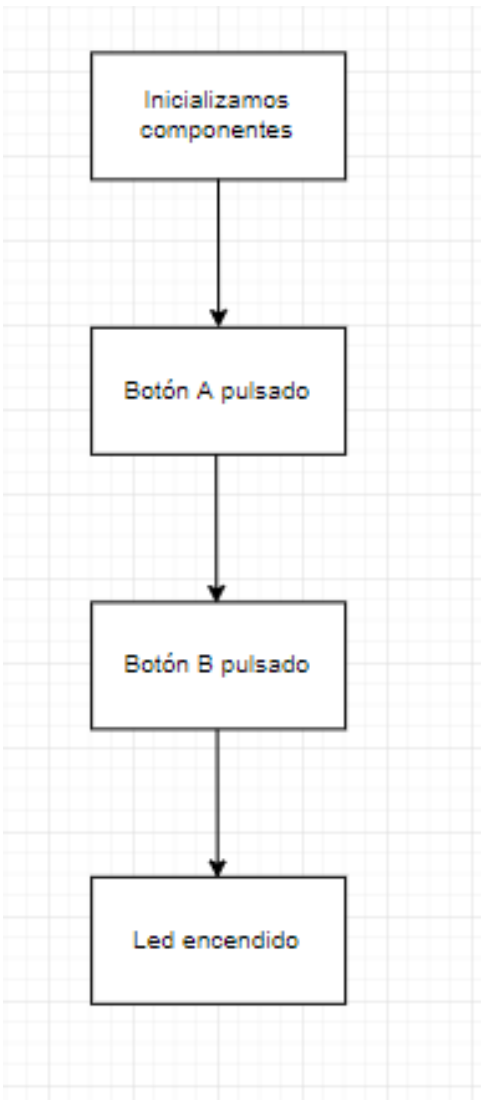


En la parte de arriba tenemos una puerta AND que como entrada tiene los pines A y B que se corresponden a los dos botones y como salida tiene el pin C que se corresponde con el led. El led solo se enciende cuando los dos botones están pulsados.

En la parte de abajo tenemos la LCD que muestra los valores del contador.

Diagrama de flujo:





Práctica 2.3. Utilidades del PSoC para Sistemas Digitales

Esta parte pretende hacer una introducción a:

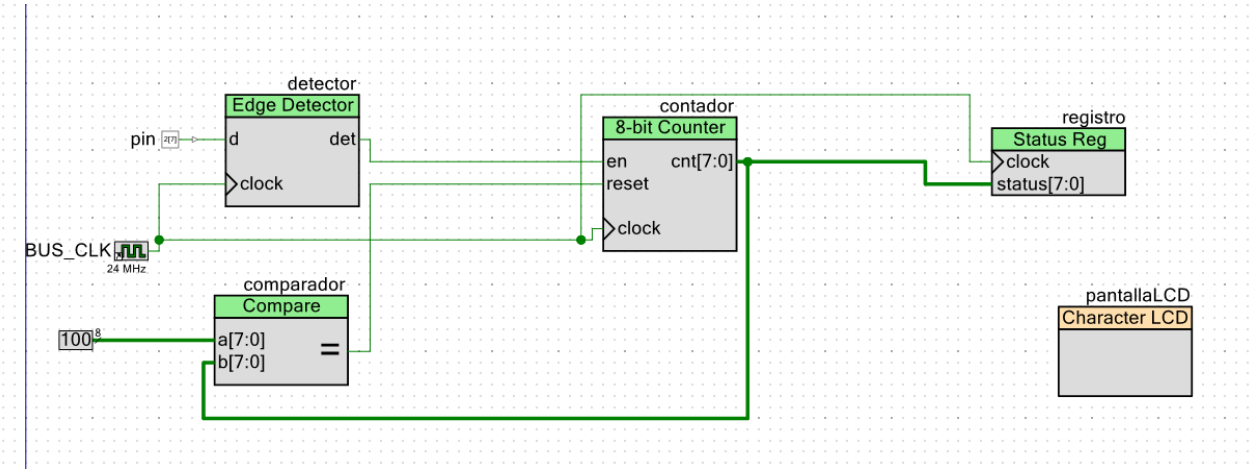
- Un contador básico para contar los flancos de bajada de un pin.
- Un comparador digital para hacer un reset sobre el contador anterior.
- Una constante digital para proveer un valor jo digital al comparador.
- Un detector de picos que produce un único pulso en los flancos de bajada a la entrada básica del contador.

Código:

```
#include "project.h"

void MuestraCadena(char* linea1,  char* linea2){
   pantallaLCD_ClearDisplay();
   pantallaLCD_Position(0U, 0U);
   pantallaLCD_PrintString(linea1);
   pantallaLCD_Position(1U, 0U);
   pantallaLCD_PrintString(linea2);
   // pantalla_PrintString(linea2);

}

int main(void)
{
   CyGlobalIntEnable; /* Enable global interrupts. */

   pantallaLCD_Start();
   pantallaLCD_PrintNumber(registro_Read());


   /* Place your initialization/startup code here (e.g. MyInst_Start()) */

   for(;;)
   {
      pantallaLCD_ClearDisplay();
      pantallaLCD_PrintNumber(registro_Read());
      CyDelay(600);

   }
}
```

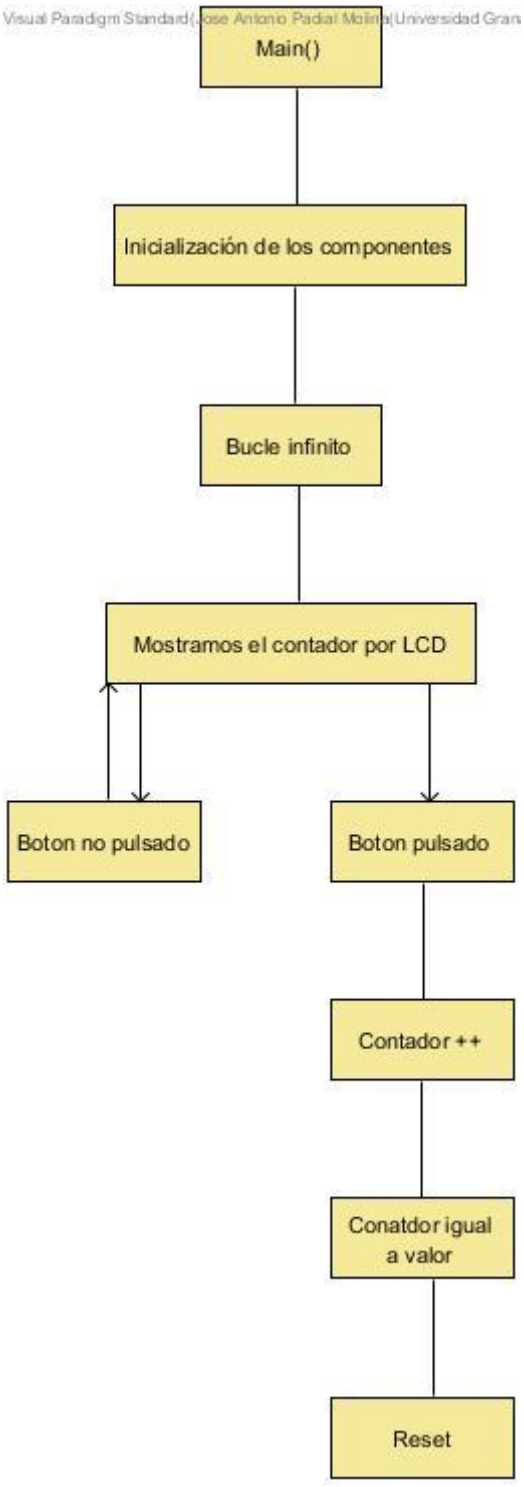
El resultado de este ejercicio será ver en la LCD un contador que se actualizará conforme el usuario pulse el botón.

Top Design:



Tenemos un pin de entrada al edge detector. Edge Detector toma muestras de la señal conectada y produce un pulso cuando se produce el borde seleccionado. Se usa cuando un circuito necesita responder a un cambio de estado en una señal. Donde la salida es la entrada del contador de 8 bit. El reset de 8 bit es el comparador cuando es igual la salida del contador a la entrada a del comparador lo resetea. Y por pantalla se imprime el valor de salida del contador que se guarda en un registro.

Diagrama de flujo:



Práctica 2.4. Diseño de un multiplicador de 2 bits

Objetivos:

Implementar un multiplicador de dos bits, cuyas entradas se controlan con tres botones, un botón servirá para aumentar el valor del multiplicador, otro para disminuirlo, y otro para pasar al siguiente valor o al resultado.

Material de apoyo:

Para el diseño del multiplicador, se utilizará la herramienta de diseño de circuitos digitales open source LogiSim.

Valores de la tabla:

A_0	A_1	B_0	B_1	R_0	R_1	R_2	R_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	1	0
1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	1	1	0
1	1	1	0	1	1	0	0
1	1	1	1	1	0	0	1

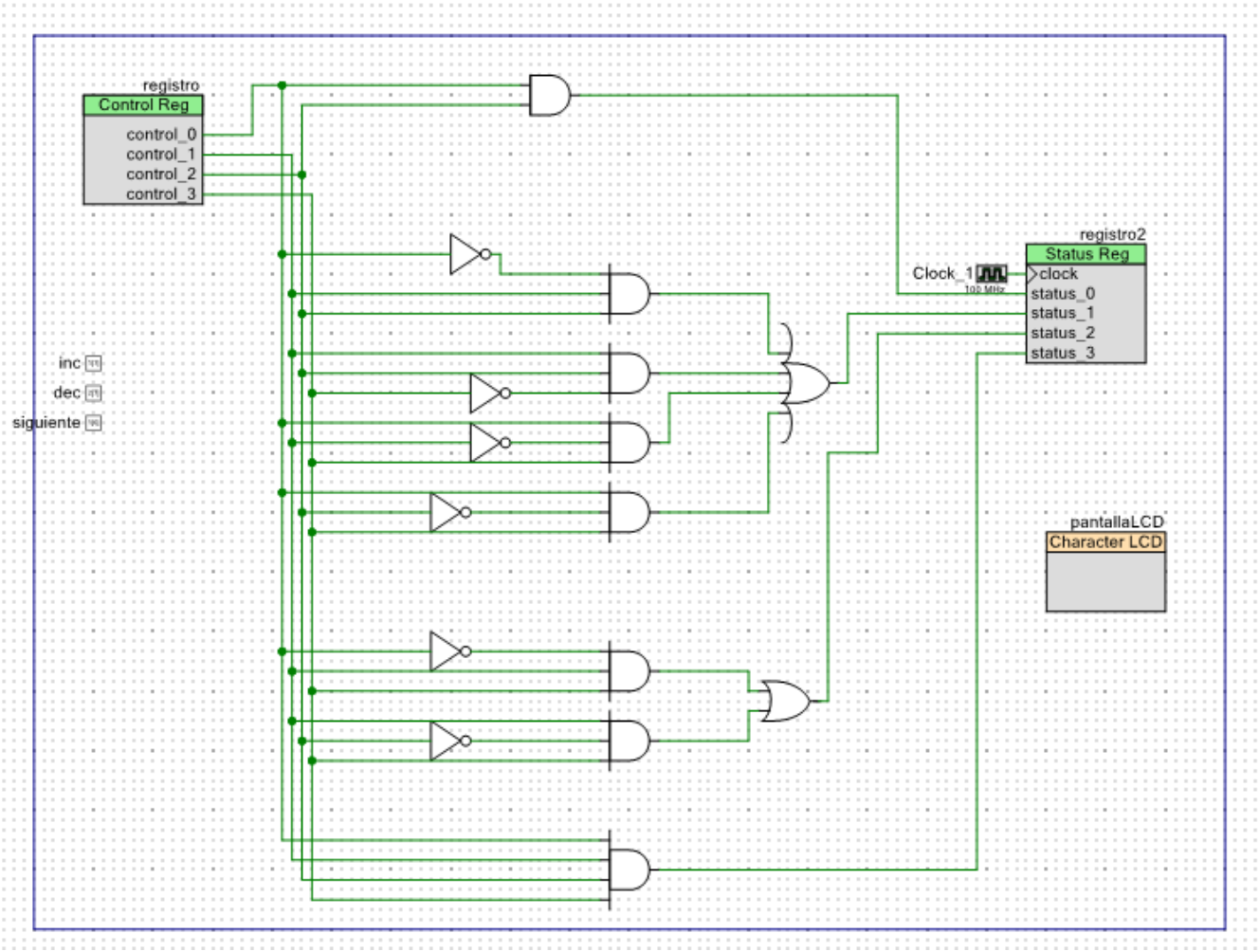
Código:

```
#include "project.h"
```

```
int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */
    pantallaLCD_Start();

    CyDelay(1000);
    pantallaLCD_PrintString("Bienvenido a la practica 2");

    CyDelay(1000);
    pantallaLCD_ClearDisplay();

    uint8 a = 0U;
    uint8 b = 0U;
    uint8 modo = 0U;
    uint8 resultado = 0U;


    /* Place your initialization/startup code here (e.g. MyInst_Start()) */

    for(;;)
    {


        if(siguiente_Read()){
            modo = (modo + 1) % 3;
            while(!siguiente_Read()){
            }
        }

        switch(modo){
            case 0:
                if(!inc_Read()){
                    a = (a+1)%4;
                    while(!inc_Read()){
                    }
                }else if(!dec_Read()){
                    a = (a-1)%4;
                    while(!dec_Read()){
                    }
                }
                break;
            case 1:
                 if(!inc_Read()){
                    b = (b+1)%4;
                    while(!inc_Read()){
                    }
                }else if(!dec_Read()){
                    b = (b-1)%4;
                    while(!dec_Read()){
                    }
```

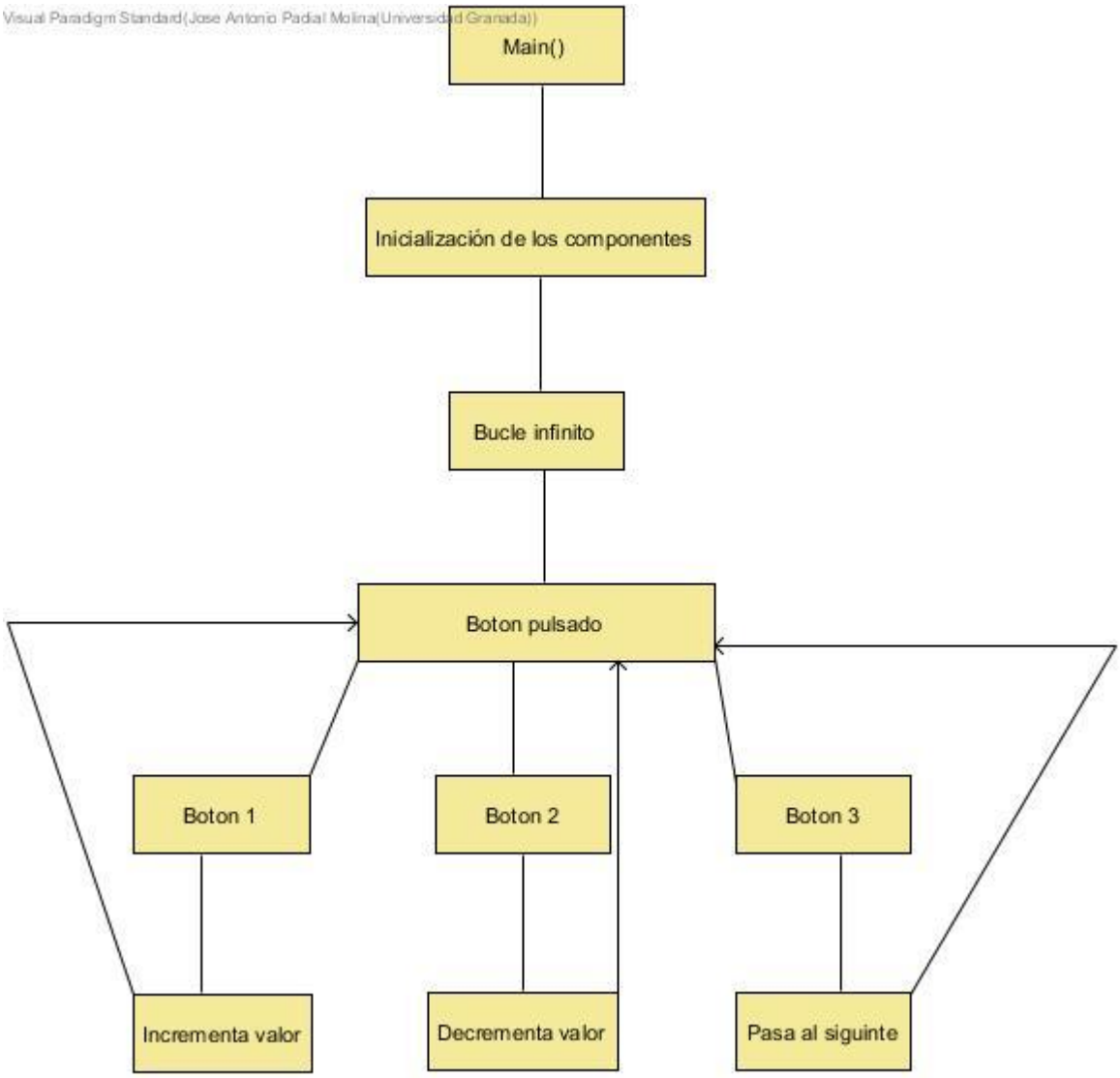
```
                    }
                    break;
                case 2:

                    registro_Write(a + (b << 2));
                    resultado = registro2_Read();

                    break;

            }

                pantallaLCD_ClearDisplay();
                pantallaLCD_Position(0U,0U);
                pantallaLCD_PrintString("A: ");
                pantallaLCD_PrintDecUint16(a);
                pantallaLCD_PrintString(" * B: ");
                pantallaLCD_PrintDecUint16(b);
                pantallaLCD_PrintString(" = ");
                pantallaLCD_Position(1U,0U);
                pantallaLCD_PrintDecUint16(resultado);
                CyDelay(1000);



        }
    }
```

Top Design:



El top design ha sido realizado según los valores obtenidos con el programa LogiSim.

Diagrama de flujo:



Práctica 2.5. Aplicación de un Sistema Digital

La práctica consiste en implementar un sistema que resuelva el problema típico de “Granjero-Zorro-Gallina-Maíz”. Los componentes usados son:

- Una LCD para mostrar mensajes de acierto y de error.
- Dos botones para controlar quién cruza.

Código:

Main

```
#include "project.h"


int main()
{

	int estado[4] = {0,0,0,0};
	int pos = 0;
	int error = 0;

	CyGlobalIntEnable; /* Enable global interrupts. */

	int uestado = estado[0]+(estado[1]<<1)+(estado[2]<<2)+(estado[3]<<3);

	Paso_sig_Write(uestado);
	LCD_Start();
	for(;;)
	{
	if (!Siguiente_Read()) {
	pos = (pos+1)%4;
	while(!Siguiente_Read()); //Esperamos a que se suelte el boton
	} else if (!Marcar_Read()) {
	if (pos != 3) {
	if ((estado[0] == 0 && estado[pos+1] == 0)||(estado[0] == 1 && estado[pos+1] == 1))
		estado[pos+1] = 1-estado[pos+1];
	}

	//Damos un pulso de reloj
	Reloj_controlado_Write(1);
	Reloj_controlado_Write(0);

	//El estado sera la combinacion de tu posicion(granjero) mas la de los tres componentes
	uestado = estado[0]+(estado[1]<<1)+(estado[2]<<2)+(estado[3]<<3);
	Paso_sig_Write(uestado);

	estado[0] = 1-estado[0];

	//Esperamos a que el usuario suelte el boton
	while(!Marcar_Read());
	}
	//Comprobamos si el paso escogido es erroneo
	error = Paso_erroneo_Read();

	LCD_ClearDisplay();
	LCD_Position(0,0);

	//Imprimimos en el LCD por que ha perdido el jugador, segun el codigo de error devuelto
	if (error == 1) {
		LCD_PrintString("La gallina te");
	LCD_Position(1,0);
		LCD_PrintString("dejo sin maiz");
```

```
            CyDelay(10000);
            } else if (error == 2) {
                        LCD_PrintString("La gallina ya no");
                        LCD_Position(1,0);
                        LCD_PrintString("te dara problema");
            CyDelay(10000);
            } else if (Paso_actual_Read() != (Ultimo_Paso_Read()+1) && (Paso_actual_Read() !=
Ultimo_Paso_Read())) {
            LCD_PrintString("Paso incorrecto");
            CyDelay(5000);
            } else if (Paso_actual_Read() == 7U) {
            LCD_PrintString("Tremenda jugada!");
            } else {

                        LCD_PrintString(" M:");
            LCD_PrintDecUint16(1-estado[1]);

            LCD_PrintString(" G:");
            LCD_PrintDecUint16(1-estado[2]);

            LCD_PrintString(" Z:");
            LCD_PrintDecUint16(1-estado[3]);

            LCD_Position(1,0);
                        LCD_PrintString(" M:");
            LCD_PrintDecUint16(estado[1]);

            LCD_PrintString(" G:");
            LCD_PrintDecUint16(estado[2]);

            LCD_PrintString(" Z:");
            LCD_PrintDecUint16(estado[3]);
            if (pos != 3) {
            if (estado[0] == 0)
                        LCD_Position(0,(pos*4));
            else
                        LCD_Position(1,(pos*4));
            LCD_PrintString("*");
            }
            }

            CyDelay(100);
            }
}
```

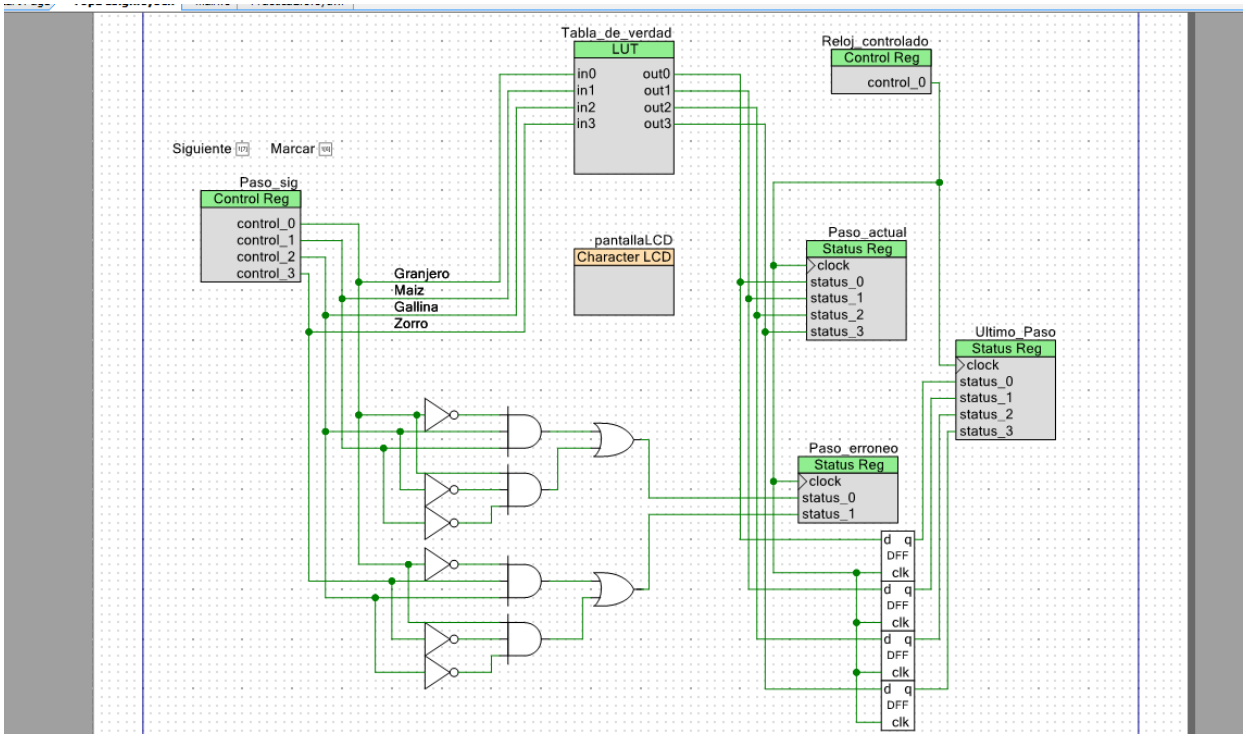
Main: Se inicializan los componentes, las interrupciones y las variables, además, la variable uestado se inicializa a los 4 bits de estado concatenados (0000).

En cada iteración del for se comprueba el estado de los botones, primero el de siguiente, que indicará la casilla que estamos cambiando de valor, si se pulsa la variable pos aumenta en uno, pudiendo marcar así cada uno de los 3 personajes del juego (maíz, gallina y zorro). Si en su lugar se ha pulsado el botón Marcar, lo que haremos será cambiar el valor de la casilla marcada, comprobando en que lado del río nos encontramos(estado[0]).

Por ultimo despues de comprobar los botones toca comprobar que resultado tiene el movimiento realizado, sera 1 si la gallina se ha comido el maiz o 2 si el zorro a la gallina (Esto nos lo da el resultado del circuito lógico creado a partir de LogiSim, el cual comprueba si se ha quedado la gallina sola con el maíz o el zorro solo con la gallina). Si no se da una de estas condiciones comprobamos si el paso realizado es erroneo, o si ha ganado.

Antes de finalizar la iteracion actualizamos la informacion que aparece en el LCD.

Top Design y explicación:

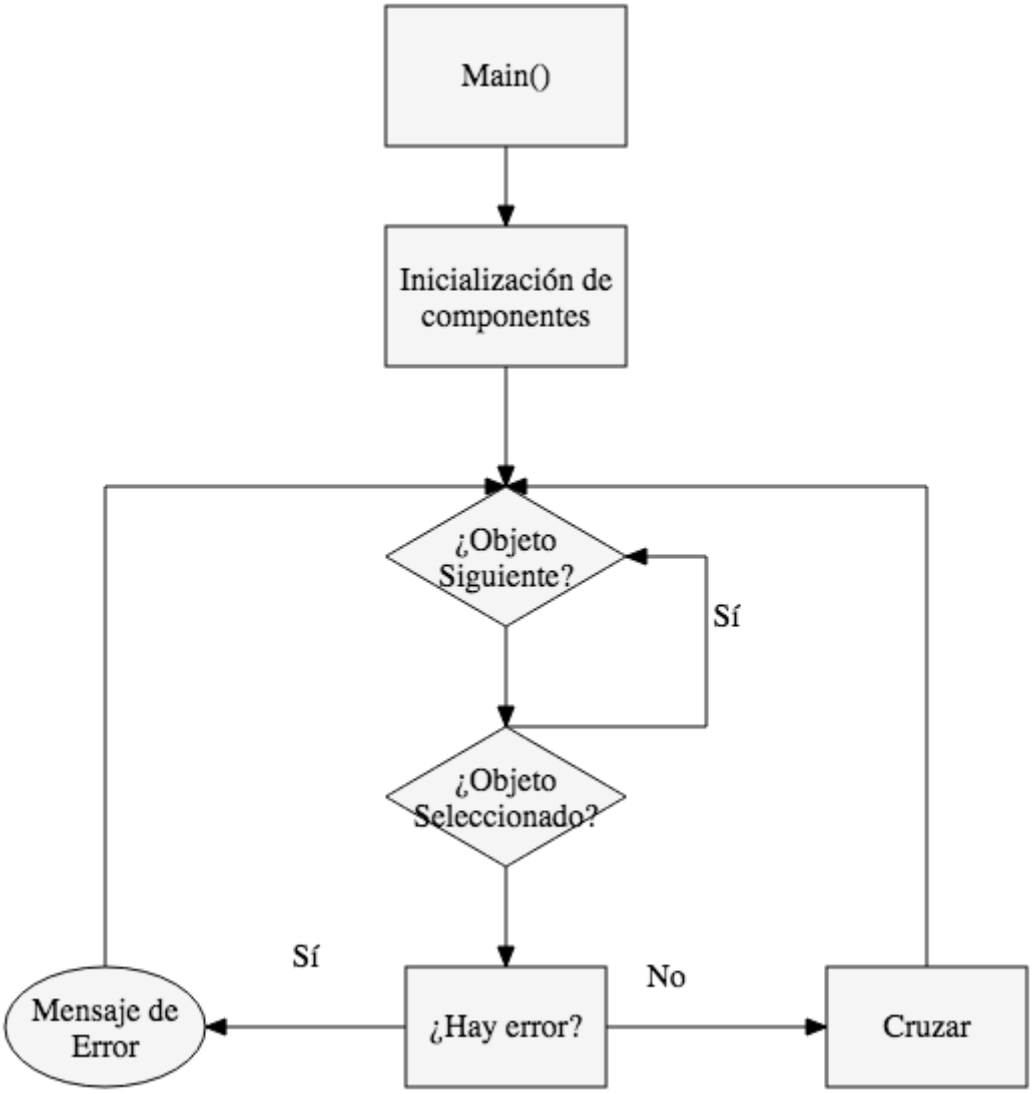


En la parte de arriba del Top Design podemos ver la pantalla LCD y los dos botones.

En la página del Top Design de abajo podemos ver:

- Un Control Reg el cual usamos como reloj, para poder avanzar el juego cuando queramos.
- Otro Control Reg usado como entradas de la look-up-table (LUT), la cual tiene en su interior la tabla de verdad con todas las combinaciones de entrada, y esas salidas las pasa a un Status Reg, el cual guarda el estado del paso actual.
- Abajo a la izquierda encontramos el circuito realizado con LogiSim, el cual se genera al meterle la tabla de verdad al LogiSim para las combinaciones de entrada que significan que has perdido. El resultado de este circuito va a parar a un Status Reg que guarda el estado de error (si se produce error o no).
- Por último, abajo a la derecha encontramos los Biestables tipo D conectados a un Status Reg, para poder sincronizar los datos con el avance del juego, el cual guarda el estado del último paso.

Diagrama de flujo:



Práctica 3.1. UART

La práctica consiste en tratar de comunicar el PSoC con nuestro ordenador a través de un cable UART-USB con el objetivo de manejar una señal PWM desde un hiperterminal del ordenador. En nuestro caso hemos utilizado un Buzzer(Pin 3.0) para ver el resultado:

- Se podrá activar o desactivar el PWM, parando o reanudando el sonido.
- Por otro lado se podrá modificar el reloj que usa el PWM, cambiando su frecuencia.

Para la realización de esta práctica hemos usado el hyperterminal proporcionado en el siguiente enlace, conectamos el PSoC como si lo fuéramos a programar (ya que los pines Tx

y Rx están también conectados al Programador del PSoC) y vemos como ya podemos comenzar la comunicación.

Código:

Cabeceras y funciones

```
#include "project.h"
#include <stdio.h>
#include "common.h"

#if defined (__GNUC__)
        asm (".global _printf_float");
#endif

uint8 errorStatus = 0u;
CY_ISR(RxIsr)
{
        uint8 rxStatus;
        uint8 rxData;

        do
        {
        /* Read receiver status register */
        rxStatus = UART_RXSTATUS_REG;

        if((rxStatus & (UART_RX_STS_BREAK    | UART_RX_STS_PAR_ERROR |
                UART_RX_STS_STOP_ERROR | UART_RX_STS_OVERRUN)) != 0u)
        {
        /* ERROR handling. */
        errorStatus |= rxStatus & ( UART_RX_STS_BREAK            | UART_RX_STS_PAR_ERROR |
                        UART_RX_STS_STOP_ERROR | UART_RX_STS_OVERRUN);
        }

        if((rxStatus & UART_RX_STS_FIFO_NOTEMPTY) != 0u)
        {
        /* Read data from the RX data register */
        rxData = UART_RXDATA_REG;
        if(errorStatus == 0u)
        {
        /* Send data backward */
        UART_TXDATA_REG = rxData;

        if(rxData == 'd'){
                Clock_1_SetSource(CYCLK_SRC_SEL_IMO);
        }else if(rxData == 'u'){
                Clock_1_SetSource(CYCLK_SRC_SEL_PLL);
        }else if(rxData == 'p'){
                generadorPulsosPWM_Stop();
                generadorPulsosPWM_Sleep();
                UART_PutString("Vamos a parar el led");
        }else if(rxData == 'a'){
                generadorPulsosPWM_Start();
        }
        Pantalla_Position(1,0);
        Pantalla_PrintString((const char8 *)&rxData);

        }
```

```
        }
        }while((rxStatus & UART_RX_STS_FIFO_NOTEMPTY) != 0u);

}
```

Main

```
int main(void)
{


        #if(INTERRUPT_CODE_ENABLED == ENABLED)
        UART_isr_StartEx(RxIsr);
        #endif /* INTERRUPT_CODE_ENABLED == ENABLED */

        CyGlobalIntEnable;          /* Enable global interrupts. */


        generadorPulsosPWM_Start();

        CyGlobalIntEnable; /* Enable global interrupts. */

        Pantalla_Start();
        UART_Start();
        Pantalla_ClearDisplay();
        Pantalla_PrintString("Bienvenido");

        generadorPulsosPWM_WriteCompare(90);

        /* Place your initialization/startup code here (e.g. MyInst_Start()) */
        for(;;)
        {

        }
}
```

Las funciones toggle_LED1() y toggle_LED2() básicamente cambian el estado de un led, es decir, si está apagado lo enciende y si está encendido lo apaga.

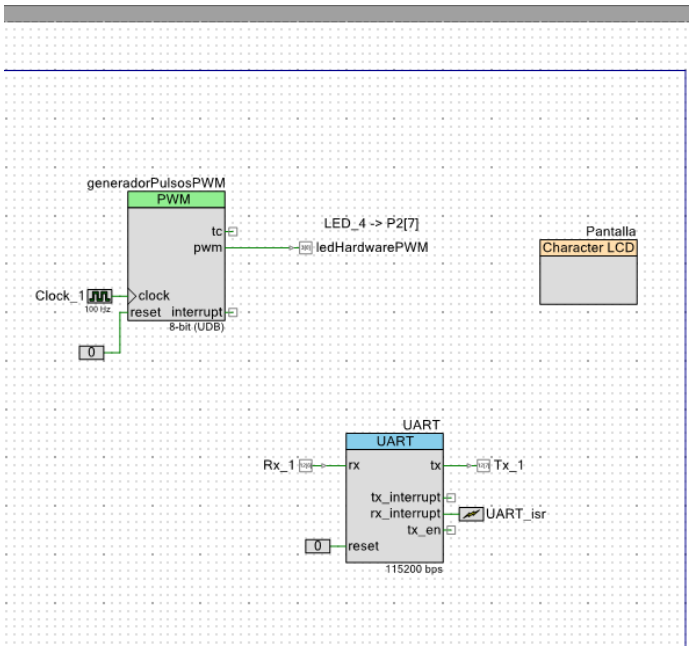
CY_ISR(RxIsr) Será la función que se ejecutará cuando se produzca la interrupción de Rx, es decir de que vamos a recibir datos del ordenador. A grandes rasgos se realiza primero una parte de comprobación de conexión y errores, después, se leen los datos escritos en el registro Rx (rxData = UART_RXDATA_REG;) y por último se comprueba el carácter recibido y se decide qué hacer en consecuencia, así como imprimirlo por el LCD

Main: El main como se puede comprobar por su extensión solo nos sirve como medio para inicializar todos los componentes y las interrupciones.

***Nota:** Como código esquema se usó el proporcionado como ejemplo por el propio PSoC Creator para comunicación UART con Tx y Rx. Por tanto se usa también el common.h que genera este ejemplo, aunque su uso es solo para definir:

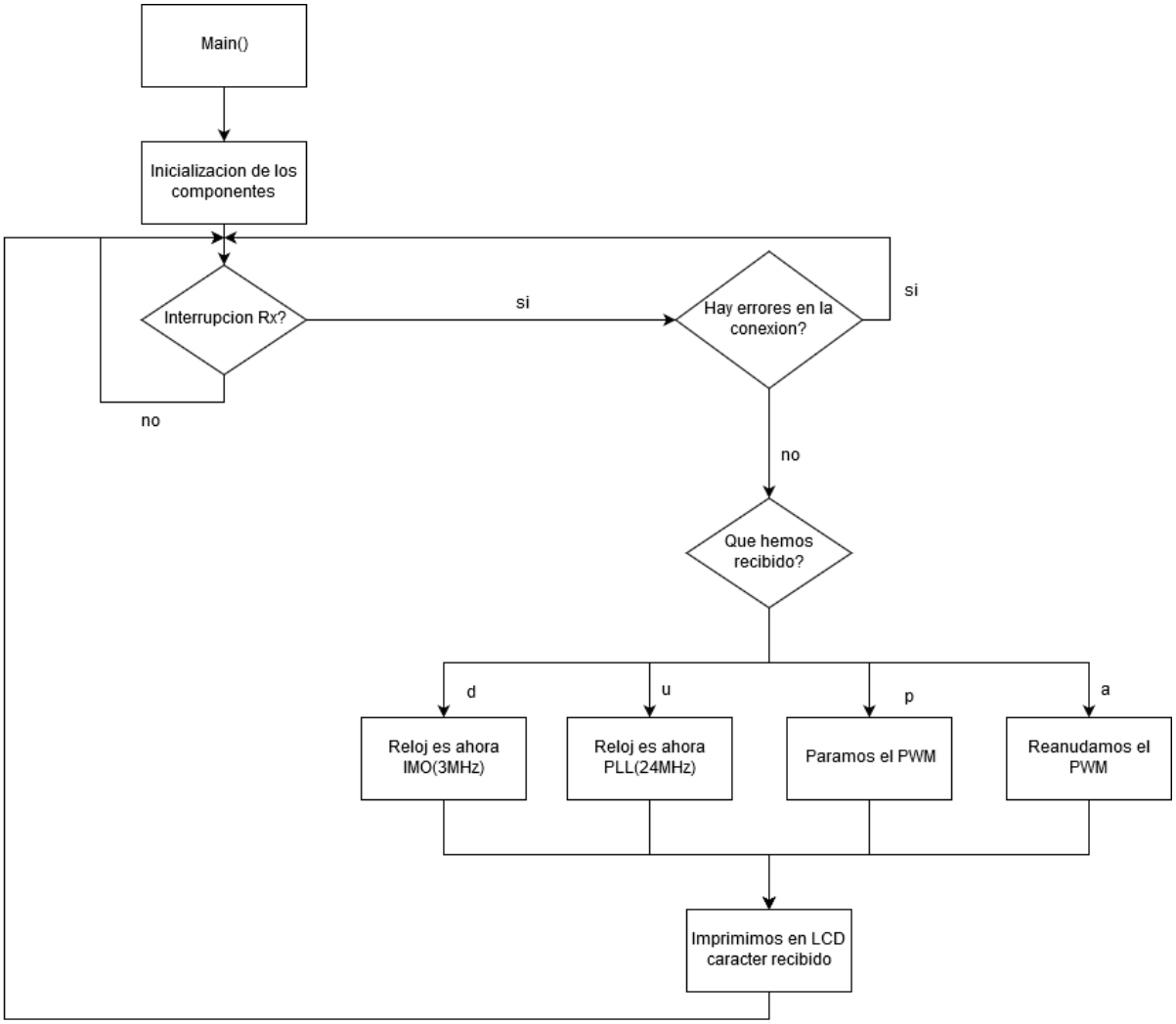
```
#define UART_PRINTF_ENABLED          ENABLED
#define INTERRUPT_CODE_ENABLED   ENABLED
```

Top Design y explicación:



El esquemático como se puede ver contendrá, por un lado, el PWM para enviar la señal modulada al Buzzer, el UART para realizar la conexión al ordenador con su interrupción solamente al recibir datos; y por otro lado la pantalla LCD para imprimir los datos recibidos y comprobar que son correctos. El UART se configura a 115200 baudios y se activa la interrupción de Rx. Al PWM se le conecta un reloj que luego modificaremos desde el ordenador.

Diagrama de flujo:



Práctica 4.1. Voltímetro

La práctica consiste en implementar un sistema que funcione como un voltímetro, así que tendremos un potenciómetro y mediremos la salida analógica de este para mostrarla por pantalla, en resumen se usará:

- Un Potenciómetro medido por un conversor analógico a digital.
- Una pantalla LCD para mostrar los datos.

Nosotros ya debimos realizar este sistema para nuestro proyecto (los pedales), como eran dos pedales y la PSOC solo tiene un conversor analógico-digital usamos un multiplexor para poder usar los dos pedales, de este modo nos quedan dos voltímetros.

Código:

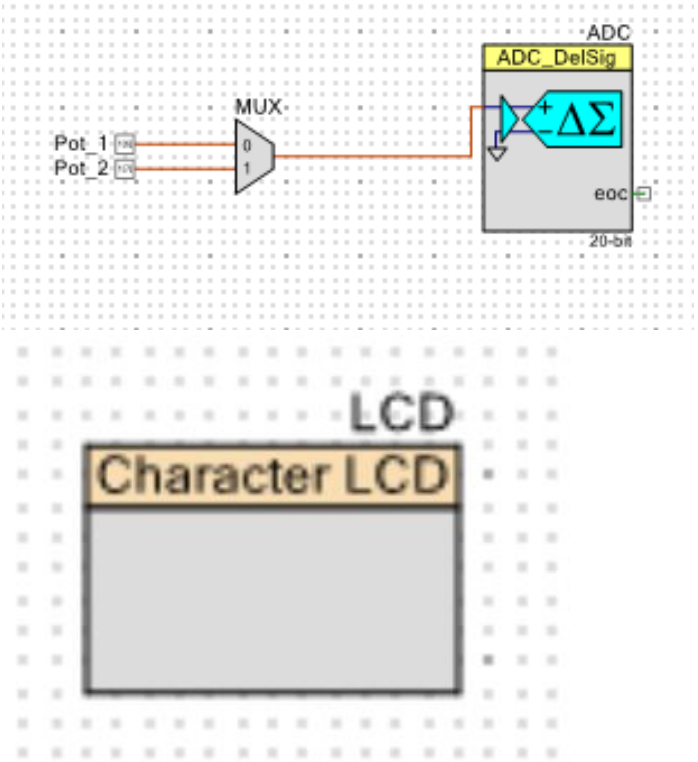
Main

```
int main()
{
   uint8_t canal = 0;
   int32 adcread;
   float voltaje;

   MUX_Start();
   LCD_Start();
   LCD_ClearDisplay();
   ADC_Start();
   ADC_StartConvert();
   ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);

   for (;;){

      MUX_FastSelect(canal);
      adcread=ADC_GetResult32();
      voltaje=floor(((5.000/1048576)*adcread));

      LCD_Position(1,canal*8);
      LCD_PrintString("V");
      LCD_PrintNumber(canal);
      LCD_PrintString(": ");
      LCD_PrintNumber(voltaje);
      canal=1-canal;

      CyDelay(50);
   }
}
```

Main: Primero se inicializa la variable canal, la cual la usamos para ir alternando en cada iteración entre un pedal y otro. A continuación se declaran otras variables como la variable que guarda el voltaje en sí y otra variable del tipo int32. Se inicializan los componentes, multiplexor, pantalla LCD y Conversor Analógico-Digital.
Dentro del bucle for(;;):

1. Se selecciona el canal (pedal 0 o pedal 1).
2. Se le da valor a adcread.
3. Se calcula voltaje y se le hace un floor para que redondee y podamos mostrarlo mejor por pantalla (en nuestro caso lo hacíamos así para decidir la posición del jugador aunque se podría no haber redondeado y mostrarlo usando un %2.3f que muestra solo los tres primeros decimales).
4. Dependiendo del canal escribimos en una parte u otra de la LCD. Lo que se escribe es algo del tipo: “V1:3”
5. Cambiamos de canal y realizamos un delay.

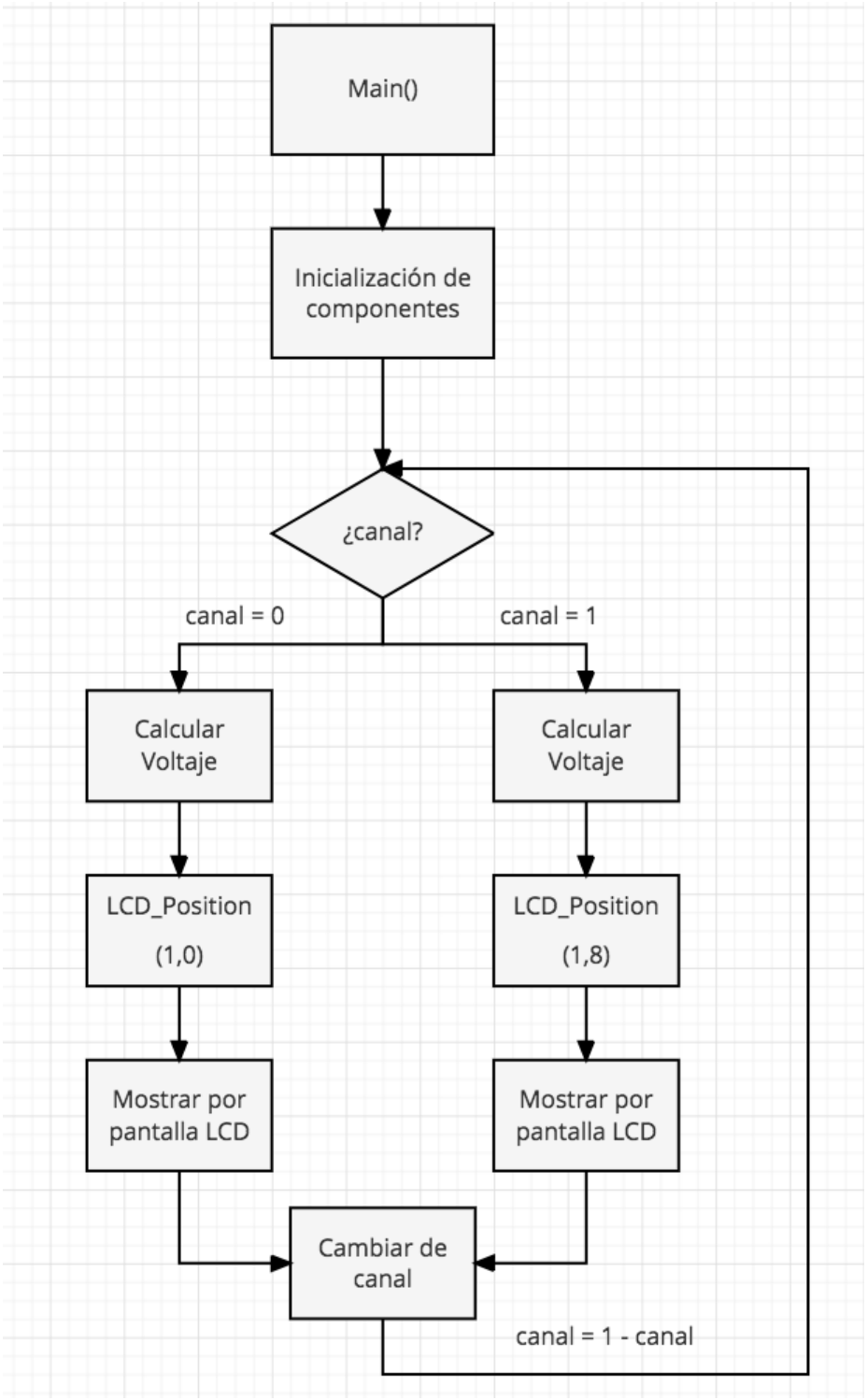
Top Design y explicación:



Podemos observar el Multiplexor con los dos canales (pedales) como entradas y la salida conectada al Conversor Analógico-Digital.

También podemos observar la pantalla LCD por la cual mostramos los mensajes.

Diagrama de flujo:


Main()
Inicialización de componentes
¿canal?
canal = 0
canal = 1
Calcular Voltaje
Calcular Voltaje
LCD_Position (1,0)
LCD_Position (1,8)
Mostrar por pantalla LCD
Mostrar por pantalla LCD
Cambiar de canal
canal = 1 - canal

Nota: Como no disponíamos de un osciloscopio esta práctica está exenta de pruebas, por tanto no se sabe si su funcionamiento es correcto.

La práctica consiste en tratar de comunicar el PSoC con nuestro ordenador a través de un cable UART-USB con el objetivo de manejar la amplitud y la frecuencia de una onda desde un hiperterminal del ordenador:

- Se podrá aumentar/decrementar la amplitud(caracteres ‘a’ o ’z’), aumentar/decrementar la frecuencia(caracteres ‘s’ o ‘x’) y cambiar el tipo de onda(seno, cuadrada, sierra o triangular, con el carácter ‘t’).

Para la realización de esta práctica hemos usado el hyperterminal proporcionado en el siguiente enlace, conectamos el PSoC como si lo fuéramos a programar (ya que los pines Tx y Rx están también conectados al Programador del PSoC) y vemos como ya podemos comenzar la comunicación.

Para el manejo de la onda definimos los parámetros en la interrupción y en el main, mediante funciones matemáticas, definimos una muestra de la onda que deseamos crear.

Código:

Cabeceras y funciones

```
#include <math.h>
#include <stdlib.h>
#include "project.h"
#include <stdio.h>
#include "common.h"

#if defined (__GNUC__)
   asm (".global _printf_float");
#endif


#define PI_DOS (3.14159 * 2)

#define MUESTRAS 4000

#define SENO 0
#define CUADRADA 1
#define SIERRA 2
#define TRIANGULAR 3

char op;
uint8 signal [MUESTRAS];
uint8 type = SENO;
uint8 amp = 127;
uint16 i = 0;
uint8 j = 0;

float incrementarFase = PI_DOS/MUESTRAS;

uint8 errorStatus = 0u;
```

```
CY_ISR(RxIsr)
{
  uint8 rxStatus;
  uint8 rxData;

  do
  {
    /* Read receiver status register */
    rxStatus = UART_RXSTATUS_REG;

    if((rxStatus & (UART_RX_STS_BREAK      | UART_RX_STS_PAR_ERROR |
                UART_RX_STS_STOP_ERROR | UART_RX_STS_OVERRUN)) != 0u)
    {
      /* ERROR handling. */
      errorStatus |= rxStatus & ( UART_RX_STS_BREAK      | UART_RX_STS_PAR_ERROR |
                            UART_RX_STS_STOP_ERROR | UART_RX_STS_OVERRUN);
    }

    if((rxStatus & UART_RX_STS_FIFO_NOTEMPTY) != 0u)
    {
      /* Read data from the RX data register */
      rxData = UART_RXDATA_REG;
      if(errorStatus == 0u)
      {
        /* Send data backward */
        UART_TXDATA_REG = rxData;

        switch(rxData)
        {
          case 'a':
          case 'A':
            amp += 5;
            UART_PutString("Amplitud aumentada\r\n");
            Pantalla_ClearDisplay();
            Pantalla_Position(0u, 0u);
            Pantalla_PrintString("Amplitud aumentada");

            break;
          case 'z':
          case 'Z':
            amp -= 5;
            UART_PutString("Amplitud decrementada\r\n");
            Pantalla_ClearDisplay();
            Pantalla_Position(0u, 0u);
            Pantalla_PrintString("Amplitud decrementada");
            break;
          case 's':
          case 'S':
            if ( ((Clock_1_GetDividerRegister()+1)/2) >=1 )
            Clock_1_SetDividerValue((Clock_1_GetDividerRegister()+1)/2);
            UART_PutString("Frecuencia aumentada\r\n");
            Pantalla_ClearDisplay();
            Pantalla_Position(0u, 0u);
            Pantalla_PrintString("Frecuencia aumentada");

            break;
          case 'x':
          case 'X':
            Clock_1_SetDividerValue((Clock_1_GetDividerRegister()+1)*2);
```

```
                UART_PutString("Frecuencia decrementada\r\n");
                Pantalla_ClearDisplay();
                Pantalla_Position(0u, 0u);
                Pantalla_PrintString("Frecuencia decrementada");

                break;
            case 't':
            case 'T':
                type = (type+1)%4;
                UART_PutString("Tipo de onda cambiado\r\n");
                Pantalla_ClearDisplay();
                Pantalla_Position(0u, 0u);
                Pantalla_PrintString("Tipo de onda cambiado");
                break;
            default:
                UART_PutString("Comando desconocido\r\n");

                Pantalla_ClearDisplay();
                Pantalla_Position(0u, 0u);
                Pantalla_PrintString("Comando desconocido");
                break;
        }

      }
    }
  }while((rxStatus & UART_RX_STS_FIFO_NOTEMPTY) != 0u);

}
```

Main

```
int main(void)
{


  #if(INTERRUPT_CODE_ENABLED == ENABLED)
  UART_isr_StartEx(RxIsr);
  #endif /* INTERRUPT_CODE_ENABLED == ENABLED */

  CyGlobalIntEnable;      /* Enable global interrupts. */

  generadorOnda_Start();

  Pantalla_Start();
  UART_Start();
  Pantalla_ClearDisplay();
  Pantalla_PrintString("Bienvenido!!!");

  /* Place your initialization/startup code here (e.g. MyInst_Start()) */
  for(;;)
  {
          switch (type) {
            case SENO:
                for (i=0; i < MUESTRAS; i++) {
                    signal[i] = amp*sin(incrementarFase*i)+amp;
                }
```

```
                break;
            case CUADRADA:
                for (i=0; i < MUESTRAS; i++) {
                    signal[i] = (amp*sin(incrementarFase*i)>=0)?amp*2:0;
                }
                break;
            case SIERRA:
                for (i=0; i < MUESTRAS; i++) {
                    signal[i] = 2*amp*(float)(i / MUESTRAS);
                }
                break;
            case TRIANGULAR:
                for (i=0; i < MUESTRAS; i++) {
                    signal[i] = 2*amp - abs(i % (2*2*amp) - 2*amp);
                }
                break;
        }
        generadorOnda_Stop();
        generadorOnda_Wave1Setup(signal, 4000);
        generadorOnda_Start();
        CyDelay(1000);
  }
}
```

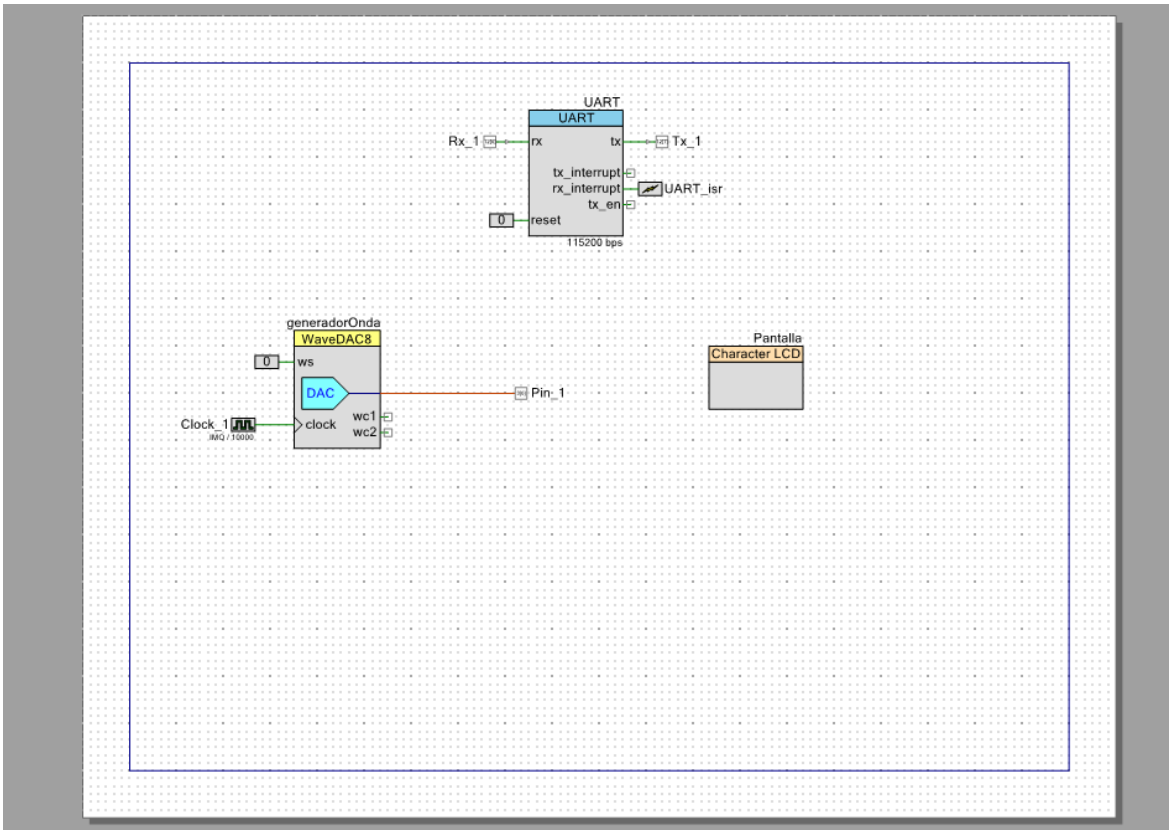
CY_ISR(RxIsr) Será la función que se ejecutará cuando se produzca la interrupción de Rx, es decir de que vamos a recibir datos del ordenador. A grandes rasgos se realiza primero una parte de comprobación de conexión y errores, después, se leen los datos escritos en el registro Rx (rxData = UART_RXDATA_REG;) y por último se comprueba el carácter recibido y se decide qué hacer en consecuencia, así como imprimirlo por el LCD

Main: El main nos sirve como medio para inicializar todos los componentes y las interrupciones, además, tendrá un bucle infinito que será el encargado de construir una muestra de la onda que queremos para que el generador construya una onda completa, pudiendo variar entre seno, cuadrada, sierra y triangular.

***Nota:** Como código esquema se usó el proporcionado como ejemplo por el propio PSoC Creator para comunicación UART con Tx y Rx. Por tanto se usa también el common.h que genera este ejemplo, aunque su uso es solo para definir:

```
#define UART_PRINTF_ENABLED         ENABLED
#define INTERRUPT_CODE_ENABLED   ENABLED
```

Top Design y explicación:



El esquemático estará formado por el propio generador de ondas, al cual se le conectara un reloj que luego variamos y un pin que servirá de salida para la onda, el UART para realizar la conexión al ordenador con su interrupción solamente al recibir datos; y por otro lado la pantalla LCD para imprimir los datos recibidos y comprobar que son correctos. El UART se configura a 115200 baudios y se activa la interrupción de Rx.

Diagrama de flujo: