Trabajo Práctico 3

Medidor de Capacidad e Inductancia

Taller de Proyecto I

Universidad Nacional de La Plata

Facultad de Ingeniería

Octubre 2021

0850/3 Gandin, Mariano Nicolás

1425/3 Lasala, Gian Franco

1804/1 Schnack Aragón, Edelmiro

Repositorio del proyecto en https://github.com/gianowolf/RLC-Meter

Índice

Introducción

El presente documento detalla el uso del microcontrolador STM32F103C como medidor de capacitancia e inductancia de condensadores y bobinas, respectivamente, conectados al sistema. Ver Figura 1.

Las fases del proyecto se dividen en la implementación, simulación y depuración del **firmware**, y diseño del circuito impreso del **hardware**.

Requerimientos Funcionales

El sistema debe informar a través de un display LCD al usuario la capacitancia e inductancia de dispositivos de almacenamiento externos, conectados al sistema como entradas del mismo.

Requerimientos no Funcionales

Módulos y Circuitos

La capacidad del condensador se obtiene mediante el conversor analógico/digital del microcontrolador, a través de lecturas de tensión de un circuito RC.

Asimismo, para poder mensurar inductancia de la bobina se hace uso de un circuito tanque y la obtención de la frecuencia resultante.

La interfaz de usuario debe refrescarse cada medio segundo.

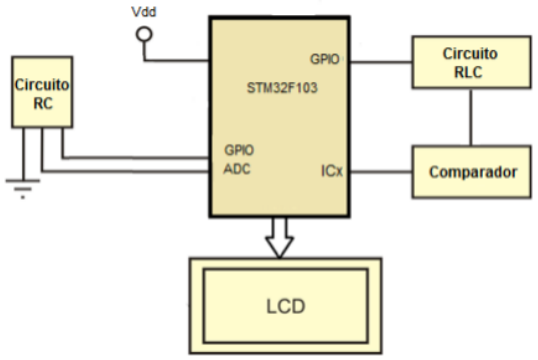


Figura 1: Esquema Simplificado del Sistema a Implementar

Herramientas Utilizadas

- Entorno de Desarrollo: uVision Keil ARM.
- Lenguajes de programación: C.
- Sistema de control de versiones: GIT.
- Familia de Microcontroladores: STM32F1.
- Microcontrolador: STM32F103C8.
- Software de automatización de diseño electrónico: Proteus 8.

Procedimientos de Prueba

El desarrollo del código fuente se realiza empleando el entorno de desarrollo uVision brindado por Keil, para dispositivos ARM. Asimismo, se emplea dicho entorno para la fase de depuración del código. La compilación del programa genera un archivo .HEX que se toma para la simulación del proyecto en Proteus 8.

Diseño del Software

El sistema se divide en módulos para facilitar la comprensión, depuración, mantenimiento y escalabilidad del mismo. Al igual que en trabajos anteriores, se divide el sistema en módulos de acuerdo a requerimientos específicos de cada uno.

Arquitectura del Sistema

Como se muestra en la **Figura 2**, el sistema se divide en capas. La capa inferior se compone de los módulos que se comunican con los periféricos. En este caso se requiere de un módulo para la medición de la capacitancia y otro para la inductancia. Ambos subsistemas excitan circuitos externos al microcontrolador mediante generación de pulsos y miden la respuesta del circuito a través de los temporizadores y convertidores. Los módulos encargados de la medición poseen el firmware para configurar y modificar el estado de los registros y las funciones de los pines de salida, y se les adiciona la extra para calcular en software los valores finales. Para la impresión, ambos medidores se sirven del módulo LCD para imprimir en pantalla.

Por ejemplo, los periféricos que se comunican con el L-Meter entregan una simple delta de tiempo entre dos flancos. Es el L-Meter el encargado de entender este valor como una frecuencia proveniente del circuito resonante, y a base de componentes con valores conocidos, encontrar la inductancia de la bobina, en principio incógnita.

Por último, el proyecto utiliza un sistema operativo cuyas funciones son recibir interrupciones sincrónicas (ticks del reloj del sistema) y asíncronas, y despachar las tareas que requieren atención, de acuerdo a interrupciones y los requerimientos temporales del proyecto.

Sistema Operativo

Un sistema operativo embebido simple (SEOS) realiza las tareas de planificación y despacho de las tareas que requieren el CPU.

El programa principal permanece en un bucle constante, y su única tarea es llamar al *dispatcher* del SEOS donde aquellas tareas indicadas son ejecutadas.

El encargado de marcar las tareas para ejecución es el *scheduler* o planificador, a través de flags. Este se ejecuta en cada interrupción del reloj del sistema, y cada una cantidad determinada de ticks, las tareas correspondientes de acuerdo a requerimientos temporales son señaladas.

Además de las interrupciones sincrónicas efectuadas por el reloj del sistema, los periféricos pueden generar interrupciones por hardware de forma asíncrona para solicitar uso de CPU en el próximo tick de reloj.

Módulos de Salida

Al igual que en entregas anteriores, la interfaz de usuario se efectúa a través de un display LCD de dos filas que le sirve a los módulos de obtención de datos, explicados en la sección anterior. Los tiempos de refresco fueron mencionados en los requerimientos no funcionales.

Medidores

Dos módulos se encargan de cuantificar las variables físicas necesarias. El módulo "RLC Meter" que busca obtener el valor de una bobina conectada por el usuario, funciona a través del envío de pulsos a un circuito tanque y evaluando la frecuencia de resonancia obtenida. El usuario también debe poder mensurar la carga de un capacitor. Esta última tarea es el objetivo del subsistema "RC Meter". Como el nombre lo indica, y a diferencia del que calcula la inductancia, utiliza únicamente una carga y un capacitor, en principio desconocido. Medir repetidamente el valor de la tensión del capacitor mediante el conversor analógico/digital del STM32F103 permite obtener la **constante de tiempo** y por ende la carga en faradios del condensador. Los medidores poseen tres tareas principales: medición, cálculo e impresión en pantalla.

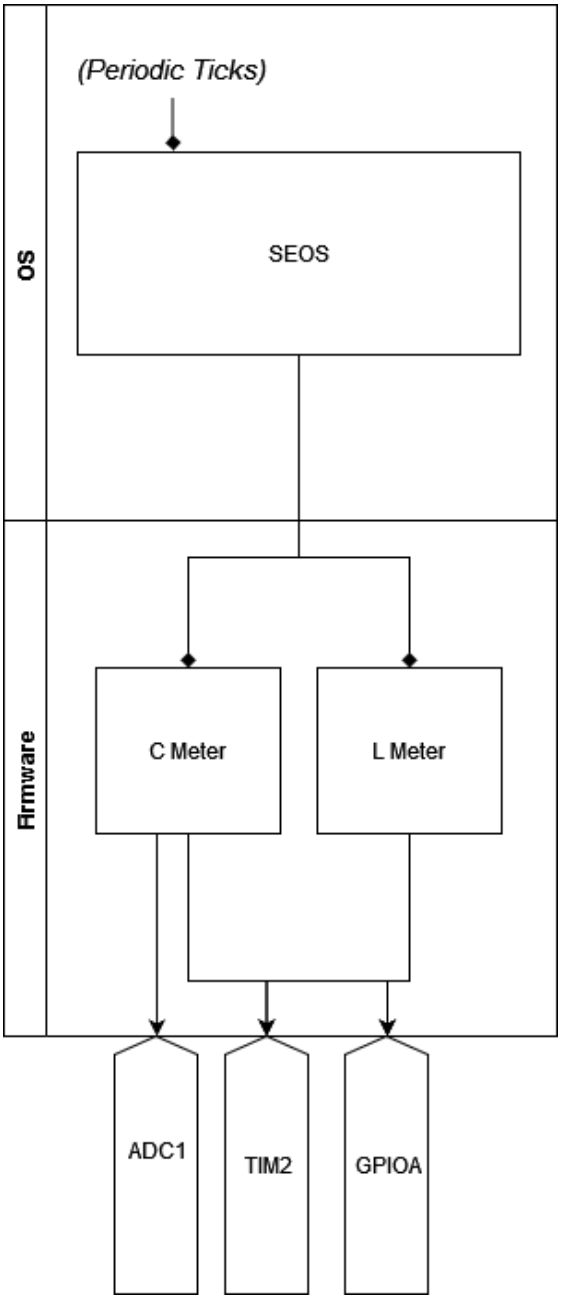


Figura 2: Arquitectura del Sistema. Se omite el módulo LCD, detallado en trabajos anteriores

Medidor de Capacidad

Tal y como es muestra en la Figura 1, para el medidor de capacidad se requiere emplear el conversor analógico digital del microcontrolador para leer la tensión del capacitor. Para obtener cambios de tensión en este, se excita el circuito mediante una señal seteada en el GPIOA.

Carga del Capacitor

Para obtener el valor de capacidad se requiere que las otras variables del circuito conectado al microcontrolador sean conocidas. Una de estas condiciones es el valor de la resistencia utilizada, y la otra es el valor de tensión con que se excitará; la misma será la tensión de señal de estado lógico alto de un pin del microcontrolador, aproximadamente 3,3 V. El pin permanecerá en alto el tiempo suficiente para cargarse al 63% de ese valor , o sea, 2,07 V, valor que le es conocido al software.

Lectura de Tensión del Capacitor

Para poder estimar la capacitancia del componente conectado al medidor, se requiere conocer el valor de la tensión y medir los tiempos de carga.

Constante de Tiempo de un Circuito RC

La tensión del condensador no cambia instantáneamente, por lo tanto, al cerrar el interruptor de la figura, la carga evoluciona con el tiempo en forma exponencial. La corriente inicialmente toma el valor Io=e/R y luego decrece exponencialmente con el tiempo.

Al producto RC se conoce como **constante de tiempo** del circuito, y equivale al tiempo que le lleva al condensador en cargarse de continuar en todo momento con la intensidad inicial I_0. También equivale al tiempo necesario para que el condensador se cargue con una carga que equivale al 63 % de la carga final, por lo cual si se obtiene el tiempo donde la carga es del 63% de la tensión de pin, al fijar R, se puede obtener C. Ver Figura 3.

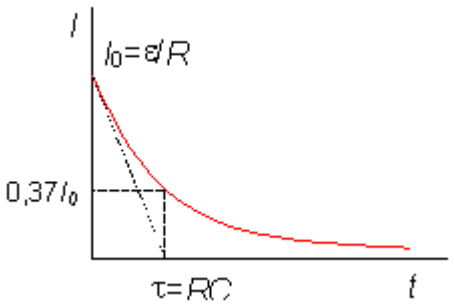


Figura 3: Constante de Tiempo en un Circuito RC

De esta manera, conocer la tensión del condensador nos permite el cálculo de la capacitancia. Para leer este valor, se utiliza el ADC (*analogic/digital converter*) del microcontrolador.

STM32F103 AD Converter

Como se describió anteriormente, se requiere tomar el delta de tiempo entre que el capacitor está descargado y recibe la excitación t_0 y el momento en el cual la carga es del 63% de 3,3 V, proporcionados por el pin del microcontrolador.

Es posible programar el ADC en modo interrupción para tomar este tiempo con mayor exactitud. Mientras se realizan otras tareas del sistema. Para la configuración del ADC se necesita:

1. Habilitar el clock del ADC1 y del puerto I/O mediante el registro RCC.
2. Configurar el puerto elegido como entrada analógica.
3. Encender el ADC1 con ADON=1.
4. Elegir el canal de muestreo.
5. Esperar 1 µs para que el ADC se estabilice.
6. De ser necesario, es posible habilitar las interrupciones del ADC con el bit EOCIE del registro CR1.
7. Elegir el canal de entrada.
8. Habilitar el modo conversión continua, mediante el bit CONT del registro de configuración CR2.
9. Iniciar la conversión con el bit ADON. Observar que setear este bit funciona para como encendido del ADC y para iniciar las conversiones, dependiendo de su valor inicial.
10. Si previamente fue configurado en modo interrupción, se debe habilitar el manejador de interrupciones ADC1_2.

En la simulación, se realiza el pulso, la espera de conversión y el cálculo de capacitancia en modo secuencial en una única función llamada RC_Measure, la cual es invocada cada medio segundo por el sistema operativo.

Modulo RC Meter

Inicialización

Como fue explicado anteriormente, este módulo requiere enviar un pulso a través del puerto A y medir la tensión del capacitor mediante un conversor analógico digital.

En la Figura 3 se muestra el código de inicialización. Se habilitan los relojes correspondientes, PA1 y PA5 se programan como entrada analógica para el ADC1 y salida digital para enviar los pulsos. El PA5 resetea su valor y se enciende el módulo ADC1, con una configuración de canales de sampleo trivial.

```
void RC_init(){
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

GPIOA->CRL=(GPIOA->CRL&0xFF3FFF0F)|0x00300000;
    GPIOA->BRR = (1<<5);
    ADC1->CR2 = 1;
    ADC1->SMPR2 = (0<<3);
    ADC1->SQR3 = 1;
    TIM2->ARR = 50000-1;
    LCD_goToXY(0, 0);
    LCD_sendString((uint8_t*) "C= ", 3);
}
```

Figura 3: Inicialización del medidor de capacidad.

Medicion de Tension y Calculo de C

La función de medición de tensión involucra el cálculo de la constante de tiempo y de la capacitancia. Una vez que se estabiliza el ADC1 con el valor inicial, se envia un pulso a través del pin y toma el tiempo inicial. Una vez que la tensión leída por el ADC alcanza el 62.3 % de la tensión de salida del pulso enviado, se toma la cantidad de ciclos del timer 2 que transcurrieron desde el inicio del proceso.

```
void RC_measure(){
    v = 0;
    TIM2->CNT = 0;
    ADC1->CR2 |= 1;
    while((ADC1->SR)&(1<<1) == 0);
    GPIOA->BSRR = (1<<5);
    delay_uss(1);
    TIM2->CR1 = 1;
    while(v < 2607) {
        ADC1->CR2 |= 1;
        while((ADC1->SR)&(1<<1) == 0);
        v = ADC1->DR; }
    cycles = TIM2->CNT;
    TIM2->CR1 = 0;
    tau = cycles / 72000000.0;
    capacitance = tau / RES;
    GPIOA->BRR = (1<<5);
    n_medida++; }
```

Figura 4: Función de medicion de capacidad

Finalmente, conocidos los ciclos y la carga, se obtiene la constante de tiempo en primera y la a través de esta última y la carga del circuito, se imprime la capacidad. Ver Figura 4. *Nota:* El código se encuentra comentado en el proyecto enviado adjunto al presente.

Impresión

Cada medio segundo aproximadamente, el sistema operativo llama al proceso de impresión. La función `sprintf` inserta una variable en una cadena de caracteres deseada. La misma es enviada para impresión sirviéndose de las funciones públicas del módulo LCD.

```
void RC_Print(void) {
    char* str;
    str = (char*) malloc(13*sizeof(char));
    if (capacitance > 0.001) {
        capacitance *= 1000;
        sprintf(str,"%7.2f mF",capacitance);
    } else if (capacitance > 0.000001) {
        capacitance *= 1000000;
        sprintf(str,"%7.2f uF",capacitance);
    } else {
        capacitance *= 1000000000;
        sprintf(str,"%7.2f nF",capacitance);
    }
    LCD_goToXY(3, 0);
```

```
    LCD_sendString((uint8_t*)str, 16);
}
```

Medidor de Inductancia

A diferencia del medidor de capacidad, en este caso se requiere la utilización de un timer para la obtención de la inductancia de la bobina a medir.

Circuito Tanque

El circuito de resonancia paralelo, conocido como **circuito tanque**, es una conexión en paralelo de una bobina y un condensador. Ver Figura 6.

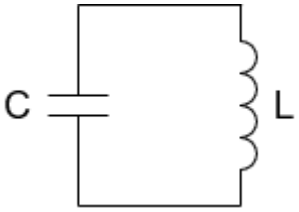


Figura 6: Circuito tanque de capacitor y bobina conectados en paralelo

Tiene la propiedad de que al ser alimentado entra en resonancia, generando una señal sinusoidal de **frecuencia fija**. Siendo la alimentación y demás componentes del circuito fijados en el sistema y conocidos, medir esta frecuencia permite obtener el valor de la inductancia de la bobina. La frecuencia de resonancia f_0 está determinada por

$$\omega_0 = \frac{1}{\sqrt{LC}}$$

Donde $\omega_0 = 2\pi f$, L es la inductancia de la bobina y C la capacidad del condensador. Despejando L:

$$L = (4\pi^2 f^2 C)^{-1}$$

Medición de Frecuencia

Como se explicó anteriormente, los timers de propósito general del STM32F103 pueden configurarse para cambiar los valores de salida de un pin o responder a cambios de la entrada para aplicaciones con tiempos más demandantes (*Input Capture*, *Output Compare*). La señal resonante es transformada en una onda rectangular de valores ALTO y BAJO para positivos y negativos obtenidos al excitar el circuito tanque. Para la carga de los componentes de almacenamiento se utiliza la misma lógica explicada en la sección Medidor de Capacidad, Carga del Capacitor.

Timer en Modo Input Capture

Los pines I/O son usados por el timer para capturar eventos de transición en las señales de entrada.

1. El bit CCnP del CCER permite detectar flancos ascendentes o descendentes.
2. El bit CCnS=01 configura el modo del canal como captura.
3. CCnE (Enabled) habilita la captura.

En cada captura, el flag CCnIF del TIMx_SR se pone en 1 y se mantiene hasta que es reseteado por software. Al leer TIMx_CCRn, CCnIF se limpia automáticamente por hardware.

Para configurar el timer en modo captura se deben seguir los siguientes pasos:

1. Habilitar el reloj para el pin GPIO como entrada y para el módulo TIMx.
2. Seleccionar el pin como entrada.
3. Configurar el prescaler del timer.
4. Seleccionar el modo de entrada con CCMR1 y CCMR2.
5. Seleccionar el tipo de flanco a notificar y habilitar la captura con CCER.
6. Configurar el valor del TIMx_ARR.
7. Habilitar el timer.
8. Esperar hasta que el bit CCNIF sea puesto en alto en el TIMx_SR.
9. Calcular el tiempo restando el valor actual del contador con el anterior.
10. Guardar el valor actual para el cálculo siguiente.

Módulos Involucrados

Al igual que en el medidor de capacitancia se utiliza una señal de salida del puerto A para cargar los elementos pasivos del sistema. La señal de salida del circuito tanque se toma a través del canal 3 del TIM2 configurado en modo *input-capture* para leer la frecuencia generada por el circuito tanque. A mayor nivel, las tareas de cálculo de la frecuencia a partir del periodo obtenido e impresión de los valores, se encuentra el módulo RLC_Meter.

En la Figura 4 se observa a modo ilustrativo, la forma de las señales que conforman el módulo de medición de inductancia. Inicialmente, el microcontrolador genera pulsos continuos para la carga y descarga del inductor y del capacitor del circuito tanque. La respuesta del mismo es una sinusoidal de frecuencia fija. Finalmente, para evitar que la señal senoidal y los valores de tensión negativa se conecten directamente al microcontrolador, se utiliza un comparador que transforma los valores positivos en 3,3 V y los negativos en 0 V, manteniendo la frecuencia buscada.

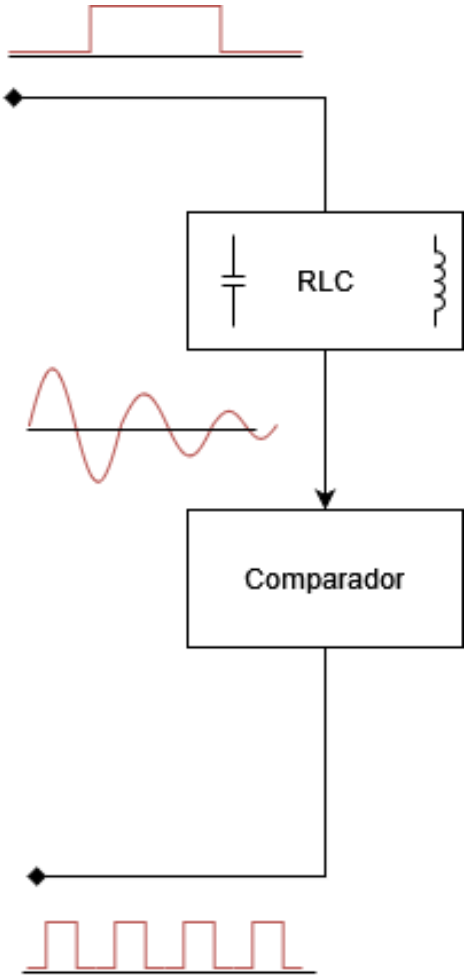


Figura 7: Señales a través del modulo RLC

Módulo RLC Meter

Al igual que el medidor de capacidad, el RLC se compone de tres funciones de inicialización, cálculo de inductancia e impresión del resultado.

Inicialización

Una vez inicializados los relojes de los periféricos utilizados, se habilita el detector de flancos del timer utilizado. Una vez finalizada la inicialización prepara al LCD para la impresión de valores.

```
void RLC_init() {

    n_measure = 0;

    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;

     RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; //
enable timer 2 clock

    GPIOA->CRL = (GPIOA->CRL & 0xF3FFF8FF) |
0x03000800;

    GPIOA->ODR |= (1<<2);

    TIM2->CCMR2 = 0x0001;

    TIM2->CCER = (1<<8);

    TIM2->PSC = PRESCALER;

    TIM2->ARR = 50000-1;

    LCD_goToXY(0, 1);

    LCD_sendString((char*) "L= ", 3);

}
```

Figura 7: Programa de inicialización del modulo RLC que calcula la inductancia de L.

Medición de F y Cálculo de L

Al iniciar la función comienza a correr el contador del timer, mientras que se envía un pulso a través del puerto 6 del GPIO. El detector de flancos del timer descarta la primera medida y aplica timestamps a partir de la segunda. Una vez detectados los flancos, la diferencia entre los tiempos determina el periodo de la onda generada por el comparador. Finalmente, se calcula por software la inductancia a partir de la frecuencia.

```
void RLC_measure() {

    uint32_t t0, t;

    TIM2->CNT = 0;

    TIM2->CR1 = 1;

    GPIOA->BSRR = (1<<6);

    while((TIM2->SR &(1<<3)) == 0);

    t0 = TIM2->CCR3;

    while((TIM2->SR &(1<<3)) == 0);

    t = TIM2->CCR3;

    Pe = (t - t0) / 72000000.0;

    F = 1 / Pe;

     inductance = 1 / ((2*PI*F) * (2*PI*F) *
CAP);

    TIM2->CR1 = 0;

    GPIOA->BRR = (1<<6);

    n_measure++;

}
```

Figura 8: Función que calcula la inductancia de L midiendo la diferencia entre dos 'timestamps'.

Diseño del Hardware

El diseño de hardware se realiza en el software Proteus 8; el objetivo es lograr un diseño de PBC con el cual poder llevar a cabo la fabricación artesanal o "en casa" de la placa.

Se comienza con el diseño del esquema eléctrico con todos los componentes. Los requerimientos especifican que se utilizará la placa Bluepill entera por lo que fue necesario crear un símbolo que la represente. Además de la Bluepill se debe contar con el LCD para mostrar los resultados y los dos circuitos que se van a medir para calcular la inductancia y capacitancia.

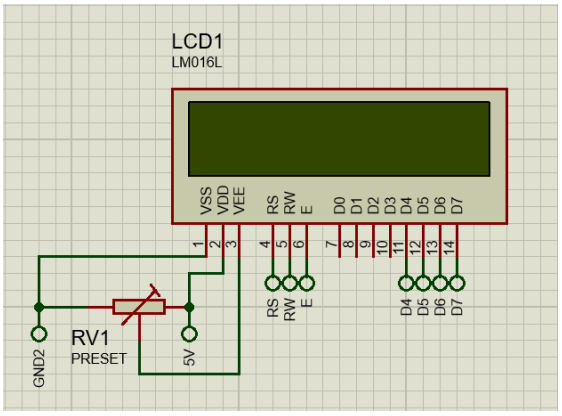


Figura 9: Conexionado del LCD.

Ya que la alimentación será a través de la Bluepill, por su puerto USB, se conectarán las tensiones y conexiones a tierra a los pines correspondientes de la misma para aquellos puntos de los circuitos que lo requieran. De esta forma será posible ejecutar las pista correctamente al diseñar el PBC. Para el LCD se requiere una alimentación de 5 V que se conectará al pin correspondiente de la Bluepill, una conexión a tierra y un preset (resistencia variable) que controla el contraste.

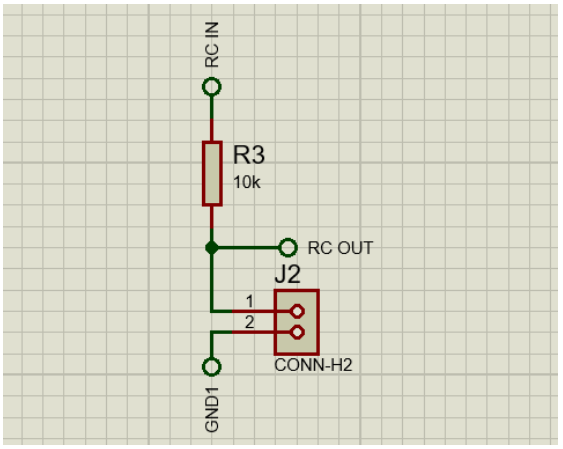


Figura 10: Esquema eléctrico del circuito RC.

El circuito RC consiste simplemente de una resistencia conectada entre un pin de salida de propósito general del la placa de desarrollo y un conector de dos pines en donde se conectará el capacitor a medir, el otro pin del conector está conectado a tierra. Para el circuito RLC es un poco más complicado, se tiene un conector, donde se conectará el inductor a medir, en paralelo con un capacitor; una resistencia se conecta entre un pin de la Bluepill y un extremo de este paralelo, el otro extremo se conecta a tierra. Como el circuito LC genera una onda sinusoidal bipolar que no se puede leer con el dispositivo de captura, ya que la placa no funciona con tensiones negativas. Se requiere el uso de un comparador conectado en forma de detector de cruce por cero unipolar de manera de convertir la señal sinusoidal en una señal cuadrada unipolar de la misma frecuencia; el comparador se alimenta con 3.3 V y tierra y requiere una resistencia de pull-up.

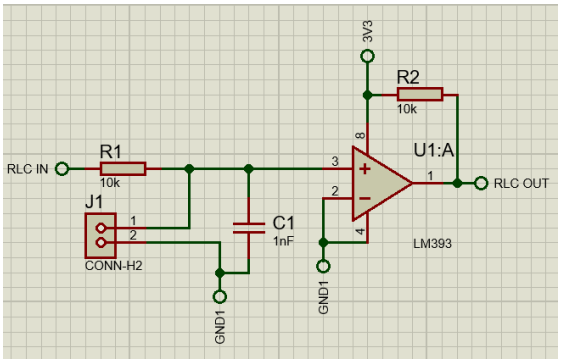


Figura 11: Esquema eléctrico del circuito RLC.

Una vez realizado el esquema eléctrico se puede comenzar a diseñar el PBC para ello se definieron las "reglas de diseño" siguiendo los requerimientos con pistas de 1 mm, vías de 1,8 mm de diámetro y agujero de 0,7 mm y la separación entre pistas de 1 mm. También se definió la placa a usar como simple faz con un tamaño de 10 x 10 cm. Para la Bluepill hubo que crear un empaquetado a medida para lograr la huella (footprint) correcta en el diseño.

En los valores utilizados se tiene en cuenta lo requerido por el proceso de fabricación objetivo; que consiste en imprimir el diseño en una impresora laser, traspasarlo al PCB mediante una plancha y luego atacar la placa con ácido cloruro férrico (la parte protegida por el tóner de la impresión será el único cobre que quede); Este proceso es susceptible a varios problemas como por ejemplo pistas cercanas que se unan porque no se disolvió correctamente una parte del cobre o pistas que se abran por roturas o separaciones en el tóner. Para mitigar la posibilidad de los mismos y facilitar la solución y construcción a Mano es que las pistas y las separaciones se eligen más amplias que si lo fuera a hacer un fabricante.

Con todo definido se disponen los componentes en la capa de componentes (top layer) y se comienza a intentar conectar todo de manera correcta. Hasta este momento se venía utilizando el LCD en modo 8 bits, pero luego de mucho probar, se llegó a la conclusión de que no era posible realizar todas las conexiones para el mismo en el diseño simple faz; entonces se volvió al esquema eléctrico a cambiar la configuración de las conexiones por la requerida para el LCD en 4 bit y se realizó los cambios en el software para permitir el funcionamiento en este modo; de esta manera se consiguió un conexionado satisfactorio.

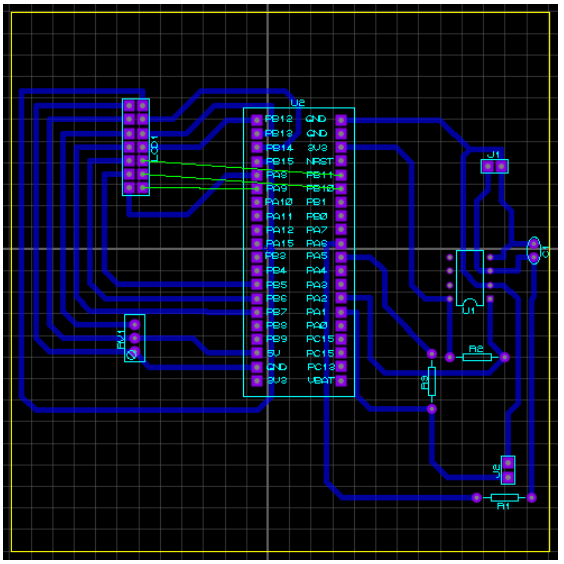
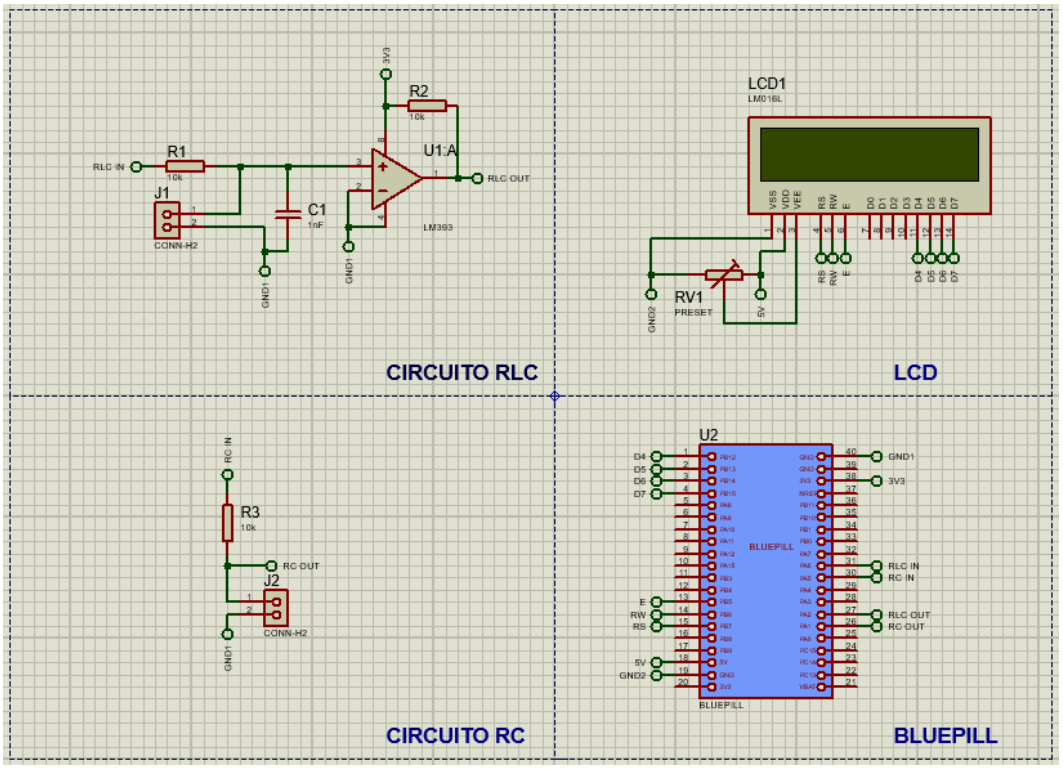


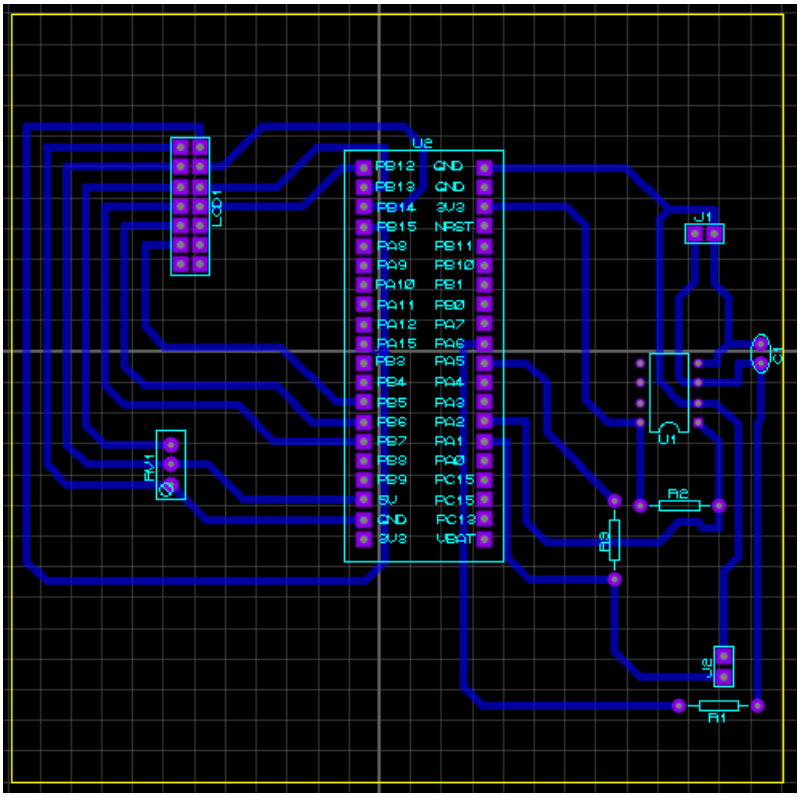
Figura 12: Trazado de conexiones con el LCD en modo 8 bits, se puede observar las conexiones que faltan porque se cruzarían con otras pistas.

Vistas:

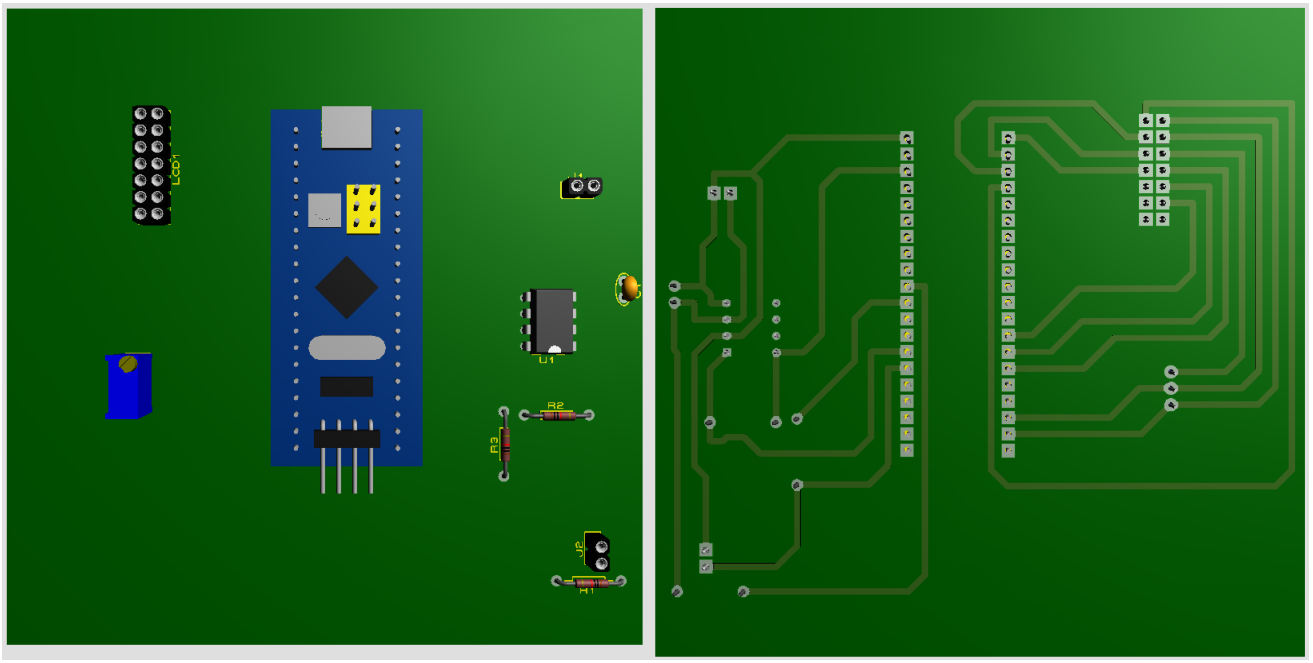
Diagrama eléctrico:

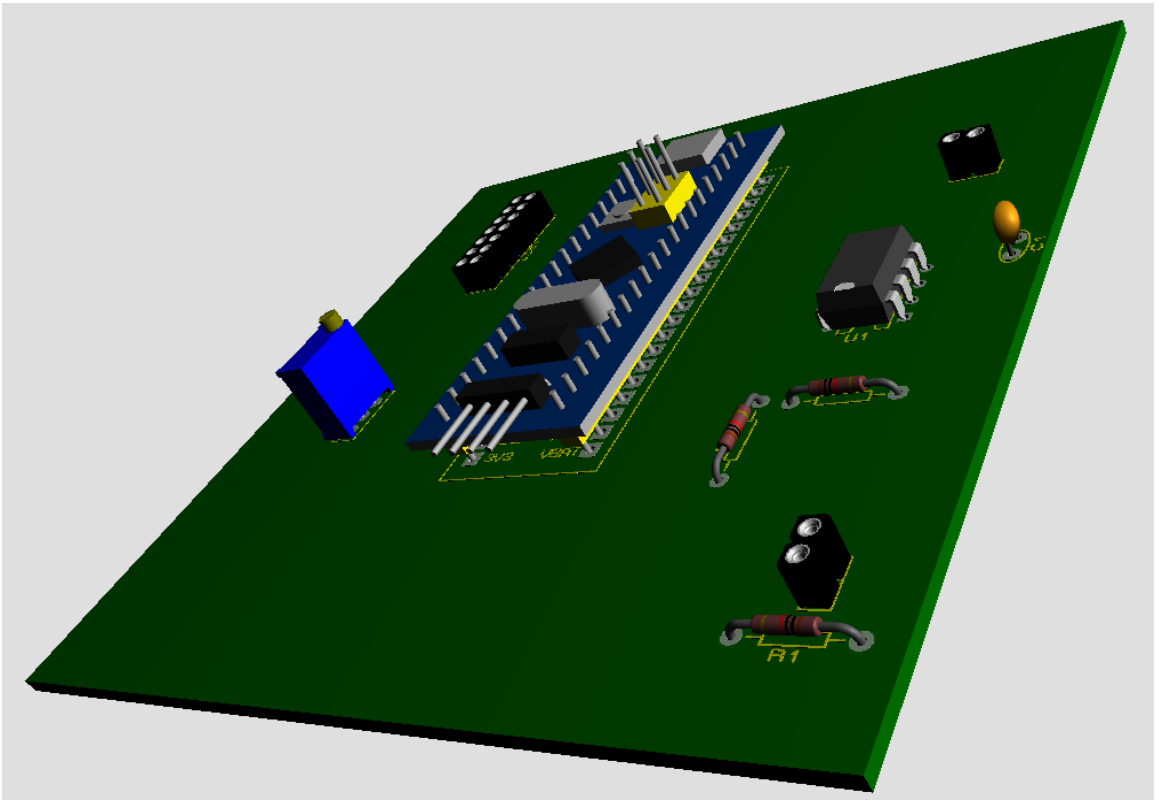


Diseño PBC:



3D:



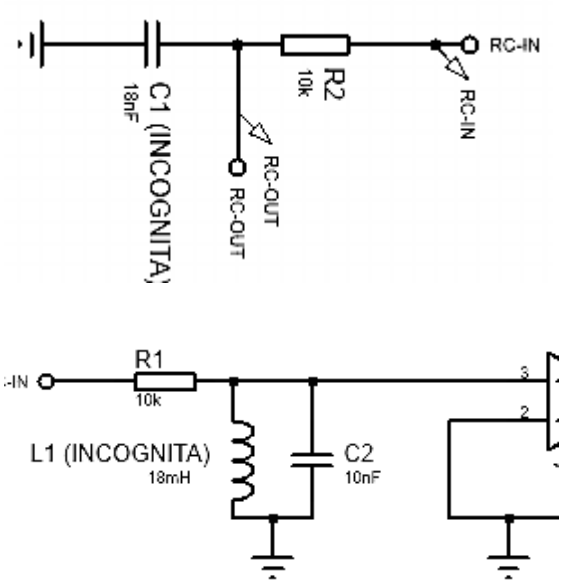


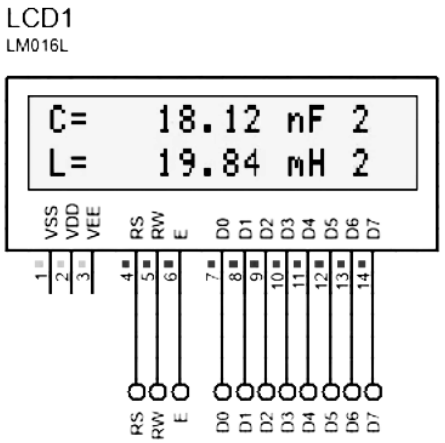
Validación

Utilizando el esquema en Proteus (ver sección Proteus Schematic) se realizó un video de simulación explicando el funcionamiento detallando el código explicando en el informe. A través de las opciones de debugging fue posible detener la simulación en tiempos claves cuyos valores se aproximan a los resultados de ecuaciones de constante de tiempo y de frecuencia obtenidas de forma analítica. Las Figuras a continuación muestran los resultados de la simulación empleando un condensador y un inductor de 18 nF y 18 mH.

Resultados

El LCD de la Figura a continuación, muestra 18.12 nF y 19.84 en la segunda medición (número a la derecha en cada fila del LCD). Los valores de C y L obtenidos representan errores cercanos al 1 y 10 %, respectivamente.





n_measure	20000018	2
inductance	2000001C	19.8393
Pe	20000020	8.85e-05
F	20000024	11299.4
v	20000004	2608
cycles	20000008	13049
n_medida	2000000C	2
capacitance	20000010	18.1236
tau	20000014	0.000181236
SystemCoreC...	20000000	72000000

Como se explicó anteriormente, el sistema mide el tiempo entre dos flancos de subida y lo almacena como `Pe` (periodo) en el módulo LRC. Mientras que medidor RC obtiene la cantidad de ciclos del timer (`cycles`) que le toma a la tensión (`v`) del capacitor llegar de 0 al 62.3 % de los 3.3 V de salida del pin.

Enlaces al Video de Simulación

- https://youtu.be/gpF3Y_Ku_Wk

Apéndice A: Temporizadores

Los temporizadores o **timers** del STM32F10x se pueden dividir en básicos, de propósito general (Timer 2, 3 y 4) y de control avanzado (Timer 1). No debe confundirse el tick del sistema operativo, generado por el clock del core ARM con los temporizadores del STM32F103, cuyas funciones están relacionadas la entrada/salida.

Registros

CNT (Counter)

El registro CNT (counter) es un contador up/down de 16 bits cuyo valor se lee desde TIMx_CNT.

ARR (Auto Reload Register)

Registro de 16-bits. Su valor se compara continuamente con TIMx_CNT.

TIMx_CR1 (Control Register)

Modos *up, down* y *yoyo*. Funciones de habilitación, auto-reload, modo continuo o pulso unico, entre otras.

TIMx_SR (Status Register)

Posee el flag de interrupción. En el modo counting-up, cuando CNT alcanza ARR el flag es seteado y CNT vuelve a cero.

Salida: Generador de Ondas

Cuando la aplicación requiere un control de la salida digital con tiempos precisos, se utilizan los timers del microcontrolador. El STM32 posee cuatro canales por cada temporizador. Cada canal tiene un circuito de salida y uno de entrada para los modos *input-capturing* y *output-compare*.

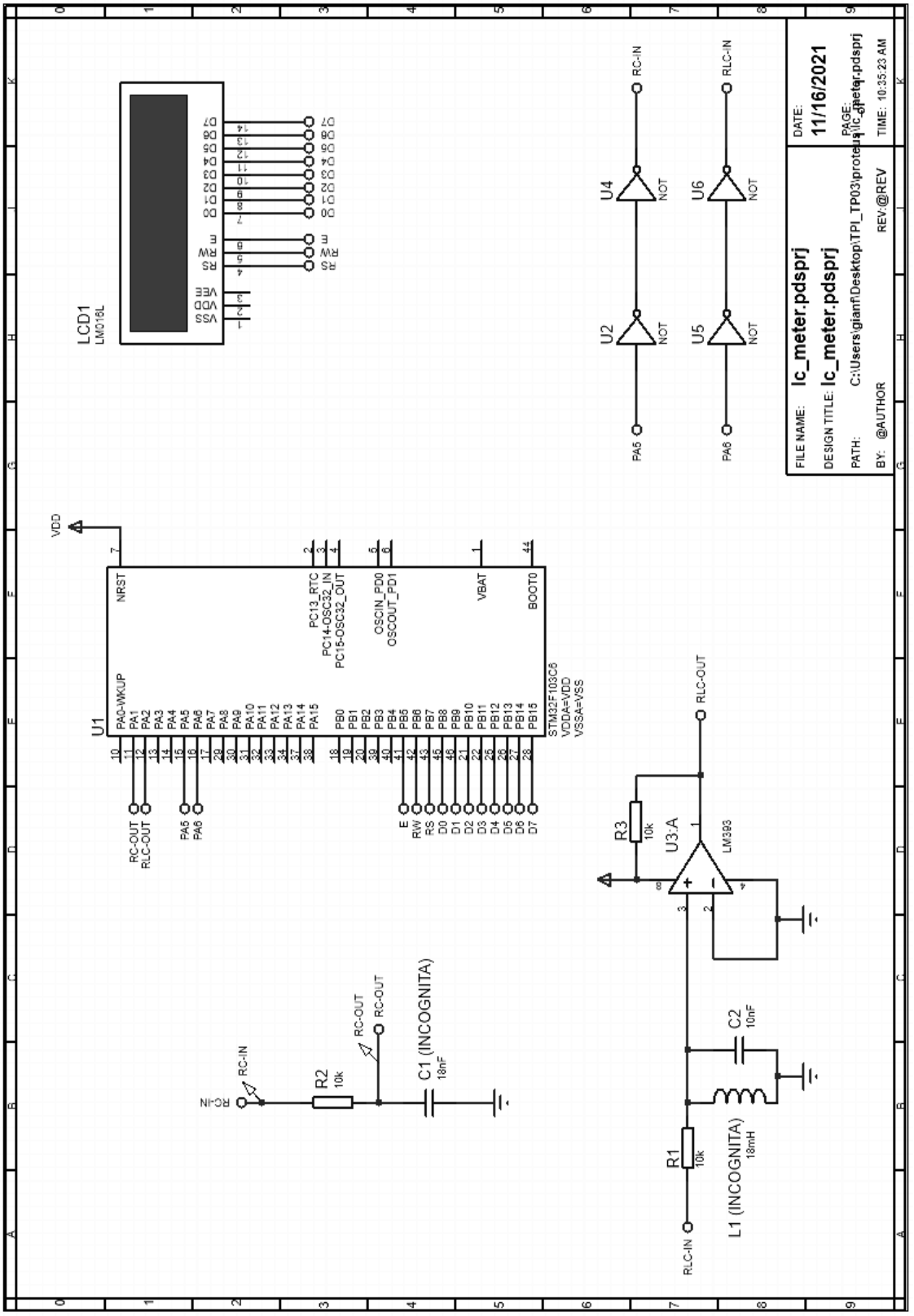
TIMx_CCRn (Capture/Compare Register)

- Registro de lectura y escritura
- TIMx_CCRn de cada canal se compara con TIMx_CNT después de cada ciclo de reloj y el flag CCnOF es puesto en alto cuando sus valores coinciden.
- Hay un generador de ondas en cada canal, lo que posibilita cambiar el estado del pin de ALTO a BAJO y viceversa (*toggling).*

TIMx_CCER (Capture/Compare Enable Register)

Cuando CCnP=0 la salida del generador va directamente al pin GPIO, mientras que CCnP=1 invierte la señal de salida y la dirige al GPIO.

Proteus Schematic



Apéndice B: Código.

seos.h

```
#ifndef _SEOS_H

#define _SEOS_H


#include <stm32f103x6.h>

#include <inttypes.h>

#include <stdio.h>


/* Overflow */

/* nro de tick del sistema en el que se debe ejecutar la tarea y reiniciar el
contador */

#define INIT_MEASURE 3

#define INIT_PRINT 0

#define OVERF_MEASURE 5 // 0.5 s

#define OVERF_PRINT 5 // 0.5 s


/* Public Functions */

int SEOS_Boot(void);


int SEOS_Schedule(void);


int SEOS_Dispatch(void);


int SEOS_Sleep(void);


#endif
```

seos.c

```
#include "seos.h"
```

```c
#include "rlc.h"
#include "rc.h"
#include "lcd.h"

/* PUBLIC FLAGS */
static volatile uint32_t Flag_measure;
static volatile uint32_t Flag_print;

/* PRIVATE */
int seos_init(void);

static volatile uint8_t counter_measure;
static volatile uint8_t counter_print;

void SysTick_Handler(void) {
    SEOS_Schedule();
}

int seos_init(void)
{
    Flag_measure  = 0;
    Flag_print = 0;
    //se inicializan los flags y contadores
    counter_measure = 4;
    counter_print = 0;

    //se configura el sistem tick para interrumpir una vez cada 100 ms
    if (SysTick_Config(SystemCoreClock / 10)){
        //error handling
    }
```

```
    return 0;
}
int SEOS_Boot(void) {
    //inicializa todos los módulos
    LCD_init();
    RLC_init();
    RC_init();
    seos_init();
    return 0;
}
int SEOS_Schedule(void)
{
      //el planificador levanta el flag de las tareas que se tengan que
despachar
    if(++counter_measure == OVERF_MEASURE)
    {
        Flag_measure   = 1;
        counter_measure = 0;
    }
    if(++counter_print == OVERF_PRINT)
    {
        Flag_print  = 1;
        counter_print= 0;
    }
    return 0;
}

int SEOS_Dispatch(void)
{
    //el despachador ejecuta las tareas que están pendientes y baja los flags
    if(Flag_measure) {
        RLC_measure();
```

```
        RC_measure();
        Flag_measure = 0;
    }
    if(Flag_print){
        RLC_print();
        RC_Print();
        Flag_print = 0;
    }
    return 0;
}

int SEOS_Sleep(void)
{
    //sleep no se implementa en la simulación
    return 0;
}
```

rc.h

```
#ifndef RC_H
#define RC_H

#define RES 10000

void RC_init(void);
void RC_Print(void);
void RC_measure(void);

#endif
```

rc.c

```
#include "rc.h"
#include "lcd.h"
#include <stm32f103x6.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

volatile uint32_t v = 0, cycles = 0, n_medida = 0;
volatile float capacitance, tau;

void delay_uss(uint16_t);
void RC_print(void);

void RC_init() {
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; // enable PORT A clock
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // enable ADC1 clock
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // enable timer 2 clock
    RCC->CFGR &= 0xFFFF3FFF;
     GPIOA->CRL = (GPIOA->CRL & 0xFF3FFF0F) | 0x00300000; // PA1 analog input,
PA5 GP output
    GPIOA->BRR = (1<<5);
    ADC1->CR2 = 1; // ADC1 on
      ADC1->SMPR2 = (0<<3); // SMPR2->SMP1 = 001 => Sample time Ch1 7.5 adc
cycles
    ADC1->SQR3 = 1; // Channel 1
    TIM2->ARR = 50000-1;
    LCD_goToXY(0, 0);
    LCD_sendString((uint8_t*) "C= ", 3);
}
```

```
void RC_measure() {
    v = 0;
    TIM2->CNT = 0; // Reset counter
    //TIM2->CR1 = 1; // Start counting
    ADC1->CR2 |= 1; // Start conversion
    while((ADC1->SR) & (1<<1) == 0); // wait for EOC
    delay_uss(1);
    GPIOA->BSRR = (1<<5); // Raise PA5
    delay_uss(1);
    TIM2->CR1 = 1; // Start counting
    while(v < 2607) { // wait to reach 62.3% of 3V3
        ADC1->CR2 |= 1; // Start conversion
        while((ADC1->SR) & (1<<1) == 0); // wait for EOC
        v = ADC1->DR;
    }
    cycles = TIM2->CNT;
    TIM2->CR1 = 0; // Stop timer counter
    //tau = cycles / 10000.0;
    tau = cycles / 72000000.0;
    capacitance = tau / RES;
    GPIOA->BRR = (1<<5);
    n_medida++;
}

void RC_Print(void) {
    char* str;
    str = (char*) malloc(13*sizeof(char));
    if (capacitance > 0.001) {
        capacitance *= 1000;
        sprintf(str, "%7.2f mF %d", capacitance, n_medida);
    } else if (capacitance > 0.000001) {
```

```
        capacitance *= 1000000;
        sprintf(str, "%7.2f uF %d", capacitance, n_medida);
    } else {
        capacitance *= 1000000000;
        sprintf(str, "%7.2f nF %d", capacitance, n_medida);
    }
    LCD_goToXY(3, 0);
    LCD_sendString((uint8_t*)str, 16);
}

void delay_uss(uint16_t t) {
    uint16_t l, i;
    for(i = 0; i < t; i++)
        for(l = 0; l < 12; l++){}
}
```

rlc.h

```
#ifndef RLC_H
#define RLC_H

#define CAP 0.000000010F

#define PRESCALER 7199

void RLC_init(void);
void RLC_measure(void);

#endif
```

rlc.c

```
#include "rlc.h"
#include "lcd.h"
#include <stm32f103x6.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#define PI 3.141592654F

float inductance, Pe, F;
uint16_t n_measure;

void RLC_print(float);

void RLC_init() {
    n_measure = 0;
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; // enable PORT A clock
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; // enable timer 2 clock
     GPIOA->CRL = (GPIOA->CRL & 0xF3FFF8FF) | 0x03000800; // PA2 (TIM2 CH3):
input pull-up/down - PA6 GP output
    GPIOA->ODR |= (1<<2); // Pull-up
     TIM2->CCMR2 = 0x0001;  // Pin TIM2_CH3 as input for channel 3 - prescaler
1 - filter 1
    TIM2->CCER = (1<<8); // CC3P = 0 (rising edge), CC3E = 1 enable
    TIM2->PSC = PRESCALER; // prescaller 7200 - 72MHz/7200 = 10kHz
    TIM2->ARR = 50000-1;
    LCD_goToXY(0, 1);
    LCD_sendString((char*) "L= ", 3);
}
```

```
void RLC_measure() {
    //float Pe, F;
    uint32_t t0, t;
    TIM2->CNT = 0;
    TIM2->CR1 = 1; // start counting up
    GPIOA->BSRR = (1<<6); //raise PA6
    while((TIM2->SR &(1<<3)) == 0); // wait until the CC3IF flag sets
    t0 = TIM2->CCR3; // gets initial count
    while((TIM2->SR &(1<<3)) == 0); // wait until the CC3IF flag sets
    t = TIM2->CCR3; // gets final count
    Pe = (t - t0) / 72000000.0; // contador sobre frecuencia de contador
    F = 1 / Pe; // frecuencia = 1 / periodo
    inductance = 1 / ((2*PI*F) * (2*PI*F) * CAP); // L = 1/F^2*C
    TIM2->CR1 = 0; // stops counting
    GPIOA->BRR = (1<<6);
    n_measure++;
}
void RLC_print(float L) {
    char *str;
    str = (char*) malloc(13*sizeof(char));
    if (inductance > 0.001) {
        inductance *= 1000;
        sprintf(str, "%7.2f mH %d", inductance, n_measure);
    } else if (inductance > 0.000001) {
        inductance *= 1000000;
        sprintf(str, "%7.2f uH %d", inductance, n_measure);
    } else {
        inductance *= 1000000000;
        sprintf(str, "%7.2f nH %d", inductance, n_measure);
    }
    LCD_goToXY(3, 1); LCD_sendString(str, 13); }
```

lcd.h

```
#ifndef _LCD_H
#define _LCD_H

#include <stm32f103x6.h>                  // Device header

#include <stdint.h>

#include "lcd.h"

#define LCD_4bit
#define LCDPORT GPIOB //puerto al que se conecta el lcd
#define LCDIOCLK RCC_APB2ENR_IOPBEN

#define LCD_D0  8   //pin conectado a LCD D0
#define LCD_D1  9   //pin conectado a LCD D1
#define LCD_D2  10  //pin conectado a LCD D2
#define LCD_D3  11  //pin conectado a LCD D3
#define LCD_D4  12  //pin conectado a LCD D4
#define LCD_D5  13  //pin conectado a LCD D5
#define LCD_D6  14  //pin conectado a LCD D6
#define LCD_D7  15  //pin conectado a LCD D7
#define LCD_RW  6 //pin conectado a LCD R/W
#define LCD_RS  7 //pin conectado a LCD RS
#define LCD_E   5   //pin conectado a LCD E

void LCD_init(void);
void LCD_sendCommand(uint8_t);
void LCD_goToXY(uint8_t, uint8_t);
void LCD_sendChar(uint8_t);
void LCD_sendString(uint8_t*, uint8_t);
```

```
void LCD_clear(void);


#endif
```

lcd.c

```
#include "lcd.h"


// cursor position to DDRAM mapping
#define LCD_LINE0_DDRAMADDR     0x00
#define LCD_LINE1_DDRAMADDR     0x40


void delay_us(uint16_t);


void LCD_init(void) {
    // esperar 15ms luego de encender para inicializar el lcd
    delay_us(15000);
    // activa el clock del puerto del lcd
    RCC->APB2ENR |= LCDIOCLK;
#ifdef LCD_4bit
    // establece todos los pines del lcd como salida
       LCDPORT->ODR  &=  ~(1<<LCD_D4  |  1<<LCD_D5  |  1<<LCD_D6  |  1<<LCD_D7  |
1<<LCD_RW | 1<<LCD_RS | 1<<LCD_E);
      LCDPORT->CRH  |=  3<<(LCD_D4-8)*4  |  3<<(LCD_D5-8)*4  |  3<<(LCD_D6-8)*4  |
3<<(LCD_D7-8)*4;
    LCDPORT->CRH &= ~(12<<(LCD_D4-8)*4 | 12<<(LCD_D5-8)*4 | 12<<(LCD_D6-8)*4 |
12<<(LCD_D7-8)*4);
    LCDPORT->CRL |= 3<<LCD_RW*4 | 3<<LCD_RS*4 | 3<<LCD_E*4;
    LCDPORT->CRL &= ~(12<<LCD_RW*4 | 12<<LCD_RS*4 | 12<<LCD_E*4);
    delay_us(4200);
    LCD_sendCommand(0x33);
    LCD_sendCommand(0x32);
    LCD_sendCommand(0x28);
    LCD_sendCommand(0x0E);
```

```
    LCD_sendCommand(0x01);

    delay_us(2000);

    LCD_sendCommand(0x06);

#else

    // establece todos los pines del lcd como salida

       LCDPORT->ODR  &=  ~(1<<LCD_D0  |  1<<LCD_D1  |  1<<LCD_D2  |  1<<LCD_D3  |
1<<LCD_D4  |  1<<LCD_D5  |  1<<LCD_D6  |  1<<LCD_D7  |  1<<LCD_RW  |  1<<LCD_RS  |
1<<LCD_E);

      LCDPORT->CRH  |=  3<<(LCD_D0-8)*4  |  3<<(LCD_D1-8)*4  |  3<<(LCD_D2-8)*4  |
3<<(LCD_D3-8)*4  |  3<<(LCD_D4-8)*4  |  3<<(LCD_D5-8)*4  |  3<<(LCD_D6-8)*4  |
3<<(LCD_D7-8)*4;

    LCDPORT->CRH &= ~(12<<(LCD_D0-8)*4 | 12<<(LCD_D1-8)*4 | 12<<(LCD_D2-8)*4 |
12<<(LCD_D3-8)*4  |  12<<(LCD_D4-8)*4  |  12<<(LCD_D5-8)*4  |  12<<(LCD_D6-8)*4  |
12<<(LCD_D7-8)*4);

    LCDPORT->CRL |= 3<<LCD_RW*4 | 3<<LCD_RS*4 | 3<<LCD_E*4;

    LCDPORT->CRL &= ~(12<<LCD_RW*4 | 12<<LCD_RS*4 | 12<<LCD_E*4);

    delay_us(4200);

    LCD_sendCommand(0x38);

    LCD_sendCommand(0x0E);


    LCD_sendCommand(0x01);

    delay_us(2000);

    LCD_sendCommand(0x06);

#endif

}


void LCD_sendCommand(uint8_t cmd){

    LCDPORT->ODR &= ~(1<<LCD_RS);

#ifdef LCD_4bit

    LCDPORT->ODR |= (cmd&0xF0)<<8;

    LCDPORT->ODR &= (0x0FFF|((cmd&0xF0)<<8));

    LCDPORT->ODR |= 1<<LCD_E;

    delay_us(2);

    LCDPORT->ODR &= ~(1<<LCD_E);
```

```
    delay_us(100);
    LCDPORT->ODR |= (cmd&0x0F)<<12;
    LCDPORT->ODR &= (0x0FFF|((cmd&0x0F)<<12));
#else
    LCDPORT->ODR |= cmd<<8;
    LCDPORT->ODR &= (0xFFFF&(0x00FF|(cmd<<8)));
#endif
    LCDPORT->ODR |= 1<<LCD_E;
    delay_us(2);
    LCDPORT->ODR &= ~(1<<LCD_E);
    if ((cmd == 1) | (cmd == 2) | (cmd == 3))
        delay_us(2000);
    else
        delay_us(100);
}

void LCD_sendChar(uint8_t c){
    LCDPORT->ODR |= 1<<LCD_RS;
#ifdef LCD_4bit
    LCDPORT->ODR |= (c&0xF0)<<8;
    LCDPORT->ODR &= (0x0FFF|((c&0xF0)<<8));
    LCDPORT->ODR |= 1<<LCD_E;
    delay_us(2);
    LCDPORT->ODR &= ~(1<<LCD_E);
    LCDPORT->ODR |= (c&0x0F)<<12;
    LCDPORT->ODR &= (0x0FFF|((c&0x0F)<<12));
#else
    LCDPORT->ODR |= c<<8;
    LCDPORT->ODR &= (0xFFFF&(0x00FF|(c<<8)));
#endif
    LCDPORT->ODR |= 1<<LCD_E;
```

```
    delay_us(2);
    LCDPORT->ODR &= ~(1<<LCD_E);
    delay_us(100);
    LCDPORT->ODR &= ~(1<<LCD_RS);
}

void LCD_sendString(uint8_t* string, uint8_t length){
    uint8_t i ;
    for (i= 0; i < length; i++) {
        LCD_sendChar(string[i]);
    }
}

void LCD_goToXY(uint8_t x, uint8_t y) {
    register uint8_t DDRAMAddr;
    // remap lines into proper order
    switch(y) {
        case 0: DDRAMAddr = LCD_LINE0_DDRAMADDR+x; break;
        case 1: DDRAMAddr = LCD_LINE1_DDRAMADDR+x; break;
        default: DDRAMAddr = LCD_LINE0_DDRAMADDR+x;
    }
    // set data address
    LCD_sendCommand(1<<7 | DDRAMAddr);
}

void delay_us(uint16_t t) {
    uint16_t l, i;
    for(i = 0; i < t; i++)
        for(l = 0; l < 12; l++){}
}
```

Bibliografía y Referencias

[1] "The STM32F103 Arm Microcontroller and Embedded Systems, Using Assembly and C, First Edition", Ali Mazidi M., Naimi S., Naimi S.

[2] "Differential Equations, with Boundary-Value Problems, Eight Edition", Zill D., Wright W.

[3] "Circuitos Electricos, Sexta Edicion" Dorf, Svoboda.

[4] "GLOSARIO: Conceptos Generales Sobre Circuitos Impresos" Brengi D., 2018.

[5] "La Suite Proteus: Primeros Pasos con la Pestana Diseno", Labcenter Electronics, 2019.

[6] "LMx93-N, LM2903-N, Low-Power, Low-Offset Voltage, Dual Comparators" Texas Instruments, 2018.

[7] "LM2903, LM393/LM393A, LM293A Dual Differential Comparator" Fairchild Semiconductor, 2012.

[8] "Mastering STM32, A step-y-step guide to the most complete ARM Cortex-M platform, using a free and powerfull development environment based on Eclipse and GCC" Noviello C., 2016.