Proyecto Final 5º Año - Ingeniería Eléctronica

Adquisición de Voz a Distancia con Conexión a PC

Integrantes:

Delfino, Ariel.
Insaurralde, Pablo.
Henze, Agustín.
Redolfi, Javier.

20 de octubre de 2008

Índice

Índice de figuras

1. Anteproyecto

Título Propuesto: Adquisición de Voz a Distancia con Conexión a PC.

1.1. Resumen Preliminar

El proyecto consiste en la construcción de un sistema de Adquisición de Voz Inalámbrico con conexión a PC para una posterior utilización de la señal adquirida.

1.2. Objetivos

Lograr un sistema capaz de adquirir el discurso de un locutor, durante un tiempo determinado, para posteriormente realizar distintas operaciones como: *mostrar la señal adquirida, su espectro de frecuencia, realizar compresiones de la misma, reproducción local o remota (vía red).*

1.3. Desarrollo

El sistema de medición de la voz se realizaría a través de un acondicionador de señal, pasando por el **A/D** de un microntrolador para su posterior envío vía RF[1] al receptor, el cual se comunicaría por **USB** con la PC, la cual tendra el control absoluto del tiempo de Adquisición.

Luego con el Software se podría elegir entre las acciones nombradas anteriormente.

El Hardware, Software y Documentación se realizaría con Software Libre (en lo posible).

Todo el proyecto estaría licenciado con la **GPLv3**[2](General Public License v3) y se publicaría en distintas WEB interesadas para la posible reutilización del mismo por parte de terceros.

1.4. Datos los Autores

Nombres y Apellido	Curso	Legajo	Firma del Alumno
Henze Agustín	5R2	48523	
Delfino Ariel	5R2	47593	
Redolfi Javier	5R2	52536	
Insaurralde Pablo	5R2	48104	

1.5. Materias Integradas

- Medidas Electrónicas II
- Electrónica Aplicada III
- Técnicas Digitales III

[1] 2.4 GHz
[2] http://www.gnu.org/copyleft/gpl.html

2. Desarrollo

El proyecto se dividió en varias partes básicas y se trabajo en la construcción de ellas por separado. Este tipo de organización facilito el trabajo. Cuando se tuvieron funcionando todas las partes por separado se realizó la unión y se hizo la prueba final.

Las partes básicas en que se dividió el proyecto son:

- Circuito de alimentación
- Acondicimiento de la señal de Audio
- Adquisición de la señal de Audio
- Transmisión vía **RF** de los datos
- Recepción vía **RF** de los datos
- Transmisión de los datos a la **PC** a través del puerto **USB**
- Programa de bajo nivel que lee los datos a través del **USB**
- Interfaz de usuario (**GUI**)

Los items anteriores serán tratados en los siguientes capítulos.

3. Acondicimiento de la señal de Audio

La señal con la que se probó el prototipo se sacó de la **PC**, el nivel de esta señal es de $3Vpp$ con un $Vmax$ de 1,5V. Debido a que el microntrolador con el que se hará el muestreo trabaja con tensiones positivas se debió llevar la referencia de la señal a $0V$, para hacer esto se realizó un circuito sumador de $1,5V$, el cual se muestra en la figura 1 .

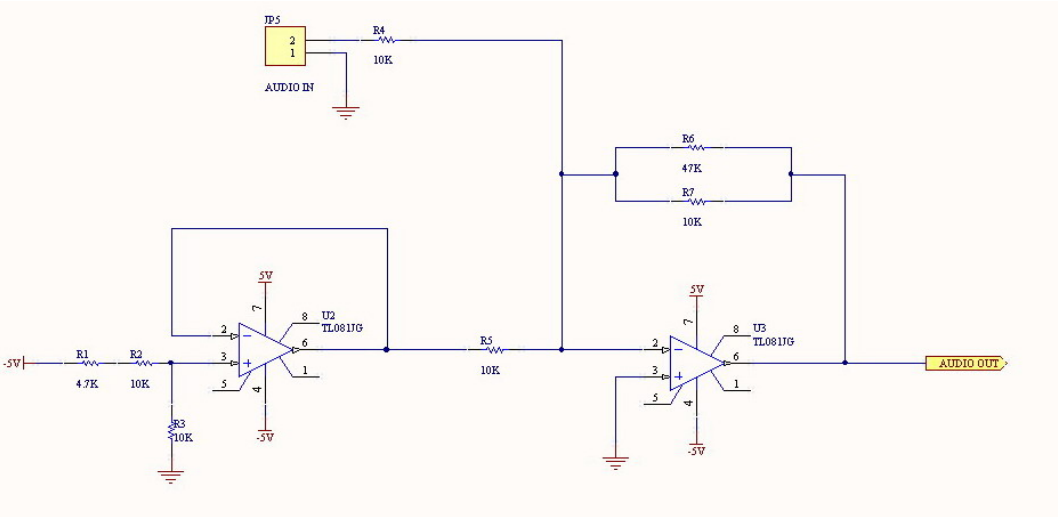

Figura 1: Esquemático del circuito de acondicionamiento.

Aclaramos que esto se podría haber solucionado usando como referencia para el **A/D** $1,5V$ y $-1,5V$, pero nos inclinamos por la primera opción debido al microntrolador elegido, en el cual la última era más complicada.

Los amplificadores operacionales usados son **TL081**. El **TL081** es un amplificador operacional con entrada **J-FET** de alta velocidad, alto voltage de entrada, con fuente partida y transistores bipolares en un sólo circuito integrado. Estos dispositivos tienen alta velocidad de crecimiento, bajas corrientes de deriva y polarización y baja deriva de voltage con respecto a la temperatura.

La desición del operacional no fue muy rigurosa debido a los bajos requisitos de la aplicación. Estos operacionales tienen buenas prestaciones y como los teníamos disponibles nos inclinamos por estos.

En la figura 2 se muestra un diagrama de este operacional.

Las características principales de este operacional son:

- Ancho rango de voltage (hasta +Vcc) de modo común y diferencial.
- Baja corriente de deriva y de polarización.
- Protección contra corto circuito.
- Alta impedancia de entrada debido a la entrada **J-FET**.
- Compensación interna de frecuencia.
- Operación libre de Latch Up.
- Alta velocidad de crecimiento: $16\frac{V}{\mu s}$ típico.

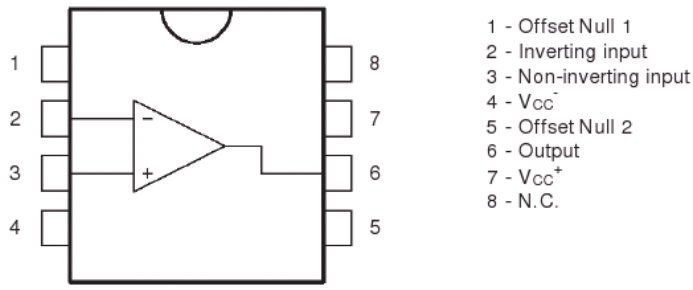

Figura 2: Diagrama **TL081**.

3.1. Cálculo

El circuito consta de dos amplificadores operacionales. El primero genera una referencia de tensión que luego sumaremos a la señal de audio. El segundo es un sumador inversor, que suma la señal de referencia y la de audio. Como a la señal de audio le debemos sumar un valor positivo y el sumador es inversor la tensión de referencia debe ser negativa.

Cálculo del voltage a sumar a la señal

$$V_{sal} = G \cdot V_{ent+}$$
$$V_{in+} = V_{cc-} \cdot \frac{R_3}{R_1 + R_2 + R_3}$$
$$V_{in+} = -5V \cdot \frac{10K}{4K7 + 10K + 10K} = -2{,}02V$$
$$G = 1$$

$$\boxed{V_{sal} = 1 \cdot (-2{,}02V) = -2{,}02V}$$

Cálculo del sumador Inversor

$$V_{sal} = G \cdot (V_{ref} + V_{aud})$$
$$G = \frac{R_7}{R_4} = \frac{10K}{10K} = 1$$

$$\boxed{V_{sal} = 1 \cdot (V_{ref} + V_{aud}) = (V_{ref} + V_{aud})}$$

Al circuito sumador se le agregó la resistencia R_7 para ajustar la ganancia.

4. Adquisición de la señal de Audio

La adquisición de señal se realizo con la placa de desarrollo **eZ430-RF2500** de Texas, la cual posee un microntrolador modelo **MSP430F2274**.

4.1. eZ430-RF2500

La placa **eZ430-RF2500** es una completa herramienta de desarrollo USB para evaluar el microntrolador **MSP430F2274** y el transmisor/receptor wireless CC2500. El **eZ430-RF2500** usa como IDEs al IAR Embedded Workbench o al Code Composer Essentials para escribir, cargar y depurar la aplicación. El depurador permite puntos de ruptura por hardware y paso a paso sin hardware adicional.

Características principales de la placa:

- Interface de programación y depuración.
- 21 pins disponibles para desarrollo.
- Completamente integrada con la familia de MCU MSP430 de 16 MHz.
- Dos LED de propósito general.
- Un push button para interacción con el usuario.

La figura 3 muestra una foto de esta placa.

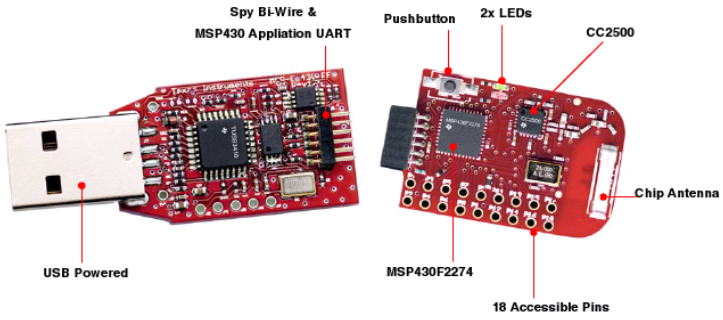

Figura 3: Foto de la placa de desarrollo **eZ430-RF2500**.

4.2. MSP430x2xx

4.2.1. Arquitectura

Los MCU de la familia MSP430x2xx incorporan una CPU RISC de 16-bit, periféricos y sistema muy flexible de clock que interconecta la memoria de direcciones y la de datos en una configuración von-Neumann.

Características principales de la familia MSP430x2xx:

- Arquitectura de ultra bajo consumo, extendiendo la vida de la batería.
  - $0{,}1\mu A$ retención de la **RAM**.
  - $0{,}8\mu A$ reloj en modo tiempo real.

  - $250\mu A$ MIPS activas.

- Conversor analógico de alta performance, ideal para mediciones de alta presición
  - Comparador con compuertas para mediciones de elementos resistivos.

- CPU RISC de 16 bits permitiendo aplicaciones con baja cantidad de líneas
  - Núcleo compacto reduciendo consumo de potencia y costo.
  - Optimizado para programación en alto nivel.
  - Sólo 27 instrucciones de núcleo y 7 modos de direccionamiento.
  - Capacidad de extensión de los vectores de interrupción.

- Flash programable en sistema, permitiendo cambios flexibles en el código.

4.2.2. Sistema Flexible de Clock

El sistema de clock está designado especificamente para aplicaciones con baterías como fuentes. Tiene un clock auxiliar de baja frecuencia (ACLK) manejado directamente desde un cristal de 32kHz. Este clock puede ser usado para despertar al micro para empezar a realizar operaciones en tiempo real. También tiene un reloj digitalmente controlado (DCO) el cual puede servir de fuente para los periféricos o como clock maestro (MCLK). Por diseño el DCO está activo en $2\mu s$ a 1 MHz.

4.2.3. Simulación embebida

El dispositivo cuenta con emulación lógica embebida en el mismo y accesible a través de JTAG sin usar recursos adicionales.

Los beneficios de la emulación embebida incluyen:

- No intrusiva: la depuración a alta velocidad de ejecución, puntos de ruptura y paso a paso son soportados.
- El desarrollo es en un circuito igual al dispositivo final.
- La integridad de señales está preservada y libre de interferencias por el cable.

4.2.4. Espacio de direcciones

La arquitectura von-Neumann del MSP430 tiene un espacio de direcciones mapeado junto con los registros de funciones especiales, periféricos, RAM y memoria Flash/ROM como se muestra en la figura 4. Los datos pueden ser accesados como bytes o palabras. La memoria accesible es de 128 KB.

4.2.5. Flash/ROM

La dirección de inicio de la Flash/ROM depende de la cantidad de Flash/ROM, y esta varía de dispositivo. La dirección final de esta es 0x1FFFF. Puede ser usada para código o datos. En esta memoria se pueden guardar tablas de bytes o palabras sin la necesidad de moverlas a la RAM antes de usarla.

La tabla de vectores de interrupción se mapean en las 16 palabras superiores de la Flash/ROM, con el vector de más peso en la dirección superior.

4.2.6. RAM

La RAM comienza en 0200h. El fin de la direccion de la RAM depende de la cantidad de RAM presente y del tipo de dispositivo. La RAM puede ser usada para código y datos.

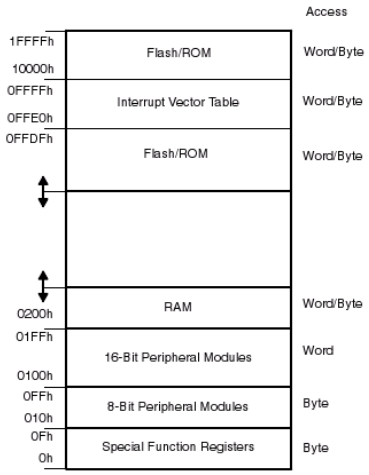

Figura 4: Mapa de memoria del **MSP430**.

4.2.7. Módulos Periféricos

Los módulos periféricos están mapeados en la espacio de memoria. El espacio de memoria desde 0100h hasta 01FFh está reservado para los módulos periféricos de 16 bits. Estos módulos deben ser accesados con instrucciones para palabras. Si son usadas instrucciones para bits, sólo las direcciones impares son accesibles y el byte más alto de la palabra es cero. Las direcciones desde 010h hasta 0FFh está reservada para los periféricos de 8 bits. Estos módulos deben ser accedidos con instrucciones de bytes.

4.2.8. Registros de Funciones Especiales

Algunos funciones de los periféricos se configuran en los registros de funciones especiales (SFR). Estos están ubicados en los 16 bytes inferiores del espacio de direcciones y están organizados por bytes. Los SFRs deben ser accedidos usando intrucciones de byte solamente.

4.2.9. Organización de la Memoria

Los bytes están localizados en direcciones pares o impares. Las palabras están sólo localizadas en direcciones impares como se muestra en la figura 5. . Cuando usamos instrucciones para palabras, sólo las

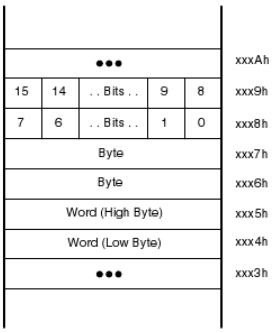

Figura 5: Organización de la memoria en bits, bytes y palabras.

direcciones impares son usadas. El byte más bajo de una palabra es siempre una dirección impar. El byte alto está siempre en la siguiente dirección par.

4.3. Código

El código a continuación muestra la parte relevante del programa, que se cargó al microntrolador, que se encarga de realizar la conversión de la señal de audio.

```
#include "msp430x22x4.h"
#define CONVERSIONES 8

unsigned int ADC[8]={0};
int resultado=0;
unsigned char incrementarPaquete=0, variable=0;

void main(void)
  {
  volatile unsigned int i;
  //---- TMR A3----------
  TACCR0 = 1233;
  TACTL = TASSEL_2 + MC_1 + ID_3 ;     // SMCLK, Up mode, Div por 8
  //-------- ADC -------
  ADC10CTL0 = ADC10ON +  ADC10IE + REFON + SREF_1;// ADC10ON, interrupt enable
  ADC10CTL0 = ADC10SHT_1 + MSC + REF2_5V;
  ADC10CTL1 = ADC10SSEL_3 + ADC10DIV_7 ;  // Repeat single channel, TA3 OUT1, SMCLK
  ADC10CTL1 = CONSEQ_2 + INCH_3 + SHS_1;
  ADC10DTC1 = CONVERSIONES;         // conversions
  // TIMER
  TACCTL1 = OUTMOD_4;              // Toggle on EQU1 (TAR = 0)
  TACCR1 = 1000;
  //PINES DE SALIDA
  P1DIR |= BIT0;                  // P1.0 led rojo salida
  P2DIR |= 0x01;                  // P2.0 CE
  P2DIR |= 0x02;                  // P2.1 CS
  P3DIR |= 0x01;                  // P3.0 CLK SALIDA
  P3DIR |= BIT4;                  // P2.1 DATA

  while (ADC10CTL1 & BUSY);         // Esperamos a que el ADC10 se active
  ADC10SA = (unsigned int)&ADC[0];     // Comienzo del buffer de datos
  ADC10CTL0 |= ENC + ADC10SC;         // Sampleo y comienzo de la conversion

  for (;;)                  // Bucle infinito
    {
    __bis_SR_register(CPUOFF + GIE);  // apagamos el CPU
    P1OUT ^= 0x01;                // prendemos el LED
    paquete[4]=  (ADC[0]>>2) & 0xff;  // llevamos de 10 a 8 bits
    paquete[5]=  (ADC[1]>>2) & 0xff;
    paquete[6]=  (ADC[2]>>2) & 0xff;
    paquete[7]=  (ADC[3]>>2) & 0xff;
    paquete[8]=  (ADC[4]>>2) & 0xff;
    paquete[9]=  (ADC[5]>>2) & 0xff;
    paquete[10]= (ADC[6]>>2) & 0xff;
    paquete[11]= (ADC[7]>>2) & 0xff;

    ADC10SA = (unsigned int)&ADC[0];  // Comienzo del buffer de datos
    ADC10CTL0 |= ENC + ADC10SC;      // Sampleo y comienzo de la conversion
    }
  }
/*-------------------------------------------------------------------
Características: rutina de servicio de interrupción del ADC10
```

```
------------------------------------------------------------------------------ */
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
  {
  __bic_SR_register_on_exit(CPUOFF);      //Prendemos el CPU
  }
```

Los pasos básicos de funcionamiento del **A/D** se muestran a continuación:

- Configurar el **A/D**
- Configurar el Timer
- Realizar la primera conversión
- Entrar en un bucle infinito
- Apagar el **CPU** dentro del for
- Leer los datos cuando termine la conversión
- Transformarlos de 10 a 8 bits
- Realizar una nueva conversión

El funcionamiento básico del programa del **A/D** es el siguiente: configuramos el **A/D** para que realice 8 conversiones consecutivas, las guarde en un arreglo de tipo int, que tome como fuente de clock la salida 3 del timer **A** y activamos la interrupción.

Luego de configurar el **A/D** pasamos a configurar el timer **A** para que su salida me de una temporización que haga comenzar la conversión del **A/D**. La temporización calculada es de aproximadamente $1233\mu s$ en la cual debemos realizar 8 conversiones. Esto nos da una frecuencia de muestreo de aproximadamente:

$$\frac{8}{1233}\frac{muestras}{\mu s} = 6488\frac{muestras}{s}$$

Realizamos la primera conversión y luego entramos en un bucle for y dentro de el apagamos la **CPU**. Cuando el microntrolador termina de realizar las 8 conversiones salta la interrupción del **A/D** y en ella prendemos la **CPU**.

Al retornar al for leemos los datos de la conversión y los dividimos por 4 para pasarlos de 10 a 8 bits. Esto lo hacemos porque sino deberíamos mandar 2 bytes por conversión degradando mucho el ancho de banda del sistema final.

Una vez transformados los datos lanzamos una nueva conversión y comienza nuevamente el bucle for.

5. Transmisión vía RF de los datos

La transmisión de los datos en una configuración Wireless (sin cable) se realizó con una placa transmisora/receptora de GFSK. La placa en cuestión es la TRW-2.4GHz Radio Transceiver marca WENSHING y se muestra en la figura 6

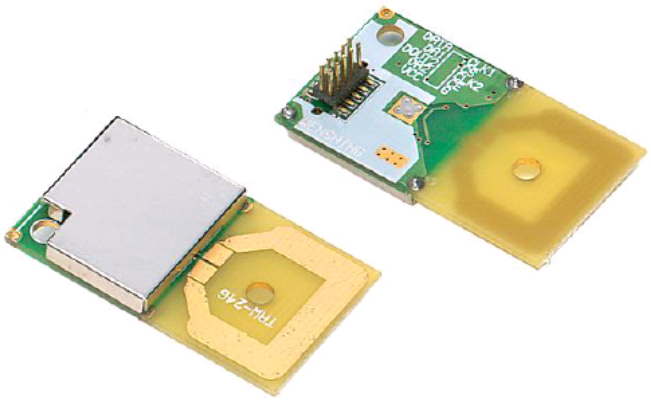

Figura 6: Imagen del módulo TRW24G.

La modulación GFSK es una modulación digital en donde se tienen dos portadoras y según la señal sea cero o uno se manda una u otra frecuencia. Se la conoce como gaussiana porque la señal modulante es pasada por un filtro pasabajos para reducir su ancho de banda, reduciendo con esto el ancho de banda de la señal modulada.

5.1. TRW24G

Las características principales de la placa son:

- Rango de frecuencia de 2.4 a 2.527GHz.
- Modulación GFSK.
- Voltage de trabajo de 3 V.
- 128 canales.
- Potencia de salida de 0 dBm.
- Tasa de transmisión de 250Kbps a 1Mbps.
- Temperatura de operación de -40 a 85ºC.
- Rango de 280 m (250Kbps) a 150 m (1Mbps).
- Sin espacios muertos en la recepción.
- Antena incluída.
- Precio competitivo.

Las aplicaciones de este tipo de módulos pueden ser joysticks, parlantes, auriculares, comunicaciones, mouses, teclados y comunicaciones de datos.

El módulo tiene dos modos de funcionamiento, DirectMode y ShockBurst. El modo directo no se explicará porque no fue usado para este proyecto.

5.2. Transmisión en modo ShockBurst

Esta tecnología usa un método FIFO para leer los datos a baja velocidad y luego transmitirlos a altas tasas de velocidad, permitiendo una gran reducción en la potencia consumida. Cuando el módulo trabaja en el modo ShockBurst, se tienen las más altas tasas de transmisión de datos (1Mbps) sin la necesidad de tener un MCU que soporte esa velocidad.

El TRW24G ofrece los siguientes beneficios:

- Alta reducción del consumo de potencia.
- Reducción del costo del sistema.
- Gran reducción del riesgo de colisiones en el aire debido a la corta duración de la transmisión.

El TRW24G puede ser programado usando una simple interfaz de 3 cables donde la tasa de transmisión esta dada por la velocidad del MCU.

5.2.1. Principio

Cuando el TRW24G es configurado en el modo ShockBurst, la operación en TX o RX es conducida de la siguiente manera (10 Kbps en el ejemplo siguiente).
En la figura 7 y 8 se muestra un ejemplo de enlace, para poder analizar el consumo de potencia con

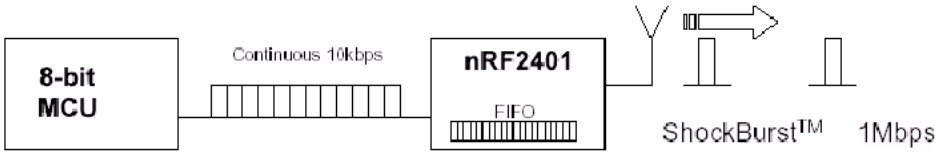

Figura 7: Transmisión de los datos desde el MCU a 10 Kbps.

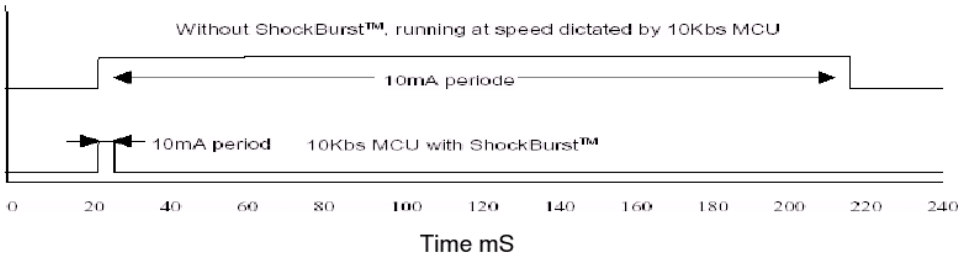

Figura 8: Consumo de corriente con y sin tecnología ShockBurst.

y sin tecnología ShockBurst.

5.2.2. Secuencia de Transmisión

La figura 9 muestra un diagrama de flujo de la transmisión en el modo ShockBurst.

Los pasos que realiza el transmisor son:

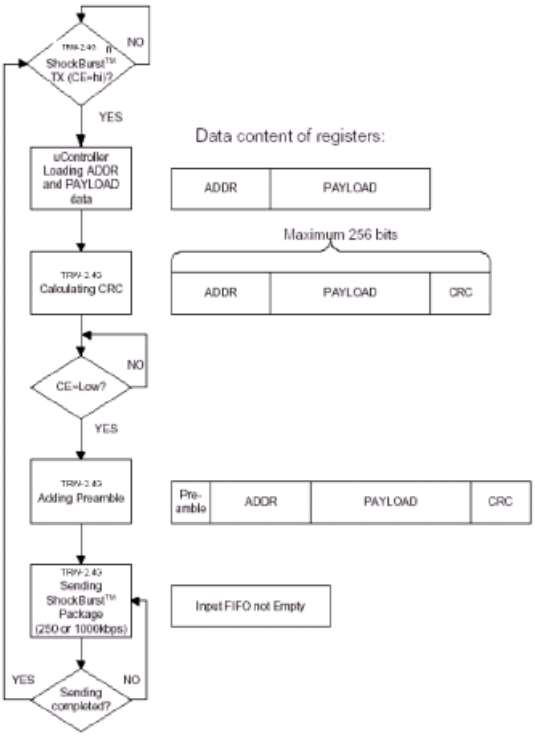

Figura 9: Diagrama de Flujo de transmisión en modo ShockBurst.

1. Cuando el microntrolador tiene datos para mandar pone a un CE. Esto activa la placa de procesamiento de datos del TRW-24G.
2. La dirección del nodo de recepción (**RX**) y los datos son clockeados al TRW-24G. La velocidad de transmisión está seteada por el **MCU** y debe ser menor a 1Mbps.
3. El MCU pone a bajo CE, esto activa la transmisión en ShockBurst.
4. TRW-24G ShockBurstTM
   - Se enciende el módulo de RF.
   - Se calcula el CRC y se agrega el preámbulo.
   - Los datos son transmitidos a alta velocidad (1Mbps o 250Kbps).
   - El TRW-24G regresa al estado bajo consumo cuando termina.

5.3. Configuración

Para configurar al dispositivo se debe usar un protocolo SPI de 2 líneas (**CLK** + Datos) y enviarle una palabra de configuración de 120 bits (15 bytes). La figura 10 muestra un resumen de la configuración del TRW-24G.

	Bit position	Number of bits	Name	Function
ShockBurst™ configuration	143:120	24	TEST	Reserved for testing
	119:112	8	DATA2_W	Length of data payload section RX channel 2
	111:104	8	DATA1_W	Length of data payload section RX channel 1
	103:64	40	ADDR2	Up to 5 byte address for RX channel 2
	63:24	40	ADDR1	Up to 5 byte address for RX channel 1
	23:18	6	ADDR_W	Number of address bits (both RX channels).
	17	1	CRC_L	8 or 16 bit CRC
	16	1	CRC_EN	Enable on-chip CRC generation/checking.
General device configuration	15	1	RX2_EN	Enable two channel receive mode
	14	1	CM	Communication mode (Direct or ShockBurst™)
	13	1	RFDR_SB	RF data rate (1Mbps requires 16MHz crystal)
	12:10	3	XO_F	Crystal frequency
	9:8	2	RF_PWR	RF output power
	7:1	7	RF_CH#	Frequency channel
	0	1	RXEN	RX or TX operation

Figura 10: Tabla de palabras de configuración.

5.3.1. Modo ShockBurst

La palabra de configuración en modo ShockBurst permite al TRW24G manejar el protocolo de RF. Una vez que el protocolo está completo y cargado, sólo es necesario un byte (bit [7:0]) para manejar el funcionamiento durante la operación actual.

Los bloques de configuración dedicados al ShockBurst son los siguientes:

- Sección del ancho del paquete:*Especifica el número de bits de carga en el paquete de RF. Esto permite distinguir entre la carga útil y los bits de CRC en el paquete recibido.*
- Ancho de la dirección:*Setea el número de bits usados para las direcciones en el paquete recibido, esto permite distinguir entre la carga útil y los bits de direcciones.*
- Direcciones:*Direcciones de los canales 1 y 2 del dispositivo.*
- CRC:*Habilita la generación y comprobación del CRC para los paquetes enviados y recibidos respectivamente.*

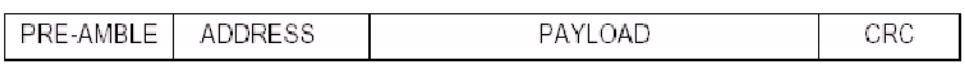

Figura 11: Ejemplo de paquete completo en modo ShockBurst.

La palabra de configuración es desplazada de tal manera que el MSB sea el primer bit enviado y este es tomado en los flancos de subida del clock. La nueva configuración es tomada en el flanco de caída de

CS.

La palabra de configuración para el transmisor fue la siguiente:

0x00,0x40,0x00,0x00,0x00,0x00,0x0F,0x00,0xAA,0xCC,0xBB,0xAA,0x80,0x4F,0x08

Las características principales de la configuración usada para el proyecto son:

- Como transmisor
- 8 bytes de datos útiles
- direccion = 0xAA,0xBB,0xCC,0xAA
- 1 Mbps
- $P_{out} = 0Dbm$
- CRC desactivado
- Modo ShockBurst

El dispositivo fue configurado con el microntrolador de la placa **eZ430-RF2500**, el cual también fue usado para samplear la señal del audio. Este microntrolador tiene protocolo **SPI** incluído, pero este no funciono con el protocolo del módulo de **RF**, por lo tanto debió ser implementado con 2 pines de **I/O** del micro.

5.4. Código

A continuación se muestra el código de configuración del TRW-24G como transmisor y las rutinas para enviar datos a través del mismo:

```
#include "msp430x22x4.h"

#define ENVIAR_BIT       N_CLK if( x & 0x80 ) DATA_OUT else N_DATA_OUT CLK x <<= 1;
#define CE     P2OUT |= 0x01;                                  // P2.0 output
#define N_CE   P2OUT &= ~0x01;                                  // P2.0 output

#define CS     P2OUT |= 0x02 ;                                 // P2.1 output
#define N_CS   P2OUT &= ~0x02 ;                                 // P2.1 output

#define DR_1   P2IN & BIT3                                    //Ledtura de P2.2 (INPUT)

#define DATA_IN   P3IN & BIT4

#define DATA_OUT    P3OUT |= BIT4 ;
#define N_DATA_OUT P3OUT &= ~BIT4;

#define CLK     P3OUT |= 0x01 ;                                 // P3.0 output HI
#define N_CLK   P3OUT &= ~0x01;                                 // P3.0 output LO

#define LONGITUD 12

#define CONVERSIONES 8

void escribirPaqueteTRW24G( unsigned char * paq, unsigned int tamanio);
void Config_TRW_24G(void);
char Read_TRW_24G_BYTE(void);
unsigned char Receive_TRW_24G(void);

unsigned char transmisor[] = {0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x00
                          , 0xAA, 0xCC, 0xBB, 0xAA, 0x80, 0x4F, 0x08};
```

```
unsigned char paquete[]={0xAA, 0xCC, 0xBB, 0xAA, 0x99, 0x00, 0x01
                         , 0x02, 0x03, 0x04, 0x05, 0xAA};

unsigned int ADC[8]={0};

int resultado=0;
unsigned char incrementarPaquete=0, variable=0;

void main(void)
  {
  //PINES DE SALIDA
  P1DIR |= BIT0;              // P1.0 led rojo salida
  P2DIR |= 0x01;              // P2.0 CE
  P2DIR |= 0x02;              // P2.1 CS
  P3DIR |= 0x01;              // P3.0 CLK SALIDA
  P3DIR |= BIT4;              // P2.1 DATA

  Config_TRW_24G();

  for(i=0;i<0xfff;i++)
    continue;

  for (;;)
    {
    __bis_SR_register(CPUOFF + GIE);          // Apagamos el CPU
    P1OUT ^= 0x01;                            // Prendemos el Led
    paquete[4]=   (ADC[0]>>2) & 0xff;
    paquete[5]=   (ADC[1]>>2) & 0xff;
    paquete[6]=   (ADC[2]>>2) & 0xff;
    paquete[7]=   (ADC[3]>>2) & 0xff;
    paquete[8]=   (ADC[4]>>2) & 0xff;
    paquete[9]=   (ADC[5]>>2) & 0xff;
    paquete[10]=  (ADC[6]>>2) & 0xff;
    paquete[11]=  (ADC[7]>>2) & 0xff;
    CE
    escribirPaqueteTRW24G(paquete,LONGITUD);
    N_CE
    }
  }

/* ------------------------------------------------------------------
Características: configuración del TRW-24G
------------------------------------------------------------------ */
void Config_TRW_24G(void)
  {
  N_CE
  CS
  escribirPaqueteTRW24G(transmisor,sizeof(transmisor));
  N_CS
  }

/*------------------------------------------------------------------
Características: escribir un paquete en el TRW-24G
------------------------------------------------------------------ */
void escribirPaqueteTRW24G( unsigned char * paq, unsigned int tamanio)
  {
  P1OUT ^= 0x01;          // Toggle P1.0
  char payload,x;

  N_CLK            // para que la pata este siempre a cero, clock
  N_DATA_OUT           // para que la pata este siempre a cero, datos
```

```
  for(payload=0;payload<tamanio;payload++) {
    x=paq[payload];
    ENVIAR_BIT // BIT 0
    ENVIAR_BIT // BIT 1
    ENVIAR_BIT // BIT 2
    ENVIAR_BIT // BIT 3
    ENVIAR_BIT // BIT 4
    ENVIAR_BIT // BIT 5
    ENVIAR_BIT // BIT 6
    ENVIAR_BIT // BIT 7

    N_CLK
  }
}

/*-------------------------------------------------------------------
Características: rutina de servicio de interrupción del ADC10
-------------------------------------------------------------------*/

#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);          // Prendemos el CPU
}
```

El programa comienza configurando al módulo como transmisor (Config_TRW_24G), luego lee los valores de la conversión del A/D (ADC[]), los transforma a 8 bits y los envía por SPI (escribirPaqueteTRW24G). Podemos ver que el **#define ENVIAR_BIT** es el encargado de escribir un bit con el protocolo SPI. También podemos ver que el paquete que enviamos está formado por la dirección más los datos útiles, un total de 12 bytes (4 de dirección más 8 de datos).

5.5. Circuito

En la figura 12 se muestra la interfaz entre el MCU y el Transmisor TRW24G.

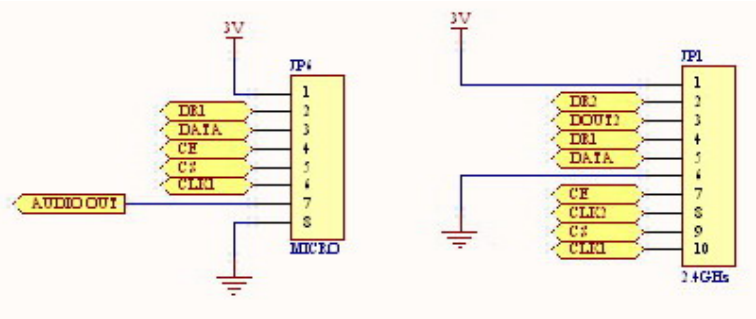

Figura 12: Circuito de interfaz entre el MCU y el Transmisor.

6. Recepción vía RF de los datos

La placa usada para la recepción es la misma que la usada para la transmisión, salvo que configurada como receptora.

6.1. Recepción en modo ShockBurst

La figura 13 muestra un diagrama de flujo de la recepción en el modo ShockBurst.

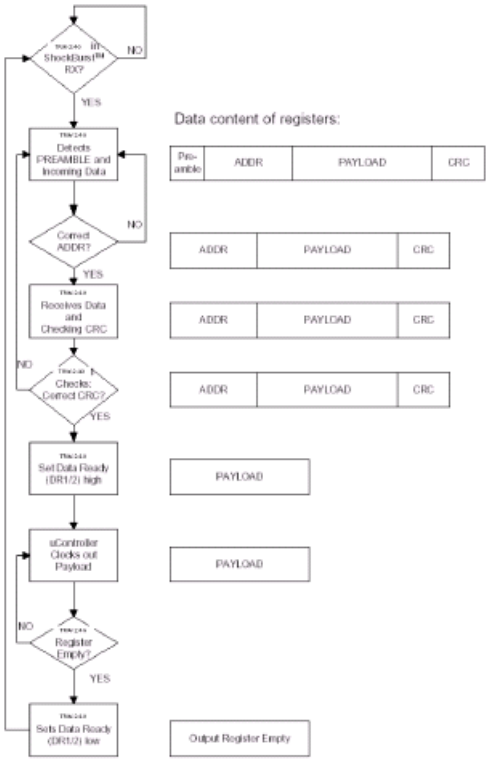

Figura 13: Diagrama de Flujo de recepción en modo ShockBurst.

Los pasos que realiza el receptor son:

1. El paquete a recibir debe tener la dirección correcta y el tamaño de datos correcto.
2. Para activar el receptor se debe poner a uno CE.
3. Después de 200 μs de estabilización, el TRW-24G empieza a monitorear el aire esperando por un paquete.
4. Cuando un paquete válido se ha recibido (dirección , CRC y datos correctos), el TRW-24G remueve el preámbulo, la dirección y el CRC.
5. El TRW-24G notifica al MCU seteando DR1.
6. El MCU puede o no bajar la pata de CE para desactivar la receptción y reducir el consumo.
7. El MCU pedirá los datos recibidos a través del SPI a una tasa menor a 1Mbps.

8. Cuando todos los datos fueron entregados, el TRW-24G pone DR1 a cero y el receptor está listo para otra recepción.

El protocolo de configuración del receptor es el mismo que el del transmisor. La palabra de configuración para el receptor fue la siguiente:

0x00,0x40,0x00,0x00,0x00,0x00,0x0F,0x00,0xAA,0xCC,0xBB,0xAA,0x80,0x4F,0x09

Las características principales de la configuración son:

- Como receptor
- 8 bytes de datos útiles
- direccion = 0xAA,0xBB,0xCC,0xAA
- 1 Mbps
- $P_{out} = 0Dbm$
- CRC desactivado
- Modo ShockBurst

6.2. Código

A continuación se muestra el código de configuración del TRW-24G como receptor y las rutinas para recibir datos a través del mismo:

```
#include  "msp430x22x4.h"
//  DEFINICIONES
#define CE     P2OUT |= 0x01;          // P2.0 output
#define N_CE   P2OUT &= ~0x01;         // P2.0 output

#define CS     P2OUT |= 0x02;          // P2.1 output
#define N_CS   P2OUT &= ~0x02;         // P2.1 output

#define DR_1   P2IN & BIT2           // Lectura de P2.2 (INPUT)

#define DATA_IN   P3IN & BIT1          // P3.1 de la placa, es el DATA del receptor

#define DATA_OUT    P3OUT |= BIT1 ;
#define N_DATA_OUT P3OUT &= ~BIT1;

#define CLK     P3OUT |= 0x01;         // P3.0 output HI
#define N_CLK   P3OUT &= ~0x01;          // P3.0 output LO

#define LONGITUD 8
//  FUNCIONES
void Write_TRW_24G_BYTE(unsigned char x);
void Config_TRW_24G(void);
char Read_TRW_24G_BYTE(void);
unsigned char *Receive_TRW_24G(void);
void Send_TRW_24G( unsigned char * paq);
//  VARIABLES
void parpadear_led(char led);
char a=0,enc_apa=0,msg[2];
int resultado=0;

unsigned char receptor[] = {0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x00
```

```
                        , 0xAA, 0xCC, 0xBB, 0xAA, 0x80, 0x4F, 0x09};

volatile unsigned int i;
unsigned char RF_Data[30];
unsigned char dir=0;
//  Main
void main(void)
  {
  //PINES DE SALIDA
  P1DIR |= 0x01;            // P1.0 led
  P2DIR |= 0x01;            // P2.0 CE
  P2DIR |= 0x02;            // P2.1 CS
  P2DIR &=~ BIT2;           // P2.3 DR1 como entrada
  P3DIR |= 0x01;            // P3.0 CLK SALIDA
  P1OUT = 0x01;
  P1DIR |= 0x01;            // colocamos P1.0 como Salida (Led rojo)

  Config_TRW_24G();

  while(1)
    {
    if(DR_1)           // si data ready esta a uno
      {
      Receive_TRW_24G();
      P1OUT ^= 0x01;        // Toggle P1.0
      }
    }
  }

/* ------------------------------------------------------------------------
Características: configuración del TRW-24G
------------------------------------------------------------------------ */

void Config_TRW_24G(void)
  {
  unsigned char i;
  N_CE
  CS

  for(i=0;i<sizeof(receptor);i++)
    Write_TRW_24G_BYTE(receptor[i]);

  N_CS
  CE
  }

/* ------------------------------------------------------------------------
Características: escribir un byte al TRW-24G
------------------------------------------------------------------------ */

void Write_TRW_24G_BYTE(unsigned char x)
  {
  char i;

  N_CLK              // para que la pata este siempre a cero, clock
  P3DIR |= 0x01;           // P3.0 CLK SALIDA
  N_DATA_OUT             // para que la pata este siempre a cero, datos
  P3DIR |= BIT1;           // P3.1 DATA

  for(i=0;i<8;i++)
    {
    N_CLK
```

```
    if( x & 0x80 )
      DATA_OUT
    else
      N_DATA_OUT

    x <<= 1;
    CLK
    CLK
    CLK
    }
  N_CLK
  P3DIR &= ~BIT1;           // P2.1 DATA
  }

/* ---------------------------------------------------------------------
Características: leer un byte del TRW-24G
--------------------------------------------------------------------- */

char Read_TRW_24G_BYTE(void)
  {
  char i,x;
  P3DIR &=~ BIT1;           // P3.1 DATA
  P3DIR |= 0x01;            // P3.0 CLK SALIDA
  for(i=0;i<8;i++)
    {
    N_CLK
    N_CLK
    CLK
    x<<=1;
    if(DATA_IN)
      x|=0x01;
    else
      x|=0x00;
    }
  N_CLK
  return(x);
  }


/* ---------------------------------------------------------------------
Características: leer una cadena TRW-24G
--------------------------------------------------------------------- */

unsigned char * Receive_TRW_24G(void)
  {
  N_CLK
  dir=0;
  while(DR_1)
    {
    RF_Data[dir] = Read_TRW_24G_BYTE();
    dir++;
    }
  CE
  return RF_Data;
  }
```

El programa comienza configurando al módulo como receptor (Config_TRW_24G), luego entra en un while infinito, en el que pregunta si DR1 (Data Ready 1) está a uno, si esto es así, significa que hay datos listos para ser leídos. Esto lo hacemos con el método Receive_TRW_24G, el cual lee una cadena de datos byte a byte con la función Read_TRW_24G_BYTE.

6.3. Circuito

En la figura 14 se muestra el circuito del receptor.

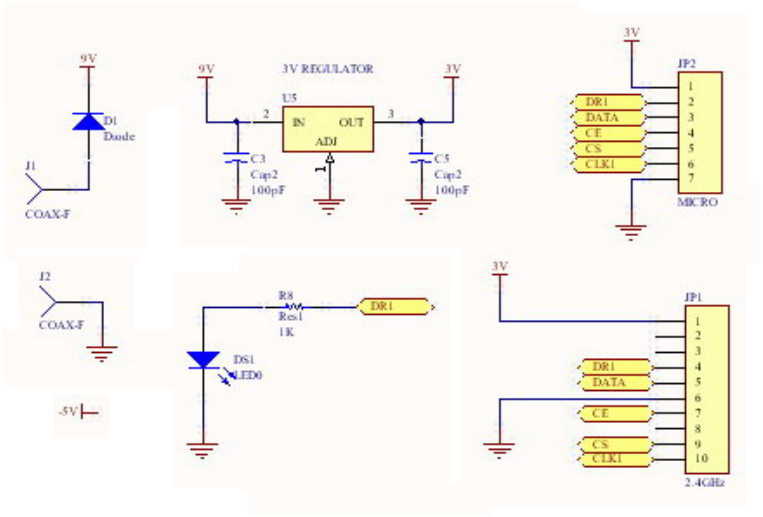

Figura 14: Circuito de Receptor.

7. Transmisión de los datos a la PC a través del puerto USB

Como el microntrolador elegido no tiene **USB** incorporado, se utilizo el chip FT232BQ de la empresa FTDI el cual nos permite transformar el protocolo serie a **USB**.

7.1. FT232BQ

7.1.1. Características

- Hardware
  - Transferencia de datos serial asincrónico USB en un solo chip.
  - Handshaking completo y señales para interfaz Modem.
  - Soporte de UART con 7/8 bits de datos, 1/2 bits de parada y paridad Par/Impar/Marca/Espacio/Sin paridad.
  - Tasa de Transferencia
    - 3 M (TTL)
    - 1 M (RS232)
    - 3 M (RS422/RS485)
  - Buffer de recepción de 348 bytes y de transmisión de 128 bytes.
  - Soporte en chip para carácteres de evento y condición de rotura de línea.
  - Soporte USB para suspensión y continuación.
  - Conversor integrado de nivel sobre la UART y control de las señales para interfaz con lógica de 5V y 3.3V.
  - Multiplicador integrado de 6 Mhz a 48 Mhz.
  - Transferencia USB Bulk o asincrónica.
  - Operación con alimentación única entre 4.35V a 5.25V.
  - Compatible con controlador de Host UHCI/OHCI/EHCI.
  - Compatible con USB 1.0 y 2.0.
  - USB VID, PID, número de serie y descripción de producto en una EEPROM externa.
  - EEPROM programable onboard a través del USB.
- Drivers de puerto COM virtual para:
  - Windows 98 y Windows 98 SE.
  - Windows 2000/ME/Server 2003/XP.
  - Windows XP 64 Bit.
  - Windows XP Embedded.
  - Windows CE 4.2.
  - MAC OS-8 and OS-9.
  - MAC OS-X.
  - Linux 2.40 y superiores.
- D2XX (Drivers directos para USB + DLL S/W Interface)
  - Windows 98 and Windows 98 SE.
  - Windows 2000/ME/Server 2003/XP.
  - Windows XP 64 Bit.

- Windows XP Embedded.
- Windows CE 4.2.
- Linux 2.40 and greater.

▪ Áreas de aplicación

- Conversores RS232 a USB.
- Conversores RS422/RS485 a USB.
- Actualización de dispositivos RS232 a USB.
- Transferencia de datos para celulares.
- Interface USB para MCUs.
- Transferencia de audio y video de baja resolución.
- Transferencia USB para PDAs.
- Lectores de USBs Smart Cards.
- Modems USB.
- Instrumentación USB.
- Lectores de barras USB.

7.1.2. Diagrama en Bloques

El la figura 15 se muestra el diagrama en bloques interno de este chip y en la figura 16 se muestra

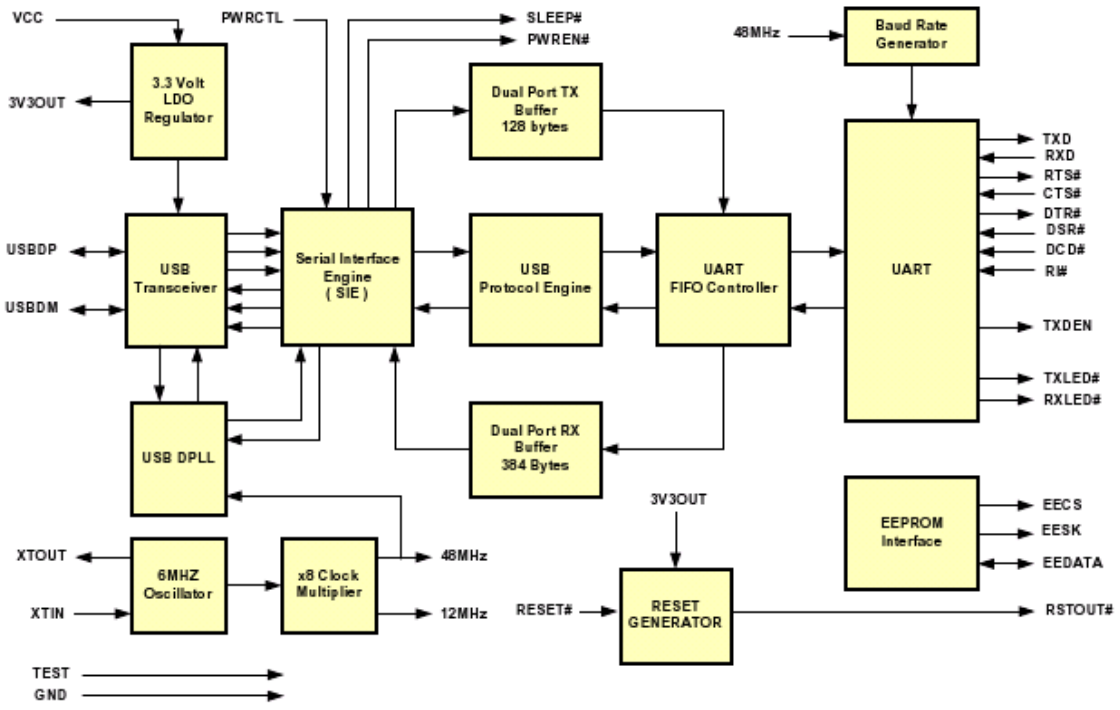

Figura 15: Diagrama en bloques interno simplificado del FT232BQ.

el símbolo esquemático del mismo.

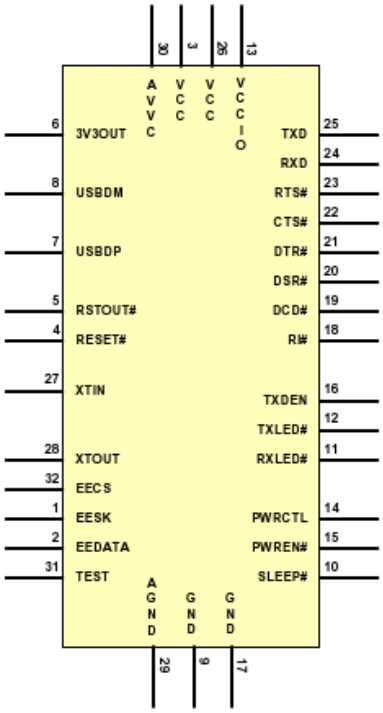

Figura 16: Diagrama esquemático del FT232BQ.

7.2. USCI en modo UART

La interface de comunicaciones serie universal (USCI) soporta múltiples modos de comunicaciones serie con un solo módulo hardware.

El microntrolador se programó para que envie datos a través de una de sus **USCIs** configurada en modo **UART**, dejando que el chip FTDI se encargué del protocolo **USB**.

7.2.1. USCI Overview

La interface de comunicaciones serie universal (USCI) soporta múltiples modos de comunicaciones serie. Las diferentes módulos USCIs soportan diferentes modos.

Cada módulo diferente USCI es nombrado con una letra diferente. Por ejemplo, la USCI_A es diferente de la USCI_B.

Si más de un módulo USCI idéntico está implementado en un dispositivo, estos son nombrados con números que se incrementan. Por ejemplo si un dispositivo tiene dos módulos USCI_A, estos son llamados USCI_A0 y USCI_A1.

La USCI_Ax soporta:

- Modo UART.
- Comunicaciones IrDA.
- Detección automática de tasa de baudios para comunicaciones LIN.
- Modo SPI.

La USCI_Bx soporta:

- Modo I2C.
- Modo SPI.

7.2.2. Introducción a la USCI: modo UART

En el modo asincrónico, el módulo USCI_Ax conecta al MSP430 a un sistema externo a través de dos pines, UCAxRXD y UCAxTXD. El modo UART es seleccionado cuando el bit UCSYNC está en cero.

Las características del modo UART incluyen:

- 7 o 8 bits de datos con paridad par, impar o sin paridad.
- Registros de corrimiento de transmisión y recepción independientes.
- Transmisión y recepción en modo LSB o MSB.
- Detección de comienzo de recepción para despertar al módulo en los modos de bajo consumo.
- Velocidad de transmisión programable con modulación para soporte de velocidades fraccionales.
- Bandera de estado para detección y supresión de errores.
- Bandera para detección de dirección.
- Capacidad de interrupción independiente para transmisión y recepción.

La figura 17 muestra al modulo USCI_Ax cuando está configurado en el modo UART.

7.2.3. Operación de la USCI en modo UART

En el modo UART, la USCI transmite y recibe carácteres a una tasa determinada de bits en forma asincrónica con otro dispositivo. El tiempo para cada carácter está basado en el tasa de baudios seleccionado para la USCI. Las funciones de transmisión usan la misma tasa de baudios para la transmisión y recepción.

Inicialización y Reset de la USCI: la USCI es reseteada por un PUC o seteando el bit UCSWRST. Después de un PUC, el bit de UCSWRST es automáticamente seteado, manteniendo la USCI en una condición de reset.

Cuando el bit UCSWRST está seteado, este resetea los bits UCAxRXIE, UCAxTXIE, UCAxRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE y UCBTOE, y setea el bit UCAxTXIFG. Poniendo a cero UCSWRST se vuelve a configurar la USCI para la operación.

El proceso recomendado para inicializar la USCI o reconfigurarla es:

1. Setear UCSWRST.
2. Inicializar todos los registros de la USCI con $UCSWRST = 1$.
3. Configurar los puetos (pines).
4. Poner a cero el bit UCSWRST a través de software.
5. Habilitar la interrupción (opcional) a través de UCAxRXIE o UCAxTXIE.

Formato de los carácteres: el formato de los carácteres para la UART, mostrados en la figura 18 , consisten en un bit de start, siete u ocho bits de datos, un bit o ninguno de paridad, un bit de dirección (en modo con bit de dirección) y uno o dos bits de stop. El bit UCMSB controla la dirección de la transferencia, seleccionando el formato LSB o MSB. El modo LSB-primero es típicamente requerido para comunicación en modo UART.

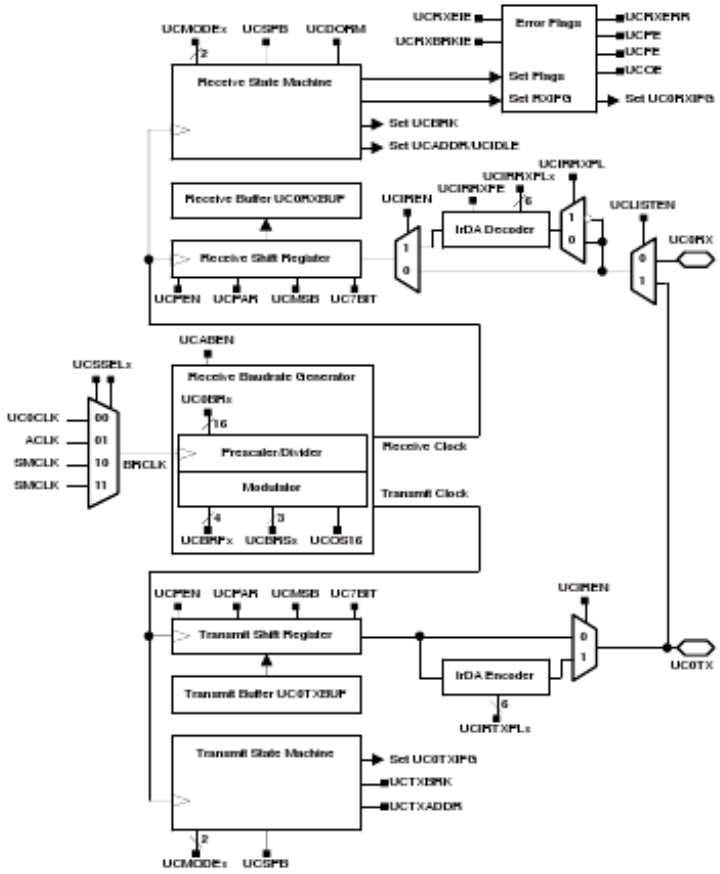

Figura 17: Diagrama en bloques de la USCI_Ax en modo UART (UCSYNC = 0).

Generación de la velocidad de Baudios: La USCI es capaz de producir velocidades estandars desde fuentes no estandars. Este provee dos modos de operación seleccionado desde el bit UCOS16.

Generación de bajas velocidades: el modo de generación en bajas velocidades es elegido cuando $UCOS16 = 0$. Este modo permite generar bajas velocidades desde relojes de baja frecuencia. Usando bajas frecuencias el consumo del módulo es reducido.

Generación de velocidades en forma SobreSampleada: el modo de sobre sampleo es seleccionado cuando $UCOS16 = 1$. Este modo soporta velocidades mayores que las fuentes de clock con las que se generan estas velocidades. Este modo permite fácilmente el uso del protocolo IrDA.

Seteando una tasa de Baudios: Para una fuente de clock dada para BRCLK, la tasa de baudios usada determina la división requerida por un factor de N:

$$N = \frac{f_{BRCLK}}{TasadeBaudios}$$

El factor de división N es a menudo un valor no entero, por esto como mínimo es necesario un divisor y un modulador para generar una frecuencia cercana a la requerida.

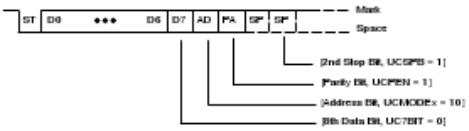

Figura 18: Formato de carácteres de la UART.

7.3. Código

La UART usada para realizar la comunicación con el FTDI es al USCI_A0. El clock se lo configuro en modo sobre sampleo a una velocidad de 460800 baudios para un reloj de 16Mhz. Esta velocidad se calculó usando las tablas de la hoja de datos del microntrolador. El formato de los datos fue de 8 bits, 1 bit de stop y sin paridad.

A continuación se muestra el código del microntrolador para manejar la **UART**.

```
#include  "msp430x22x4.h"
//  FUNCIONES
void TXString( unsigned char* string, int length );
//  VARIABLES
void parpadear_led(char led);
char a=0,enc_apa=0,msg[2];
int resultado=0;
unsigned char receptor[] = {0x00, 0x40, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x00
                          , 0xAA, 0xCC, 0xBB, 0xAA, 0x80, 0x4F, 0x09};
volatile unsigned int i;
unsigned char RF_Data[30];
unsigned char dir=0;
//  Main
void main(void)
  {
  // Configuracion de la UART
  P3SEL = 0x30                          // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSSEL_2                      // SMCLK

  //Valores de tabla para 460800 a 16 Mhz de CLK
  UCA0MCTL = UCBRF_2| UCBRS_3| UCOS16;
  UCA0BR0 = 2;                          // 16MHz 460800
  UCA0BR1 = 0;                          // 16MHz 460800
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**

  P1OUT = 0x01;
  P1DIR |= 0x01;                          // colocamos P1.0 como Salida (Led rojo)
  while(1)
    {
    if(DR_1)
      {
      TXString( Receive_TRW_24G(), LONGITUD );
      P1OUT ^= 0x01;                        // Toggle P1.0
      }
    }
  }

/* ------------------------------------------------------------------
Features: enviar una cadena por UART a la PC
------------------------------------------------------------------ */

void TXString( unsigned char* string, int length )
  {
```

```
int pointer;
for( pointer = 0; pointer < length; pointer++)
  {
  volatile int i;
  UCA0TXBUF = string[pointer];
  while (!(IFG2&UCA0TXIFG));                // USCI_A0 TX buffer ready?
  }
}
```

7.4. Circuito

La configuración circuital usada fue la de 3 volt Self Powered la cual nos permite tener dos alimentaciones independientes, una de 5 V para la interfaz **USB** y otra de 3 V para la interfaz con la **UART**.

Esta configuración fue sacada de la guía de diseño del FT232BQ, la cual se muestra en la figura 19 .

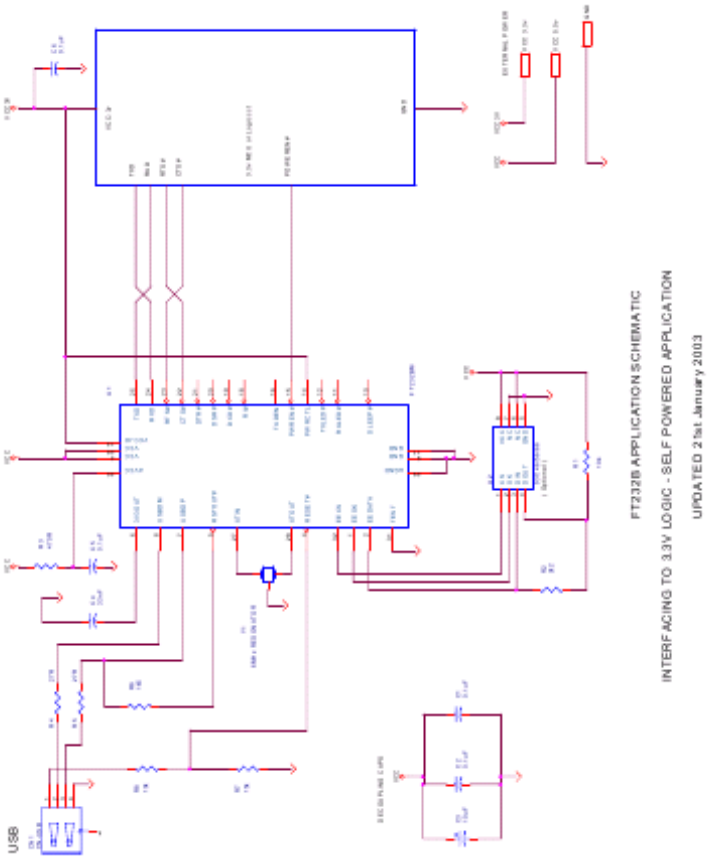

Figura 19: Circuito del FTDI.

.

8. Programa de bajo nivel que lee los datos a través del USB

Se decidió que el software de la **PC** corra sobre un sistema operativo **GNU/Linux**, por lo tanto la librería que se uso para leer el FTDI fue la libftdi.

Estas librerías poseen todas las funciones para poder trabajar con estos chips, también vienen con documentación y ejemplos, esto hace que uso sea muy fácil.

8.1. Código

A continuación mostramos el programa que maneja el FTDI desde la PC.

```
#include <stdio.h>
#include <ftdi.h>
#include <time.h>
#include <sys/time.h>

int configurarFTDI( struct ftdi_context *ftdi, int baudrate
                          , int bits, int sbit, int parity );

int main(int argc, char **argv) {
    struct ftdi_context ftdic;
  ftdi_init(&ftdic);

  FILE *archivo = fopen("archivo.raw","w+");

  int i = 0, error,contador=0;
  unsigned char buf[10],comparador;

  if( configurarFTDI( &ftdic, 460800, BITS_8, STOP_BIT_1, NONE ) < 0 ) {
        return EXIT_FAILURE;
  }

  error = -1;
  while( 1 ) {
    if((error = ftdi_read_data( &ftdic, buf, 1 )) < 0) {
      fprintf(stderr, "unable to leer un byte: %d (%s)\n"
                          , error, ftdi_get_error_string(&ftdic));

      return EXIT_FAILURE;
    }
    else if ( error > 0 ) {
      fprintf( archivo,"%c", buf[0] );
    }
    error = -1;
  }

  ftdi_usb_close(&ftdic);
  ftdi_deinit(&ftdic);

  return EXIT_SUCCESS;
}

int configurarFTDI( struct ftdi_context *ftdi, int baudrate
                          , int bits, int sbit, int parity ) {

  int retorno = 1;
    if((retorno = ftdi_usb_open( ftdi, 0x0403, 0x6001)) < 0) {
        fprintf(stderr, "unable to open ftdi device: %d (%s)\n"
                          , retorno, ftdi_get_error_string(&ftdi));

        return EXIT_FAILURE;
```

```
    }
  retorno = 1;
    if((retorno = ftdi_set_baudrate( ftdi, baudrate )) < 0) {
        fprintf(stderr, "unable to setear el baudrate: %d (%s)\n"
                        , retorno, ftdi_get_error_string(&ftdi));

        return EXIT_FAILURE;
    }
  retorno = 1;
    if((retorno = ftdi_set_line_property( ftdi, bits, sbit, parity )) < 0) {
        fprintf(stderr, "unable to setear las propiedades de linea: %d (%s)\n"
                        , retorno, ftdi_get_error_string(&ftdi));

        return EXIT_FAILURE;
    }
  return retorno;
}
```

Lo que hace el programa aquí mostrado es llamar a la función configurarFTDI, la cual abre el **USB**, setea las propiedades del puerto serie que simulará el FTDI (baudrate, bits de datos, bits de stop, paridad). Luego entra en un while infinito. En este último leemos los datos del puerto serie y los guardamos en un archivo.

9. Interfaz de usuario (GUI)

Para la interfaz gráfica se usaron las librerías GTK+[3] y como lenguaje de programación python[4]. El programa consta de varios archivos:

9.1. ventanaPrincipal.glade

Es un xml que genera la parte gráfica, solamente los botones, etiquetas, etc. sin acción alguna. Esto se hace en el programa llamado glade[5] que nos permite obtener un acabado gráfico desarrollado y nos permite desconcentrarnos de la complejidad gráfica y concentrarnos en lo que debe realizar el programa.

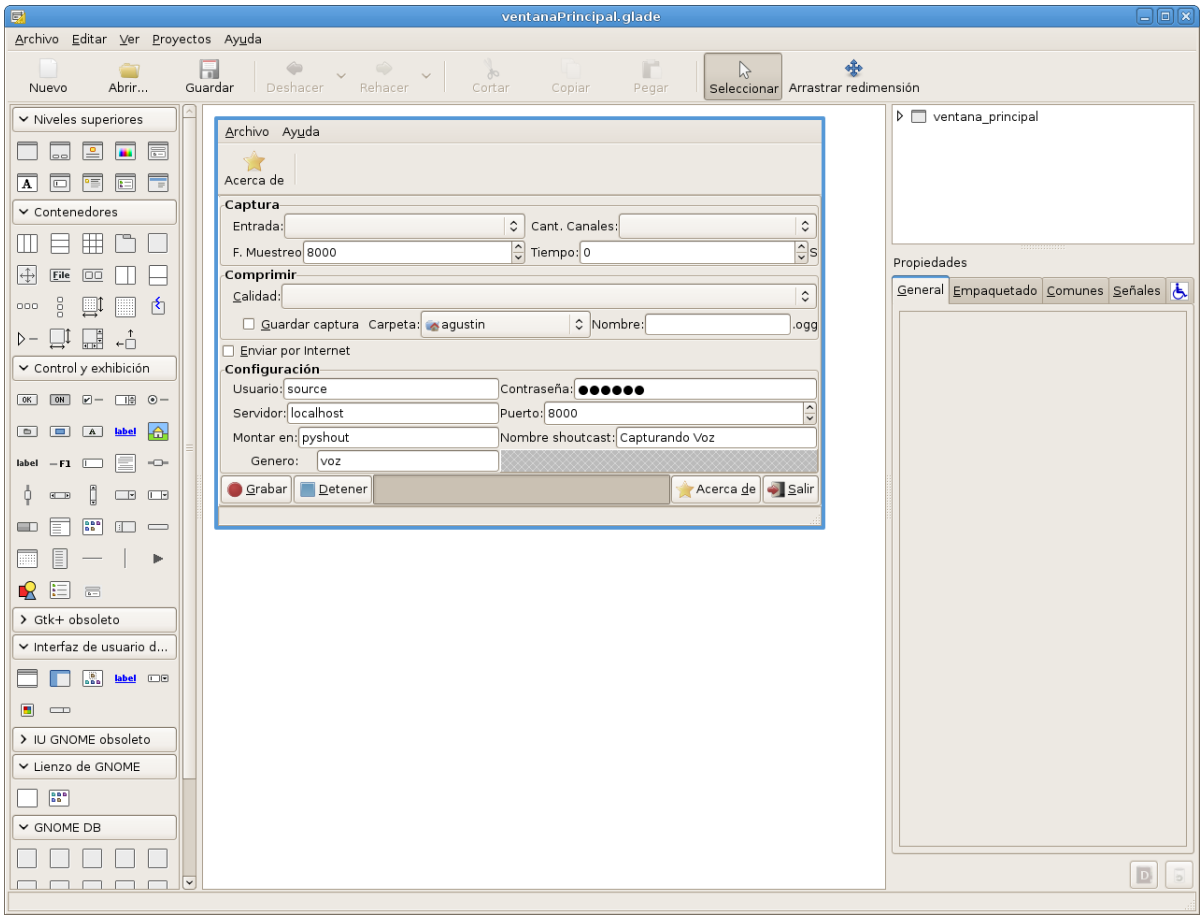

Figura 20: Pantallazo de Glade

[3]GTK es una biblioteca del equipo GTK+, la cual contiene los objetos y funciones para crear la interfaz gráfica de usuario. Maneja widgets como ventanas, botones, menús, etiquetas, deslizadores, pestañas, etc.

[4]Python es un lenguaje de programación creado por Guido van Rossum cuyo nombre está inspirado en el grupo de cómicos ingleses "Monty Python". Es interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos. Viene con una gran colección de módulos estándar.

[5]Glade (o Glade Interface Designer, que significa Diseñador de interfaz Glade) es una herramienta de desarrollo visual de interfaces gráficas mediante GTK/GNOME. Es independiente del lenguaje de programación y predeterminadamente no genera código fuente sino un archivo XML.

9.2. acciones.py

Acá se realizan todas las acciones que debe tener el programa. Consiste en controlar varios procesos comunicados entre sí, para lograr tener compresión de la señal, guardarla y transmitirla por medio de la tecnología shoutcast. Todo esto lo fusiona en una sola clase para poder ser utilizada fácilmente. La misma abre un proceso lo cual nos facilita el desarrollo de la interfaz gráfica sin cuelgues ni demoras excesivas.

9.3. proyecto_final.py

Es el encargado de interactuar con la parte gráfica y acciones.py, lo que el usuario introduce en la interfaz gráfica se lo comunica a acciones.py

10. Circuito de Alimentación

Para la alimentación se necesitan 3 tensiones diferentes, una fuente simétrica de $\pm 5V$ para los operacionales de la placa de acondicionamiento y $3V$ para alimentar las placas de RF y los microntroladores.

La fuente simétrica se construyo con 2 reguladores de $\pm 5V$ respectivamente y los $3V$ con otro regulador.

Los reguladores fueron 7805 y 7905 para $\pm 5V$, y el HT7130 para los $3V$. Al final del informe se adjuntan las hojas de datos.

En la figura 21 se muestra el circuito de alimentación.

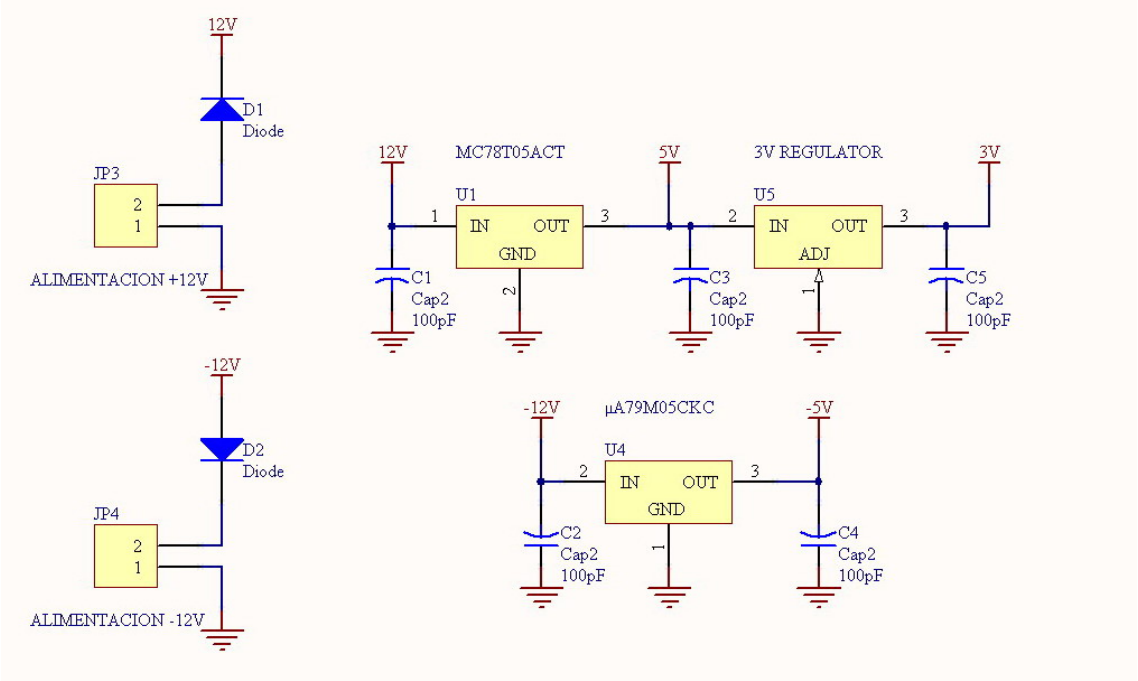

Figura 21: Fuente de alimentación.