Excitador
La vaca
Manual de usuario

Índice

El excitador puede ser configurado de dos maneras; con 8 interruptores que están soldados a la tarjeta y mediante un puerto de comunicación serial RS232.

Con los interruptores solo es posible seleccionar la frecuencia de operación, mientras que con el puerto serial el excitador puede comunicar información referente a su funcionamiento, como el estado de algunos contadores, además de poder seleccionar la frecuencia y la potencia de salida.

1. Sin computadora

Si no tienes disponible una computadora con puerto serial o simplemente no quieres usarla para configurar los parámetros del excitador, debes prender el interruptor número 8 del dipswitch; entonces, el aparato leerá la frecuencia de los 7 interruptores restantes del mismo dipswitch y no de su memoria.

1.1. Cómo seleccionar la frecuencia

Podemos considerar que los 7 interruptores definen un número binario, el cual corresponde secuencialmente a las frecuencias de operación que van de 87.9 MHz hasta 107.9 MHz. Es decir al número binario 0000000 corresponde la frecuencia 87.9 MHz, al número 0000001 la frecuencia 88.0 MHz y así sucesivamente hasta llegar a 107.9 MHz, o sea 1100100.

En lugar de contar desde 0 hasta el número que corresponde a la frecuencia que queremos configurar, podemos usar la siguiente relación:

$$n = \frac{f \times 10 - 879}{2} \tag{1}$$

Donde, f es la frecuencia en MHz, y n es el número que hay que configurar en el dipswitch; n está en base decimal, hace falta convertirlo a base binaria.

Digamos por ejemplo, que queremos operar en la frecuencia de los 93.5 MHz.

Si aplicamos la relación (1), tenemos:

$$\begin{aligned} n &= \frac{93{,}5 \times 10 - 879}{2} \\ &= 28 \end{aligned}$$

28 en base binaria es 0011100.

2. Con computadora

2.1. Cómo conectar a una computadora y probar la comunicación

El excitador cuenta con un puerto de comunicación serial que sigue el estándar RS232; puedes conectarlo a un puerto serial de una computadora, necesitarás un programa emulador de terminal que se pueda conectar a un puerto serie, en GNU/Linux puedes usar gtkterm; además debes configurarlo con los siguientes parámetros:

Velocidad	9600
Bits de datos	8
Bits de paro	1
Paridad	Ninguna

Prueba la comunicación, por ejemplo, pregúntale al excitador en qué frecuencia está trabajando; debes escribir en la terminal el comando:

```
f?
```

Para que el comando sea enviado al excitador, debes oprimir la tecla enter después de escribir el comando. Si la comunicación funciona correctamente, el excitador responderá con la frecuencia de operación, así:

```
F1021
```

Eso significa que la frecuencia de operación del excitador es 102.1 MHz.

2.2. Cómo seleccionar la frecuencia

El comando para seleccionar la frecuencia tiene la forma

```
Fiiii
```

La `f` puede ser minúscula o mayúscula, `iiii` debe sustituirse por la frecuencia de operación en megahertz multiplicada por 10.

El excitador responde con el mensaje:

```
Fiiii
```

Por ejemplo, si queremos operar el excitador a 105.3 MHz el comando sería:

```
F1053
```

2.3. Cómo controlar la potencia

Es posible controlar la potencia del excitador; el comando tiene la forma:

```
Ahh
```

`hh` es un número expresado en base hexadecimal, que va desde 00 hasta FF; 00 es la potencia mínima y FF la máxima.

El excitador debe responder con:

```
Ahh
```

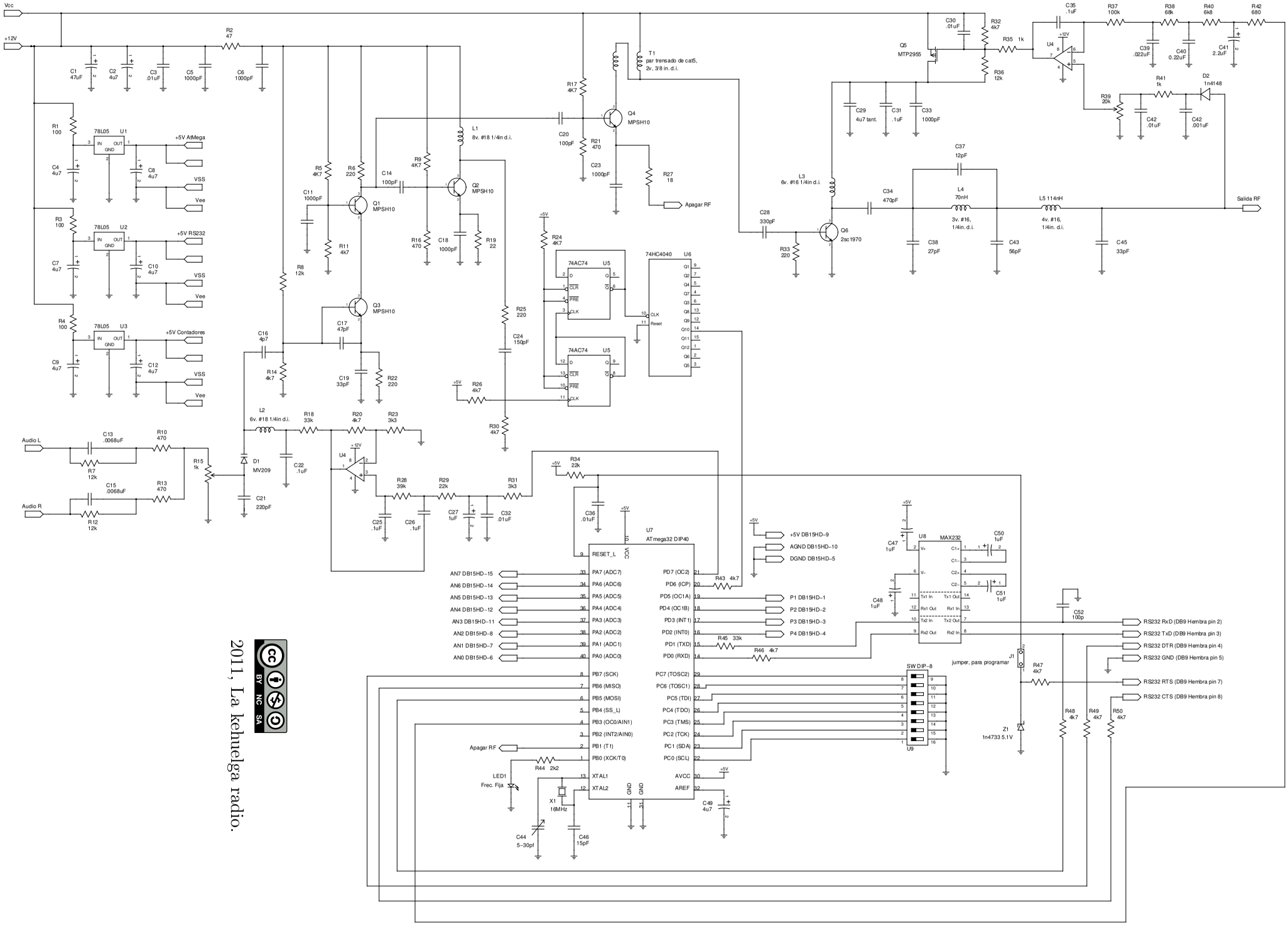
2.4. Referencia de comandos

Hay bastantes comandos para cambiar y conocer el estado del excitador, están resumidos en el siguiente cuadro, en el que i se refiere a un dígito decimal y h a un dígito hexadecimal.

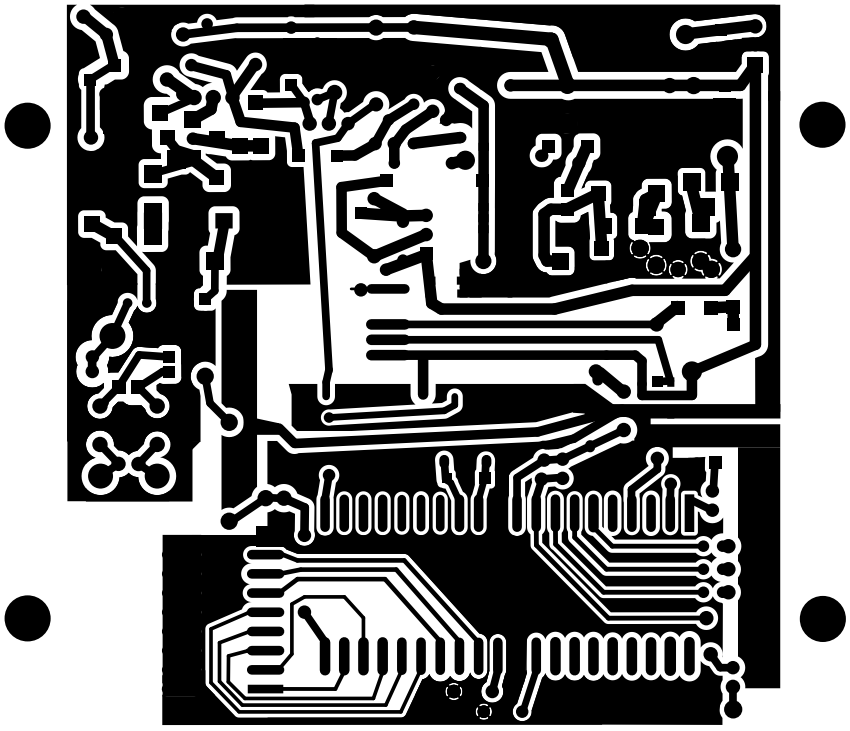
Comando	Descripción	Respuesta	Error
A*hh*	Cambia la potencia	A*hh*	
A?	Reporta la potencia	A*hh*	
GA	Guarda la potencia en la EEPROM	GA*hh*	
RA	Restaura la potencia desde la EEPROM	RA*hh*	
IAM	La potencia de inicio se lee de la EEPROM	IAM	
IA0	Hace que la potencia de inicio sea 0	IA0	
IA?	Reporta la fuente de la potencia de inicio	IAM, IA0	
F*iiii*	Cambia la frecuencia	F*iiii*	ERROR: Frec. INVALIDA!
F?	Reporta la frecuencia	F*iiii*	
GF	Guarda la frecuencia seleccionada en la EEPROM	GF*iiii*	
IFS	La frecuencia de inicio se lee del dipswitch	IFS	
RF	Restaura la frecuencia desde la EEPROM	RF*iiii*	ERROR: Frec. INVALIDA!
IFM	La frecuencia de inicio se lee de la EEPROM	IFM	
IF?	Reporta la fuente de la frecuencia de inicio	IFM, IFS	
S?	Reporta el dipswitch	S*hh*	
P[0-3][01]	Cambia el valor de un pin de control	P[0-3][01]	
P[0-3]	Reporta el valor del pin de un control	P[0-3][01]	
V[0-7]	Reporta un canal del ADC (convertidor analógico digital)	V[0-7]*hhh*	

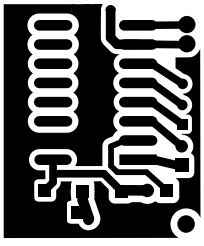
3. Diagramas

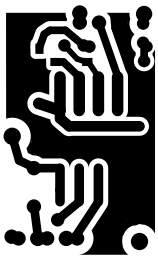
3.1. Esquemático

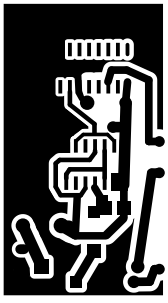


3.2. PCB

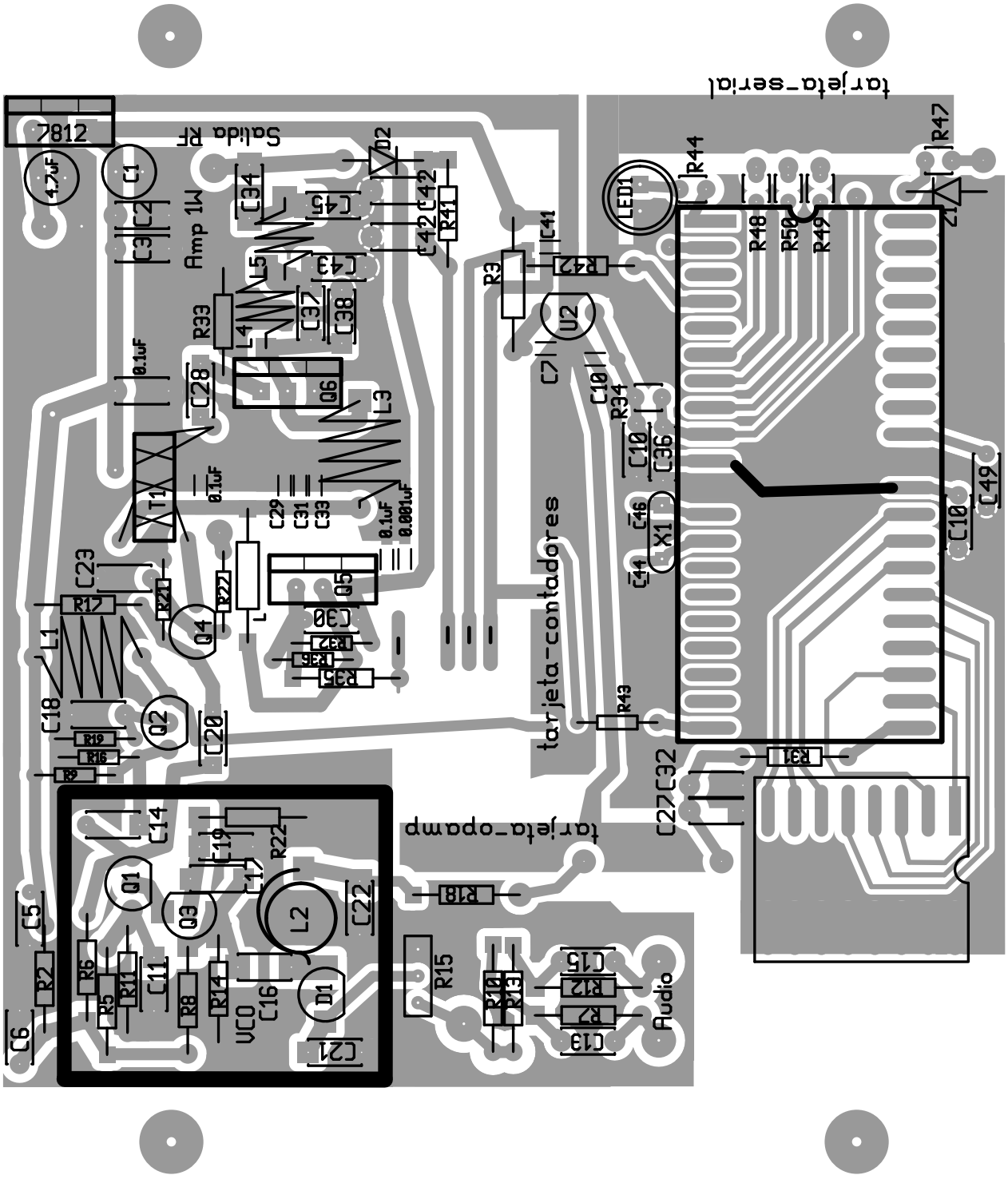




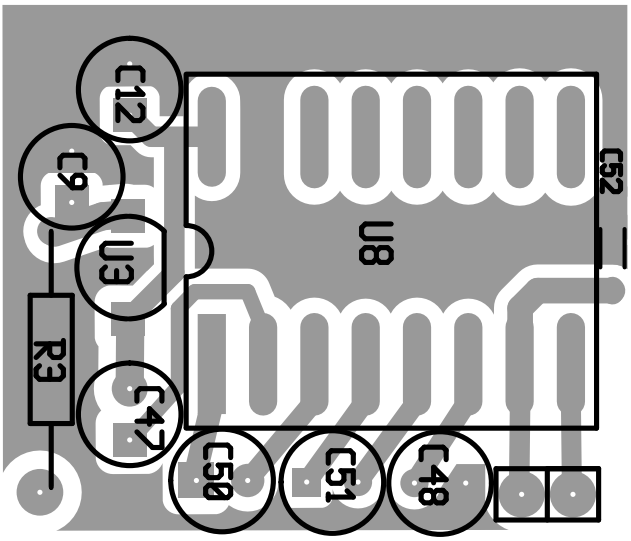




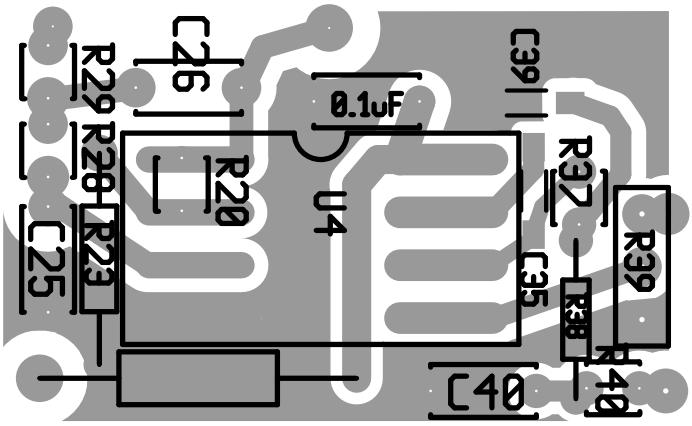
3.2.1. Tarjeta madre



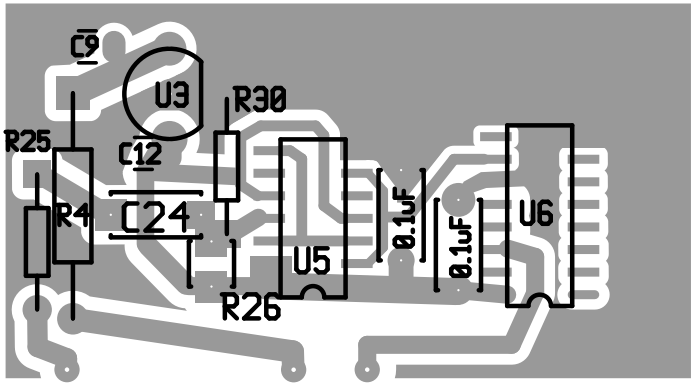
3.2.2. Tarjeta de comunicación serial



3.2.3. Tarjeta del amplificador operacional



3.2.4. Tarjeta de contadores



4. Código Fuente

4.1. modos.c

```
/*
  Copyright 2010, la kehuelga radio.
  This program is free software: you can redistribute it and/or modify qit under
  the terms of the GNU General Public License as published by the Free Software
  Foundation, either version 3 of the License, or (at your option) any later
  version.

  This program is distributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
  FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
  details.

  You should have received a copy of the GNU General Public License along with
  this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "vaca.h"
#include "serial.h"

/* las funciones que corren en main y en la interrupción, respectivamente */
void (*funmain)(), (*funinter)();

/* semáforo de la interrupción pwm, definido en vaca.c */
uint8_t pwmsema;

uint8_t estado = 0, amp;
uint16_t dif, contvco, ucontvco, veces;
int16_t frec;
uint32_t ds, lvco=7, vco = 0x80707007;
int32_t error;
static uint32_t acc = 7;
```

```
uint16_t nvfrec __attribute__((section (".eeprom"))) = (NV_FREC);
uint8_t nvamp __attribute__((section (".eeprom"))) = (NV_AMP);
uint8_t nvfi __attribute__((section (".eeprom"))) = (NV_IF);
uint8_t nvai __attribute__((section (".eeprom"))) = (NV_IA);

void pll_main()
{
    static uint8_t upwmsema=0;
    if ( estado == 1 ) {
        PORTB |= _BV(PB0);
    } else {
        PORTB &= ~_BV(PB0);
    }

    if (pwmsema ^ upwmsema) {
        if(veces < 65500)  veces += (uint8_t)(pwmsema - upwmsema);
        upwmsema = pwmsema;

        if(veces > 62000) {
            estado = 1;
        }

        if(veces > 50000) {
            OCR0 = amp;
        }
    }
}

void pll_inter()
{
    OCR2 = (uint8_t)(acc >> 24);

    if(TIFR & _BV(ICF1)) {
        TIFR |= _BV(ICF1);
        contvco = ICR1;
        dif = contvco - ucontvco;
        ucontvco = contvco;

        error = ((int32_t)dif * (int32_t)(BITS_FREC) * (int32_t)((ESCALA) / (DIV_HW))) - ds;
        lvco &= (0xffffffff >> (KS));
        lvco += error;
        vco += (((int32_t)lvco) >> (KS));
        if( estado == 0) {
            vco += (error >> (KRAPIDO));
        }
    }
```

```
    acc &= 0x00ffffff;
    acc += vco;

}

void modo_pll()
{
    estado = 0;
    veces = 0;

    /* si el pin #7 del dipswitch. no está en modo manual, y la fuente de la
       amplitud de inicio es cero, entonces la amplitud de inicio es 0. En todos
       los demás casos, la amplitud de inicio es la que está guardada en la
       EEPROM. */
    if((PINC & 0x80) && (eeprom_read_byte(&nvai) == '0')) {
        amp = 0;
    } else {
        amp = eeprom_read_byte(&nvamp);
    }

    if((PINC & 0x80) && (eeprom_read_byte(&nvfi))) {
        frec = eeprom_read_word(&nvfrec);
    } else {
        frec = 879 + 2 * (uint16_t)(PINC & 0x7f);
    }

    ds = calcds(frec);
    funmain = pll_main;
    funinter = pll_inter;
}

void debug_main() {}

void debug_inter()
{
    OCR2 = (uint8_t)(acc >> 24);
    if(TIFR & _BV(ICF1)) {
        TIFR |= _BV(ICF1);
        contvco = ICR1;
        dif = contvco - ucontvco;
        ucontvco = contvco;
        error = ((int32_t)dif * (int32_t)(BITS_FREC) * (int32_t)((ESCALA) / (DIV_HW))) - ds;
    }
    acc &= 0x00ffffff;
    acc += vco;
```

```
}

void modo_debug()
{
    amp = 0x00;
    acc = 0x80000007;

    funmain = debug_main;
    funinter = debug_inter;
}

int32_t calcds(int16_t frec)
{
    ldiv_t debe_ser;
    int32_t dsint;

    debe_ser = ldiv(((long)(FREC_CPU)*(long)(ESCALA)),(long)frec);
    dsint = debe_ser.quot * (BITS_FREC);
    debe_ser = ldiv((debe_ser.rem * (BITS_FREC)),(long)frec);
    dsint += debe_ser.quot;
    return dsint;
}
```

4.2. modos.h

```
/*
  Copyright 2010, la kehuelga radio.
  This program is free software: you can redistribute it and/or modify qit under
  the terms of the GNU General Public License as published by the Free Software
  Foundation, either version 3 of the License, or (at your option) any later
  version.

  This program is distributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
  FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
  details.

  You should have received a copy of the GNU General Public License along with
  this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef MODOS_H
#define MODOS_H

/* para definir un modo, hay que escribir 3 elementos:

   1. las funciones que corren en main y en la interrupción del pwm.  Estas 2
```

```
   funciones tienen la forma

   void nombre_main(void)
   void nombre_inter(void)

   2. una función para inicializar el modo, con la forma

   void modo_nombre(void)

   3. Un comando del puerto serial definido en serial.rl para entrar en el
   modo. Y quizás más comandos para usarlo.

   Para ver ejemplos, mira los que están en serial.rl y modos.c
*/

extern void modo_debug();
extern void modo_pll();
extern void (*funinter)(), (*funmain)();

extern uint8_t pwmsema, estado, vestado, amp;
extern int16_t frec;
extern uint16_t veces, dif;
extern int32_t ds;
extern int32_t error;
extern uint32_t vco;

extern uint16_t nvfrec __attribute__((section (".eeprom")));
extern uint8_t nvamp __attribute__((section (".eeprom")));
extern uint8_t nvai __attribute__((section (".eeprom")));
extern uint8_t nvfi __attribute__((section (".eeprom")));

#endif
```

4.3. serial.h

```
/*
  Copyright 2010, la kehuelga radio.
  This program is free software: you can redistribute it and/or modify qit under
  the terms of the GNU General Public License as published by the Free Software
  Foundation, either version 3 of the License, or (at your option) any later
  version.

  This program is distributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
  FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
  details.
```

```
  You should have received a copy of the GNU General Public License along with
  this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef SERIAL_H
#define SERIAL_H

extern uint8_t *btx, *brx, bufrx[], buftx[];

void init_UART();
void init_scaner();
void bufrx_inicio();
void envia_UART();
void leer_cmds();

#endif
```

4.4. serial.rl

```
/*
  Copyright 2010, la kehuelga radio.
  This program is free software: you can redistribute it and/or modify qit under
  the terms of the GNU General Public License as published by the Free Software
  Foundation, either version 3 of the License, or (at your option) any later
  version.

  This program is distributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
  FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
  details.

  You should have received a copy of the GNU General Public License along with
  this program.  If not, see <http://www.gnu.org/licenses/>.
*/

/* -*-c-*- */
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "vaca.h"
```

```
#include "modos.h"

/* para transmitir datos solo es necesaria la función printf */

char *btx, *brx, bufrx[(CARS_BUFRX)];
char buftx[(CARS_BUFTX)] = "La Vaca: Muuuu!\r";
int8_t cs, act, hay=0;
char *ts, *te=0, *p=bufrx, *pe=bufrx, *eof=0;

uint8_t resadch, resadcl, bitc;
uint16_t argnum;
uint32_t argnum32;

%%{
    machine cmds;

    action err_cmd {
        strcat(buftx, "KE?\r");
    }

 main := |*
# Cambiar la Amplitud:
    ( [Aa] (xdigit{2} @{sscanf((ts+1),"%02x", &argnum);}) ('\r')) =>
        {
            amp = (uint8_t)argnum;
            OCR0 = amp;
            sprintf((buftx+strlen(buftx)), "A%02X\r", amp);
        };

# Reportar la Amplitud:
    ( [Aa] ('?') ('\r')) =>
        {
            sprintf((buftx+strlen(buftx)), "A%02X\r", amp);
        };

# Guardar la Amp. en la EEPROM:
    ( [Gg] [Aa] ('\r')) =>
        {
            eeprom_write_byte(&nvamp, amp);
            sprintf((buftx+strlen(buftx)), "GA%02X\r", amp);
        };

# Restaurar la Amp. de la EEPROM:
    ( [Rr] [Aa] ('\r')) =>
        {
            amp = eeprom_read_byte(&nvamp);
```

```
            _delay_us(100);
            OCR0 = amp;
            sprintf((buftx+strlen(buftx)), "RA%02X\r", amp);
        };

# Amplitud de Inicio = Memoria
    # (si dipsw #7 = 1, cuando estado = 1, la amplitud se inicia con el valor de la nvram):
    ( [Ii] [Aa] [Mm] ('\r') ) =>
        {
            eeprom_write_byte(&nvai, 'M');
            sprintf((buftx+strlen(buftx)), "IAM\r");
        };

# Amplitud de Inicio = cero
    # (si dipsw #7 = 1, cuando estado = 1, la amplitud no se prende sola):
    ( [Ii] [Aa] ('0') ('\r') ) =>
        {
            eeprom_write_byte(&nvai, 0x00);
            sprintf((buftx+strlen(buftx)), "IA0\r");
        };

# Reportar fuente de la amplitud de inicio:
    ( [Ii] [Aa] ('?') ('\r') ) =>
        {
            argnum = eeprom_read_byte(&nvai);
            sprintf((buftx+strlen(buftx)), "IA%1c\r", (argnum)? 'M' : '0');
        };

# Cambiar la Frecuencia:
    ( [Ff] (digit{4}) @{sscanf((ts+1), "%04d", (int *)&argnum);} ('\r')) =>
        {
            if((argnum >= 879) && (argnum <= 1079)) {
                veces = 0;
                estado = 0;
                frec = argnum;
                ds = calcds((uint16_t)frec);
                sprintf((buftx+strlen(buftx)), "F%04d\r", frec);
            } else {
                sprintf((buftx+strlen(buftx)), "ERROR: Frec. MALA!\r");
            }
        };

# Reportar la Frecuencia:
    ( [Ff] ('?') ('\r')) =>
        {
            sprintf((buftx+strlen(buftx)), "F%04d\r", frec);
```

```
        };

# Guardar la frecuencia seleccionada en la EEPROM:
    ( [Gg] [Ff] ('\r')) =>
        {
            eeprom_write_word(&nvfrec, frec);
            sprintf((buftx+strlen(buftx)), "GF%04d\r", frec);
        };

# Restaurar la frecuencia de la EEPROM:
    ( [Rr] [Ff] ('\r')) =>
        {
            argnum =  eeprom_read_word(&nvfrec);
            if((argnum >= 879) && (argnum <= 1079)) {
                veces = 0;
                estado = 0;
                frec = argnum;
                ds = calcds((uint16_t)frec);
                sprintf((buftx+strlen(buftx)), "RF%04d\r", frec);
            } else {
                sprintf((buftx+strlen(buftx)), "ERROR: Frec. MALA!\r");
            }
        };

# Frecuencia de inicio de los dipswitch:
    ( [Ii] [Ff] [Ss] ('\r') ) =>
        {
            eeprom_write_byte(&nvfi, 0x00);
            sprintf((buftx+strlen(buftx)), "IFS\r");
        };

# Frecuencia de inicio de la nvram:
    ( [Ii] [Ff] [Mm] ('\r') ) =>
        {
            eeprom_write_byte(&nvfi, 0x01);
            sprintf((buftx+strlen(buftx)), "IFM\r");
        };

# Reportar fuente de la frecuencia de inicio:
    ( [Ii] [Ff] ('?') ('\r') ) =>
        {
            argnum = eeprom_read_byte(&nvfi);
            sprintf((buftx+strlen(buftx)), "IF%1c\r", (argnum) ? 'M' : 'S');
        };

# Reportar los interruptores:
```

```
    ( [Ss] ('?') ('\r')) =>
        {
            sprintf((buftx+strlen(buftx)), "S%02X\r", leer_dipsw());
        };

# Cambiar un pin de control:
    ( [Pp] ([0-3] @{argnum = *(ts+1) - '0';}) ([01] @{bitc = *(ts+2);}) ('\r') ) =>
        {
            if(bitc - '0') {
                PINES_CONTROL |= _BV(PCIZQ + (uint8_t)argnum);
            } else {
                PINES_CONTROL &= ~(_BV(PCIZQ + (uint8_t)argnum));
            }
            sprintf((buftx+strlen(buftx)), "P%1X%1c\r", argnum, bitc);
        };

# Reportar un pin de control:
    ( [Pp] ([0-3] @{argnum = *(ts+1) - '0';}) ('?') ('\r') ) =>
        {
            printf((buftx+strlen(buftx)), "P%1X%1c\r",
                   argnum, (PINES_CONTROL & _BV(PCIZQ + argnum)) ? '1' : '0');
        };

# Reportar un canal del ADC:
    ( [Vv] ([0-7] @{sscanf((ts+1),"%1x", &argnum);}) ('?') ('\r') ) =>
        {
            ADMUX &= 0xf8;
            ADMUX |= ((uint8_t) argnum);
            /* un poco más tiempo para el muestreo */
            _delay_us(100);
            ADCSRA |= _BV(ADSC);
            loop_until_bit_is_clear(ADCSRA, ADSC);
            resadcl = ADCL;
            resadch = ADCH;
            sprintf((buftx+strlen(buftx)), "V%1X%01X%02X\r", argnum, resadch, resadcl);
        };

# Cambiar el vco:
    ( [Pp] [Vv] (xdigit{8} @{sscanf((ts+2),"%08lx", &argnum32);}) '\r') =>
        { /* espera la próxima consulta del UART antes de cambiar el
             apuntador del bufrx */
            lcpwmsema = pwmsema;
            while(pwmsema == lcpwmsema)
                ;
            vco = argnum32;
            sprintf((buftx+strlen(buftx)), "PV%08lX\r", argnum32);
```

```
        };

# Reportar el vco:
    ( [Pp] [Vv] '?' '\r') =>
        { /* espera la próxima consulta del UART antes de cambiar el
             apuntador del bufrx */
            lcpwmsema = pwmsema;
            while(pwmsema == lcpwmsema)
                ;
            argnum32 = vco;
            sprintf((buftx+strlen(buftx)), "PV%08lX\r", argnum32);
        };

# Reportar el error:
    ( [Pp] [Ee] '?' '\r') =>
        { /* espera la próxima consulta del UART antes de cambiar el
             apuntador del bufrx */
            lcpwmsema = pwmsema;
            while(pwmsema == lcpwmsema)
                ;
            argnum32 = error;
            sprintf((buftx+strlen(buftx)), "PE%08lX\r", argnum32);
        };

# Reportar el contador:
    ( [Pp] [Cc] '?' '\r') =>
        { /* espera la próxima consulta del UART antes de cambiar el
             apuntador del bufrx */
            lcpwmsema = pwmsema;
            while(pwmsema == lcpwmsema)
                ;
            argnum = dif;
            sprintf((buftx+strlen(buftx)), "PC%04d\r", argnum);
        };

# Reportar el estado del pll::
    ( [Ee] '?' '\r') =>
        {
            sprintf((buftx+strlen(buftx)), "E%1X\r", estado);
        };

# Cambiar al modo debug:
    ( [Mm] [Dd] '\r' ) =>
        {
            cli();
            modo_debug();
```

```
            sei();
            strcat(buftx, "MD\r");
        };

# Cambiar al modo pll:
    ( [Mm] [Pp] '\r' ) =>
        {
            cli();
            modo_pll();
            sei();
            strcat(buftx, "MP\r");
        };

    *|;

}%%

%% write data;

void init_scaner()
{
    ts = bufrx;
    te=0;
    p=bufrx;
    pe=bufrx;
    eof=0;
    %% write init;
}

void init_UART()
{
    UBRRH = (uint8_t)((BAUD) >> 8);
    UBRRL = (uint8_t)(BAUD);
    UCSRB = (1 << RXEN) | (1 << TXEN);
    UCSRC = (1 << URSEL) | (3 << UCSZ0);
    brx = bufrx;
    btx = buftx;
}

void envia_UART()
{
    /* UART TX: */
    if(*btx) {
        if(UCSRA & (1<<UDRE)) {
            UDR = *btx++;
        }
```

```
    } else {
        btx = buftx;
        *btx = 0;
    }
}

void bufrx_inicio()
{
    uint8_t bipwmsema;

    /* espera una interrupción de captura */
    bipwmsema = pwmsema;
    while(pwmsema == bipwmsema)
        ;
    brx = p = pe = bufrx;

}

void err_bufrx()
{
    bufrx_inicio();
    strcat(buftx, "ERROR: BUFRX LLENO!\r");
}

void leer_cmds()
{
    uint8_t lcpwmsema;
    int8_t espacio=((CARS_BUFRX)-hay);

    /* ¿hay algo nuevo? */
    if(pe == brx) return;

    /* error, se ha llenado el bufrx: */
    if(espacio == 0) {
        err_bufrx();
        return;
    }

    /* espera la próxima consulta del UART antes de cambiar el apuntador del
       bufrx */
    lcpwmsema = pwmsema;
    while(pwmsema == lcpwmsema)
        ;

    pe = brx;
    %%write exec;
```

```
    if(cs == cmds_error) {
        strcat(buftx, "KE?\r");
        init_scaner();
        bufrx_inicio();
    }
    if(ts == 0) {
        hay = 0;
        bufrx_inicio();
    }
    else {
        /* espera la próxima consulta del UART antes de cambiar el apuntador del
           bufrx */
        lcpwmsema = pwmsema;
        while(pwmsema == lcpwmsema)
            ;

        hay = brx - ts;
        if(brx-bufrx > 12) {
            brx = bufrx + hay;
            memmove(bufrx, ts, hay);
            te = bufrx + (te-ts);
            p = bufrx +(p - ts);
            ts = bufrx;
        }
    }
}
```

4.5. vaca.c

```
/*
  Copyright 2010, la kehuelga radio.
  This program is free software: you can redistribute it and/or modify qit under
  the terms of the GNU General Public License as published by the Free Software
  Foundation, either version 3 of the License, or (at your option) any later
  version.

  This program is distributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
  FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
  details.

  You should have received a copy of the GNU General Public License along with
  this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdlib.h>
```

```
#include <stdint.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "vaca.h"
#include "serial.h"
#include "modos.h"

/*
  FUSES =
  {
  .low = 0xff,
  .high = (FUSE_BOOTSZ0 & FUSE_BOOTSZ1 & FUSE_SPIEN & FUSE_JTAGEN & FUSE_OCDEN)
  };
*/

void init_pines()
{
    DDRB |= _BV(PB0);
    PORTB &= ~_BV(PB0);
    PINES_CONTROL &= ~(0x0f << (PCIZQ));
    PC_DDR |= (0x0f << (PCIZQ));
}

void init_dipsw()
{
    /* activar pull-ups para el dipswitch: */;
    SFIOR &= ~_BV(PUD);
    DDRC = 0;
    PORTC        = 0xff;
    PINC = 0xff;
}

void init_adc()
{
    DDRA = 0;
    PORTA = 0;
    ADMUX = (ADCREF);
    ADCSRA = (ADCPRE);

    /* una conversión para tirar */
```

```
    ADCSRA |= _BV(ADSC);
}

void init_timers()
{
    DDRB |= _BV (PB3);
    DDRD |= _BV (PD7);
    PORTB |= _BV (PB3);
    PORTD |=  _BV (PD7);

    TCCR1A = 0x0;
    TCCR1B = 0x01;

    TCCR2 = 0x61;
    TCCR0 = 0x69;

    TIMSK = 0x40; /* cambia a 0x60 para habilitar capture de timer1 */
}

uint8_t leer_dipsw()
{
    return (PINC);
}

ISR(BADISR_vect)
{
}

/* el timer2 genera el pwm para controlar la frecuencia */
ISR(TIMER2_OVF_vect)
{
    (*funinter)();

    /* UART rcp: */
    if(UCSRA & (1<<RXC)) {
        *brx++ = UDR;
    }

    pwmsema++;
}


int main()
{
    /* inicialización de los periféricos */
    init_pines();
```

```
    init_adc();
    init_dipsw();
    init_timers();
    init_UART();

    /* scaner de los comandos que llegan por el puerto serial */
    init_scaner();

    /* modo inicial: */
    modo_pll();

    sei();

    while (1) {
        leer_cmds();
        envia_UART();
        (*funmain)();
    }

    return 0;
}
```

4.6. vaca.h

```
/*
  Copyright 2010, la kehuelga radio.
  This program is free software: you can redistribute it and/or modify qit under
  the terms of the GNU General Public License as published by the Free Software
  Foundation, either version 3 of the License, or (at your option) any later
  version.

  This program is distributed in the hope that it will be useful, but WITHOUT
  ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
  FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
  details.

  You should have received a copy of the GNU General Public License along with
  this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef VACA_H
#define VACA_H

#define F_CPU 16000000UL

/* estado inicial programado en la lnvram: */
#define NV_FREC 1029
```

```
#define NV_AMP 0xff
#define NV_IF 0x01
#define NV_IA 'M'

/* valores para el pll */
#define KV 1
#define KS 16
#define KRAPIDO 8

/* unos valores para el reloj de UART, UBRR, con fosc=16MHz, UB2x = 0 */
#define BAUD_2400 416
#define BAUD_4800 207
#define BAUD_9600 103
#define BAUD_19200 51
#define BAUD_38400 25

#define BAUD BAUD_9600

/* ref defs en ADMUX: */
#define ADCREF _BV(REFS0)

/* ADC preescaler etc en ADCSRA */
#define  ADCPRE  _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0)

/* pines para controlar otros equipos */
#define PINES_CONTROL PORTD
#define PC_DDR DDRD
#define PCIZQ 2

/* tamaños de los buffers */
#define CARS_BUFTX 32
#define CARS_BUFRX 32

/* valor máximo del error del VCO para considerar fija la frecuencia */
#define ERROR_MAX 5000000

/* cuantas veces el error del VCO tiene que ser menor que ERROR_MAX para
   considerar fija la frecuencia */
#define ERROR_ESPERA 300

/* frecuencia del cpu en multiplos de 100kHz: */
#define FREC_CPU 160

/* divisor de vco en hardware: */
#define DIV_HW 4096
```

```
/* Potencia mínima de 2 más grande que la frec. máxima en multiplos de
   100kHz: */
#define BITS_FREC 0x400

/* factor escalador de frec. para aprovechar la precisión máxima de un
   int32_t: */
#define ESCALA 0x800000

int32_t calcds(int16_t frec);
uint8_t leer_dipsw();

#endif
```

4.7. Makefile

```
#############################################################################
# La vaca: MUUUU! la vaca
#############################################################################

## General:
PROJECT = vaca
MCU = atmega32
TARGET = vaca.elf
CC = avr-gcc

#############################################################################
# Opciones para los varios programadores / debug:
#############################################################################

## HFUSE debe ser 0x10 para usar el JTAG, y 0xd8 para usar los dipswitch.
## OR con 0x01 para deshabilitar el bootloader;
## OR con 0x02 para tener espacio de bootloader de 1k en lugar de 2k;
## AND con 0xf7 para preservar el eeprom a pesar del "chip erase";
## LOCK totalmente deshabilitado es 0xff;
## OR con 0x7f para proteger el bootloader (deshabilitar la instrucción SPM
##    en la sección del bootloader)

PORT='/dev/ttyUSB0'
BAUD=9600
HEXFILE=vaca.hex
EEPROMFILE=vaca.eep
PROGRAMMER=avr109
DIRBOOT="avrprog_boot_v0_85"

# Deshabilita el modo seguro, el bootloader no podrá restaurar los fusibles
```

```
# de ningun modo
DIS_SAVE=-u

## Opciones de Ragel:
RAGEL_OPCIONES = -C -G1

## Opciones comunes a las reglas para compilar, ligar y ensamblar
COMMON = -mmcu=$(MCU)

## Opciones de compilación
CFLAGS = $(COMMON)
CFLAGS += -Wall -pedantic -g -O3  -funsigned-char -funsigned-bitfields \
          -fpack-struct -fshort-enums
CFLAGS += -Wp,-M,-MP,-MT,$(*F).o,-MF,dep/$(@F).d

## Banderas del ensamblador
ASMFLAGS = $(COMMON)
ASMFLAGS += -x assembler-with-cpp -Wa,-gdwarf2

## Banderas del linker
LDFLAGS = $(COMMON)
LDFLAGS += -lm

## Banderas para producir un archivo Intel Hex
HEX_FLASH_FLAGS = -R .eeprom

HEX_EEPROM_FLAGS = -j .eeprom
HEX_EEPROM_FLAGS += --set-section-flags=.eeprom="alloc,load"
HEX_EEPROM_FLAGS += --change-section-lma .eeprom=0

INCLUDES = -I/usr/avr/include/

## Objetos necesarios para ligar
OBJECTS = vaca.o serial.o modos.o

##
all: $(TARGET) vaca.hex vaca.eep

boot:
        $(MAKE) -C $(DIRBOOT)

programar: all
        avrdude $(DIS_SAVE) -p $(MCU) -P $(PORT) -c $(PROGRAMMER) -b $(BAUD) \
        -v -U flash:w:$(HEXFILE) -U eeprom:w:$(EEPROMFILE)

## Compilar
```

```
vaca.o: vaca.c
	$(CC) $(INCLUDES) $(CFLAGS) -c   $< -o $@

modos.o: modos.c
	$(CC) $(INCLUDES) $(CFLAGS) -c   $< -o $@

serial.o: serial.c
	$(CC) $(INCLUDES) $(CFLAGS) -O0 -c   $< -o $@

# Ragel:
serial.c: serial.rl
	ragel $(RAGEL_OPCIONES) $<

## Ligar
$(TARGET): $(OBJECTS)
	 $(CC) $(LDFLAGS) $(OBJECTS) $(LIBDIRS) $(LIBS) -o $(TARGET)

%.hex: $(TARGET)
	avr-objcopy -O ihex $(HEX_FLASH_FLAGS)  $< $@

%.eep: $(TARGET)
	avr-objcopy $(HEX_EEPROM_FLAGS) -O ihex $< $@

%.lss: $(TARGET)
	avr-objdump -h -S $< > $@

## Clean target
.PHONY: clean
clean:
	-rm -rf $(OBJECTS) vaca.elf serial.c dep/ vaca.hex vaca.eep

## Otras dependencias
-include $(shell mkdir dep 2>/dev/null) $(wildcard dep/*)
```