



# Universidad Nacional de Córdoba

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

## ELECTRÓNICA DIGITAL III PROYECTO FINAL INTEGRADOR

Autores:

*Alaniz, Darío  
Ferraris, Domingo  
Gomez, Augusto*

Docentes:

*Ing. Gallardo, Fernando  
Ing. Sanchez, Julio*

6 de noviembre de 2021

# 1. Objetivo

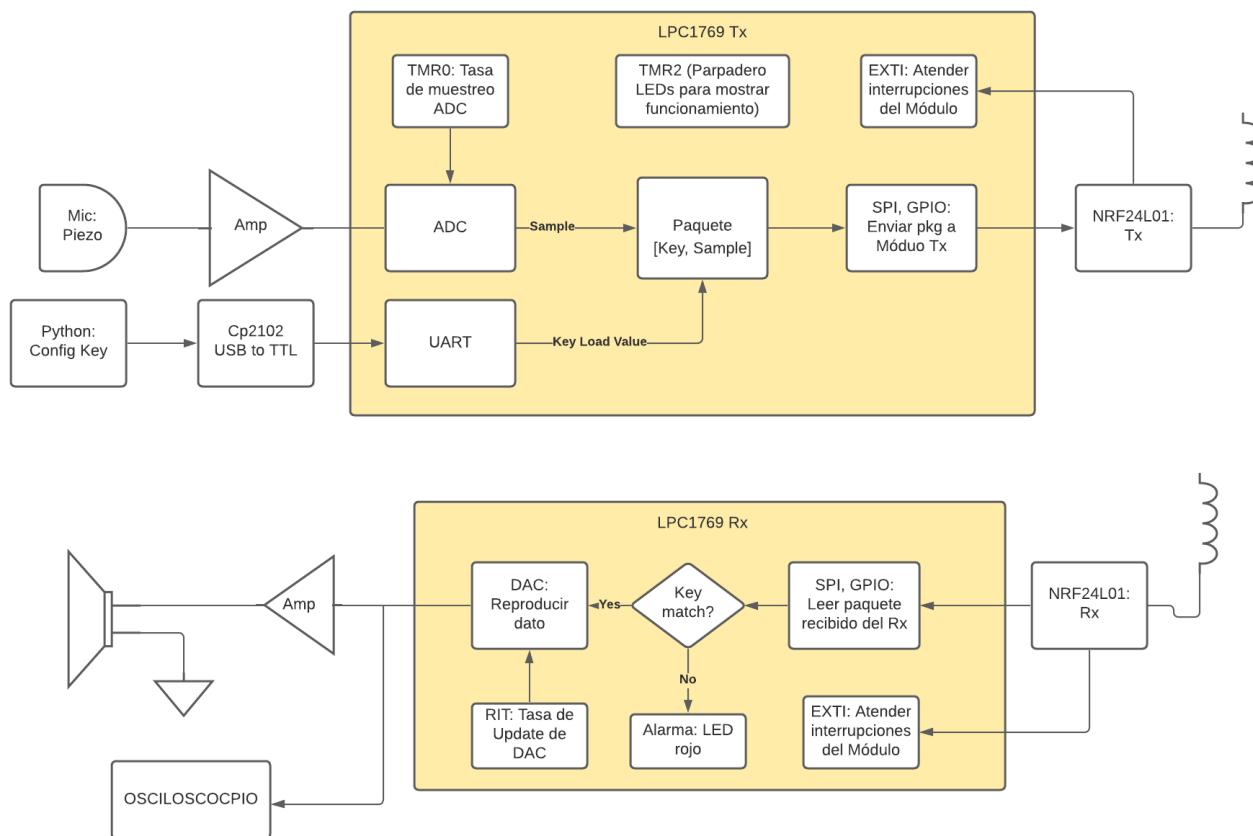
Como parte del proyecto final de la materia, se decidió establecer una comunicación RF entre dos placas *LPC1769* de la firma *NXP*.

Una de ellas es el dispositivo transmisor, encargado de muestrear la señal analógica proveniente de un micrófono mediante el módulo ADC, y enviar dichos datos haciendo uso de los módulos RF *nRF24L02* hacia el receptor, además, cuenta con la capacidad de tener una clave configurable a través de una comunicación UART con la computadora (script en *Python*), los datos son empaquetados con la clave elegida como cabecera.

El dispositivo receptor es quien recibe esos paquetes con información, reconstruyendo la señal analógica a la salida del DAC. Esto sólo se dará si tanto el receptor como el emisor comparten la misma clave, de lo contrario este último mostrará una señal de alarma a través de LEDs.

# 2. Diagrama de bloques

A continuación se muestra un diagrama con los bloques fundamentales utilizados para la aplicación propuesta:



### 3. Desarrollo

#### 3.1. ADC y TMR0

Dado que se muestran señales de voz y que se planeó una aplicación lo más cercano a tiempo real se decidió elegir una tasa de muestreo baja  $F_s = 3kHz$ . Además debido al ruido presente en las muestras por el propio conversor, se implementó un filtro promediador de  $N = 8$  muestras.

El inicio de la conversión está regido por la transición del bit MAT0.1 de alto a bajo (flanco de bajada) del Timer0, es decir que trabaja en modo Match, teniendo en cuenta lo mencionado anteriormente, para lograr el muestro deseado la frecuencia del con la que el timer alcanza el valor de MAT1 es:

$$\begin{aligned} F_{MR1} &= 2 N F_s \\ F_{MR1} &= 2 * 8 * 3kHz \\ F_{MR1} &= 48kHz \end{aligned}$$

En la práctica se logró una tasa de muestreo algo menor, debido a que no se puede alcanzar exactamente el valor de  $F_{MR1}$ :

$$\begin{aligned} F'_s &= 2,976kHz \\ F'_{MR1} &= 47,619kHz \end{aligned}$$

#### 3.2. UART

Configuración general:

- $baudRate = 115200$  (114882 en la práctica)
- $bitPerChar = 8$
- Paridad desactivada
- $stopBit = 1$

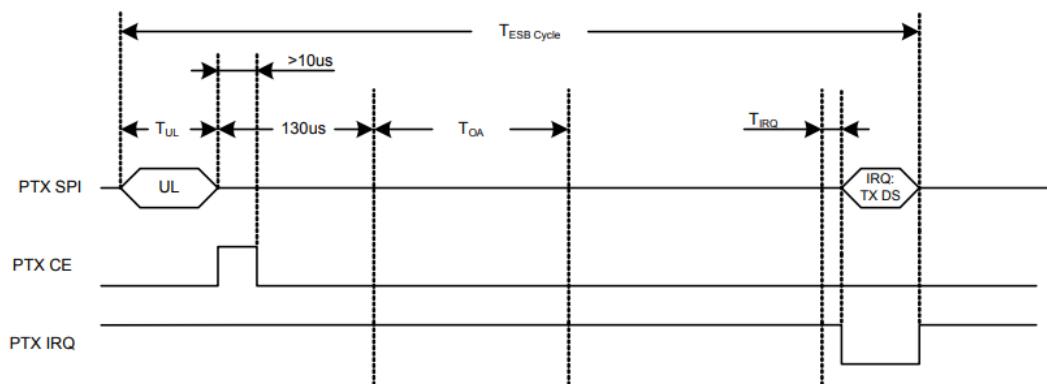
#### 3.3. SPI

Necesario para configurar el microcontrolador del módulo RF nRF24L01, se muestra la configuración básica:

- $SPI_{Clk} = 2MHz$
- $bitPerChar = 8$
- $CPHA = 1$  (flanco asc. dato estable, flanco desc. transición de dato)
- $CPOL = 0$

### 3.4. GPIO, EXTI y TMR1

Estos dos módulos también son necesarios para interactuar con el módulo RF, ya que este una vez configurado (ya sea como transmisor o receptor), debe ser sacado de modo Stand By poniendo en alto el PIN CE del mismo por lo menos  $10 \mu s$ , esto es llevado a cabo mediante un pin configurado como GPIO y la temporización usando el Timer1. Cuando una transacción de datos fue exitosa (o por el contrario que se haya perdido un paquete) el módulo envía una señal de interrupción activa por bajo a través del pin IRQn, entonces se configura un pin como interrupción externa por flanco descendente para notificar al MCU, que luego mediante SPI debe consultar el registro STATUS del módulo RF para identificar la causa de la interrupción.



### 3.5. DAC y RIT

Se configuró el RIT (Repetitive Interrupt Timer) para que cada vez se alcance la cuenta, en la interrupción se actualice la muestra del DAC, el tiempo entre muestra y muestra es de  $T_s = \frac{1}{3KHz} = 333,3 \mu s$ .

### 3.6. TMR2

Este timer esta configurado en modo free running con resolución de 1 ms, se usa para lograr el efecto de parpadeo de LEDs, para evidenciar al usuario el funcionamiento del dispositivo (por ejemplo, cuando se carga una carga clave correcta al transmisor, el led verde parpadea).

### 3.7. Formato del paquete

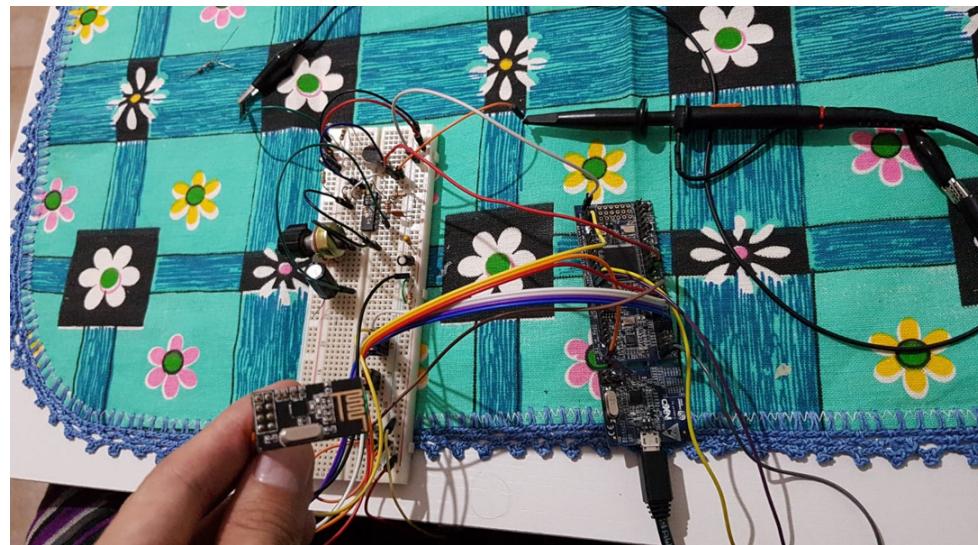
El paquete es un array de 3 Bytes:

```
uint8_t pckt[3]={currentKey,msb,lsb};
```

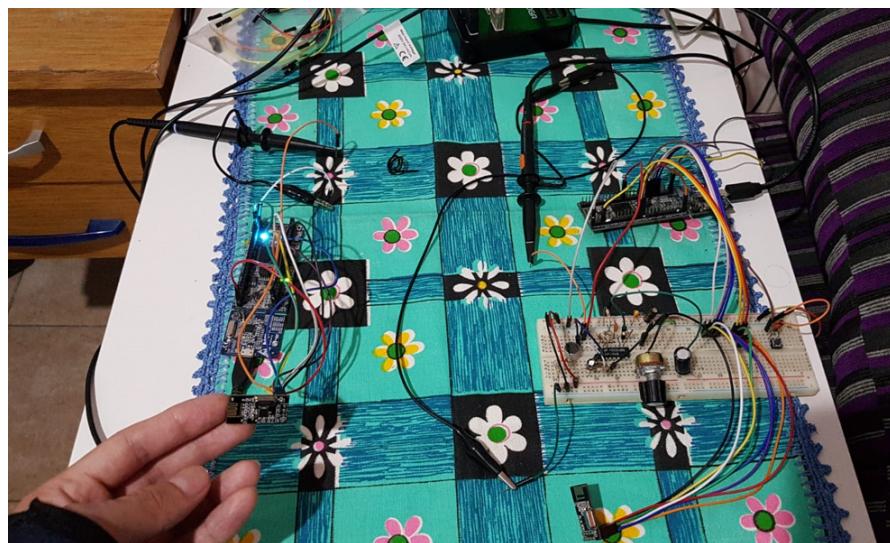
En donde msb y lsb son la parte alta y baja respectivamente del resultado del promedio de 8 muestras (almacenada en una variable tipo uint16\_t).

## 4. Ilustraciones del sistema en funcionamiento

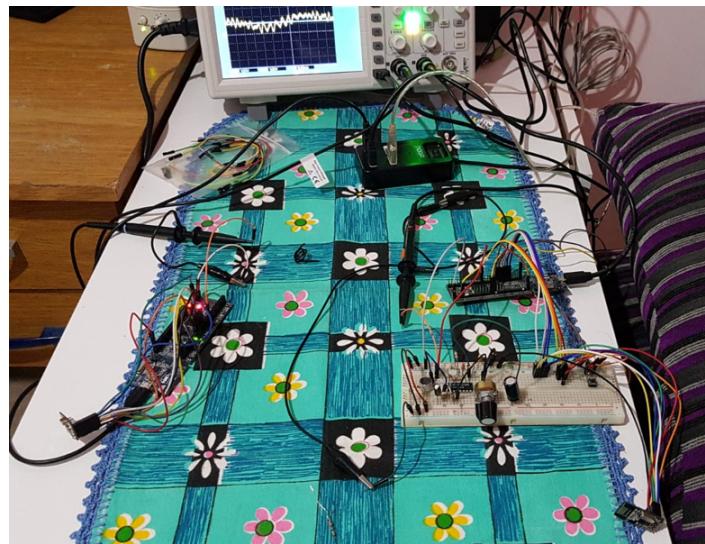
- Circuito de acondicionamiento del micrófono + Transmisor



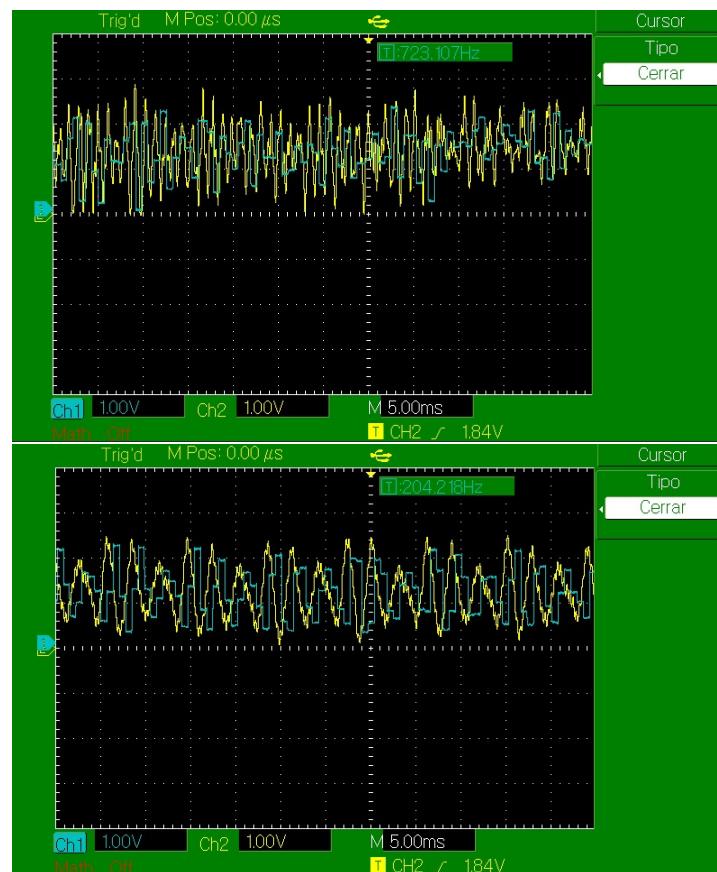
- Transmisión cuando la clave es correcta (Receptor muestra LED en celeste)



- Transmisión cuando la clave es incorrecta (Receptor muestra LED en rojo), notar como la salida del DAC en el osciloscopio (trazo azul) no copia a la entrada del ADC (trazo amarillo).



- Capturas del osciloscopio, cuando la clave es correcta (en azul: salida del DAC, en amarillo: entrada al ADC).



## 5. Código fuente

### 5.1. Transmisor

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>

#include <stdio.h>
#include "pin_conf.h"
#include "ssp_spi.h"
#include "timers.h"
#include "nRF24L01.h"
#include "lpc17xx_exti.h"
#include "LPC17xx.h"
#include "lpc17xx_uart.h"
#include "lpc17xx_gpio.h"
#include "lpc17xx_pinsel.h"
#include "lpc17xx_adc.h"
#include "lpc17xx_timer.h"
#include "lpc17xx_dac.h"
#include "lpc17xx_rit.h"
#include <cr_section_macros.h>

/* Macros UART */
//Macros UART
#define MULVAL 14
#define DIVADDVAL 2
#define Ux_FIFO_EN (1 << 0)
#define Rx_FIFO_RST (1 << 1)
#define Tx_FIFO_RST (1 << 2)
#define DLAB_BIT (1 << 7)
#define Rx_RDA_INT (1 << 0)
#define MASK_INT_ID (0xE)
#define MASK_RDA_INT (2)
#define MASK_CTI_INT (6)
#define UARTx (LPC_UART_TypeDef *)LPC_UART2_BASE

//Modulo Tx
#define COUNT_ERR_TRG 30
#define BUFFER_LENGTH PAYLOAD_WIDTH

/*Variables y punteros */
```



UNC

Universidad  
Nacional  
de Córdoba

FCEFyN

```
uint8_t buffer_tx[BUFFER_LENGTH];
uint8_t receiveData[BUFFER_LENGTH];
uint8_t count_error = 0;
uint8_t currentKey = 0xF1;
uint8_t readyToLoad = 1;
uint8_t cmd[2];
uint8_t *p_cmd = cmd;

//Var de ADC + filtro prom
volatile uint32_t acc0 = 0;
volatile uint8_t samp = 0;
volatile uint8_t n = 8;
volatile uint16_t res0 = 0;
volatile uint16_t dacval = 0;
volatile uint8_t flag_adc;

/* Prototipos */
void initUART2(void);
//proto Gpio
void blinkRedLed(int);
void blinkGreenLed(int);
void blinkBlueLed(int);
void initLEDPins(void);
//proto ADC
void initADC(void);
//proto Timers
void initTMR0(void);
void initTMR2(void);
void confIntGPIO(void);

int main(void)
{
    SystemInit();
    initLEDPins();
    initUART2();
    initADC();
    initTMR0();
    confTimer1();
    confTimer2();
    confPin();
    confSpi();
    nrf24_init(1); //Modo TX-con auto acknowledge
    confIntExt();
    confIntGPIO();
```



UNC

Universidad  
Nacional  
de Córdoba

FCEFyN

```
/* Algorithm */
while (1)
{
    /*
    * Decide que comando ejecutar:
    * '.g': Get current Key
    * '.s': Set a new Key
    * '.r': Record and send message
    * '.o': Turn on radio
    */
    switch (cmd[0])
    {
        //getkey
        case 'g':
            //Get current key
            UART_Send(UARTx, (uint8_t *)&currentKey, sizeof(currentKey), BLOCKING);
            blinkBlueLed(100);
            cmd[0] = 0;
            cmd[1] = 0;
            break;
        case 's':
            //Set a new key
            currentKey = cmd[1];
            UART_SendByte(UARTx, readyToLoad);
            blinkBlueLed(100);
            cmd[0] = 0;
            cmd[1] = 0;
            break;
        case 'o':
            //Turn on radio
            //TMR0 enciende el adc, donde ocurre la transmision
            TIM_ResetCounter(LPC_TIM0);
            TIM_Cmd(LPC_TIM0, ENABLE);
            ADC_ChannelCmd(LPC_ADC, 0, ENABLE);
            cmd[0] = 0;
            cmd[1] = 0;
            break;
    }
}
return 0;
}

void confIntGPIO(void)
{
```

```

//P0.1
LPC_PINCON->PINSEL0 &= ~(0b11 << 2); // PINSEL0 1:0
LPC_GPIO0->FIODIR &= ~(1 << 1); //ENTRADA
LPC_PINCON->PINMODE0 &= ~(0b11 << 2); //PINMODE0 1:0 pullup
//Selecciono la interrupcion por flanco de bajada
LPC_GPIOPIN->IO0IntEnF |= (1 << 1);
LPC_GPIOPIN->IO0IntClr |= (1 << 1); //Limpia la bandera
NVIC_EnableIRQ(EINT3_IRQn); //Habilita las interrupciones por GPIO
}

/*
* BaudRate= ~115200(114882) (si PCLK=25MHz, lo es por defecto)
*/
void initUART2(void)
{
    //p0.10 y p0.11 to UART2, pull-up
    PINSEL_CFG_Type cfgTXRX = {
        .Funcnum = 1,
        .Pinnum = 10,
        .Portnum = 0};
    PINSEL_ConfigPin(&cfgTXRX);

    cfgTXRX.Pinnum = 11;
    PINSEL_ConfigPin(&cfgTXRX);

    //enciendo UART2 (por default encendido)
    LPC_SC->PCONP |= 1 << 24;
    //8 bits, sin paridad, 1 Stop bit y DLAB enable
    LPC_UART2->LCR = 3 | DLAB_BIT;
    LPC_UART2->DLL = 12;
    LPC_UART2->DLM = 0;
    //MULVAL=15(bits - 7:4), DIVADDVAL=2(bits - 3:0)
    LPC_UART2->FDR = (MULVAL << 4) | DIVADDVAL;
    //bloquear divisor latches
    //desactivando DLAB lockeamos los divisores para el Baudrate
    LPC_UART2->LCR &= ~(DLAB_BIT);

    //int cada vez que se reciban cierto numero de char (1 en este caso)
    LPC_UART2->IER |= Rx_RDA_INT;
    LPC_UART2->FCR = 0;
    LPC_UART2->FCR |= Ux_FIFO_EN | Rx_FIFO_RST | Tx_FIFO_RST;
    NVIC_ClearPendingIRQ(UART2_IRQn);
    NVIC_EnableIRQ(UART2_IRQn);
    NVIC_SetPriority(UART2_IRQn, 1);
}

```

```
/*
 * Config ADC
 * * P0.23 to AD0.0
 * * pull-off
 */
void initADC(void)
{
    PINSEL_CFG_Type ADC_pin;
    ADC_pin.Pinnum = 23;
    ADC_pin.Portnum = 0;
    ADC_pin.Funcnum = 1;
    ADC_pin.Pinmode = 2; //TRISTATE

    PINSEL_ConfigPin(&ADC_pin);

    //conf del ADC
    // CCLK/8 = 100Mhz/8 = 12.5 MHZ
    LPC_SC->PCLKSEL0 |= (3 << 24);
    //enciende PCONP, da PDN = 1 y carga un CLKDIV medio choto
    ADC_Init(LPC_ADC, 192307);
    //selecciona bits SEL, o sea el canal habilitado, recordar que es modo hardware
    ADC_ChannelCmd(LPC_ADC, 0, ENABLE);
    //configura START de ADCR
    ADC_StartCmd(LPC_ADC, ADC_START_ON_MAT01);
    //configura EDGE de ADCR
    ADC_EdgeStartConfig(LPC_ADC, ADC_START_ON_FALLING);
    //configura individualmente las int por canales o la global
    ADC_IntConfig(LPC_ADC, ADC_ADGINTEN, SET);
    NVIC_ClearPendingIRQ(ADC_IRQn);
    NVIC_SetPriority(ADC_IRQn, 3);
    NVIC_EnableIRQ(ADC_IRQn);

    //AD0.1-AD0.7 a GPIO, pull-down, salida digital, estado bajo
    //tener en cuenta: AD0.3 DACOUT, AD0.6 y AD0.7 UART0
    unsigned char nullCh[6] = {2, 3, 24, 25, 30, 31};

    PINSEL_CFG_Type cfgNullCh = {
        .Funcnum = 0,
        .OpenDrain = PINSEL_PINMODE_NORMAL,
        .Pinmode = PINSEL_PINMODE_PULLDOWN,
    };

    for (unsigned char ch = 0; ch < sizeof(nullCh); ch++)
    {
```

```
//ch del puerto 0
if (ch < 4)
{
    //pull-down, pin como salida
    cfgNullCh.Portnum = 0;
    cfgNullCh.Pinnum = *(nullCh + ch);
    PINSEL_ConfigPin(&cfgNullCh);
    GPIO_SetDir(0, 1 << (*(nullCh + ch)), 1);
    //estado bajo
    GPIO_ClearValue(0, 1 << (*(nullCh + ch)));
}

//ch del puerto 1
else
{
    //pull-down, pin como salida
    cfgNullCh.Portnum = 1;
    cfgNullCh.Pinnum = *(nullCh + ch);
    PINSEL_ConfigPin(&cfgNullCh);
    GPIO_SetDir(1, 1 << (*(nullCh + ch)), 1);
    //estado bajo
    GPIO_ClearValue(1, 1 << (*(nullCh + ch)));
}
}

/*
*   Config TMR0
*   * Recordar F_MRx = 2*prom*F_s(deseada)
*   * F_s = 3 KHz, prom = 8 -> F_MRx = 48 KHz -> 20.8us
*   * Con 12us -> F_s(real)= 47.619 KHz
*/
void initTMR0(void)
{
    //enciende TMR0 (recordar que ya lo esta por defecto);
    LPC_SC->PCONP |= (1 << 1);
    //toglear (funcion 3) EM0 (MAT0.1)
    LPC_TIM0->EMR |= (3 << 6);
    //si PCLK = 25MHz entonces con ese PR se logra resolucion de 1us
    LPC_TIM0->PR = 25 - 1;
    LPC_TIM0->MR1 = 21 - 1;
    //solo resetar cuando se llegue match, no int, no stop
    LPC_TIM0->MCR |= (2 << 3);
}
```



UNC

Universidad  
Nacional  
de Córdoba

FCEFyN

```
void UART2_IRQHandler(void)
{
    uint32_t uartIntSt = UART_GetIntId(LPC_UART2); //valor del registro IIR

    if (((uartIntSt & MASK_INT_ID) >> 1) == MASK_RDA_INT)
    {
        UART_Receive(LPC_UART2, cmd, sizeof(cmd), BLOCKING);
    }
    else if (((uartIntSt & MASK_INT_ID) >> 1) == MASK_CTI_INT)
    {
        uint8_t dummy = LPC_UART2->RBR;
    }
}

void ADC_IRQHandler()
{
    //incrementar contador de muestras
    samp++;
    //leer valor del adc
    res0 = ADC_ChannelGetData(LPC_ADC, 0);
    //acumular valor de conversion y bajar banderas
    acc0 += res0;
    //promediar cada n muestras
    if (samp == n)
    {
        acc0 >>= (n / 2) - 1; // acc0/n
        //armar paquete
        uint8_t msb = (uint8_t)((acc0 >> 8) & 0xFF);
        uint8_t lsb = (uint8_t)(acc0 & 0xFF);
        uint8_t pckt[] = {currentKey, msb, lsb};
        //enviar a nRF
        nrf24_transmit_payload(pckt, sizeof(pckt));
        //reiniciar variables
        acc0 = 0;
        samp = 0;
    }
    return;
}

void EINT0_IRQHandler(void)
{
    uint8_t aux0[1];
    EXTI_ClearEXTIFlag(0);
    uint8_t status = nrf24_status();
    if (status & RF24_MASK_MAX_RT)
```

```
{  
    //error de retransmision  
    aux0[0] = status | RF24_MASK_MAX_RT;  
    //limpio el flag MAX_RT (maximo de retransmisiones)  
    nrf24_writeToNrf(W, RF24_STATUS, aux0, sizeof(aux0));  
    count_error++;  
    if (count_error > COUNT_ERR_TRG)  
    {  
        FlagErrorTransmission = 1;  
        count_error = 0;  
    }  
}  
else if (status & (RF24_MASK_TX_DS | RF24_MASK_RX_DR))  
{  
    //transmision exitosa  
    aux0[0] = status | RF24_MASK_TX_DS | RF24_MASK_RX_DR;  
    //limpio el flag TX_DS y RX_DR  
    nrf24_writeToNrf(W, RF24_STATUS, aux0, sizeof(aux0));  
    blinkGreenLed(10);  
}  
}  
  
void EINT3_IRQHandler(void)  
{  
    LPC_GPIOINT->IO0IntClr |= (1 << 1);  
    TIM_Cmd(LPC_TIM0, DISABLE);  
    ADC_ChannelCmd(LPC_ADC, 0, DISABLE);  
}  
  
void blinkRedLed(int time)  
{  
    LED_RED_TOGGLE();  
    delay_ms(time);  
    LED_RED_TOGGLE();  
}  
  
void blinkGreenLed(int time)  
{  
    LED_GREEN_TOGGLE();  
    delay_ms(time);  
    LED_GREEN_TOGGLE();  
}  
  
void blinkBlueLed(int time)  
{
```

```
LED_BLUE_TOGGLE();
delay_ms(time);
LED_BLUE_TOGGLE();
}

5.2. Receptor

#ifndef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>

#include <stdio.h>
#include "pin_conf.h"
#include "ssp_spi.h"
#include "timers.h"
#include "nRF24L01.h"
#include "adc_dac.h"
#include "lpc17xx_exti.h"
#include "lpc17xx_dac.h"

#define BUFFER_LENGTH PAYLOAD_WIDTH
#define KEY 0xAA

uint8_t val[5];
uint8_t buffer_tx[BUFFER_LENGTH];
uint8_t receiveData[BUFFER_LENGTH];
uint8_t count_error;
uint8_t status;
uint8_t flag_recv_data;
uint16_t dac_value;

void checkData();

int main(void)
{
    // for(int i =0;i<BUFFER_LENGTH;i++){
    // buffer_tx[i] = i;
    // }
    /* System Clock Init */
    SystemInit();
    confPin();
    initLED Pins();
    confSpi();
    confIntExt();
```

```

confTimer1();
confTimer2();
//initRit();
initDAC();
nrf24_init(0); //Modo Rx

//habilito modulo en RX
nrf24_listen_payload();
LPC_GPIO3->FIOCLR |= (1 << 26); //luz azul , escuchando
while (1)
{
    if (flag_recv_data)
    { //llego un paquete
        flag_recv_data = 0;
        if (receiveData[0] == KEY)
        { //tiene el inicio de trama valido
            dac_value = (((uint16_t)receiveData[1]) << 8) | (receiveData[2]);
            DAC_UpdateValue(LPC_DAC, (dac_value >> 2) & 0x3FF);
        }
        else
        {
            LPC_GPIO3->FIOSET |= (1 << 25);
            LPC_GPIO3->FIOSET |= (1 << 26);
            LPC_GPIO0->FIOCLR |= (1 << 22); //luz violeta
        }
    }
    return 0;
}

void EINT0_IRQHandler(void)
{
    uint8_t aux0[1];
    EXTI_ClearEXTIFlag(0);
    nrf24_CE_low();
    uint8_t status = nrf24_status();
    if (status & RF24_MASK_RX_DR)
    {
        //si la interrupcion de un paquete es valida, extraigo todo lo de la fifoRX
        flag_recv_data = 1;
        //valor para limpiar el flag de interrupcion del nrf
        aux0[0] = status | RF24_MASK_TX_DS | RF24_MASK_RX_DR;
        //leo la FIFO
        nrf24_writeToNrf(R, RF24_R_RX_PAYLOAD, receiveData, sizeof(receiveData));
        //limpio los flags
    }
}

```

```
nrf24_writeToNrf(W, RF24_STATUS, aux0, sizeof(aux0));
}
else
{
    LPC_GPIO3->FIOSET |= (1 << 25);
    LPC_GPIO0->FIOCLR |= (1 << 22);
}
nrf24_CE_high();
}

void checkData()
{
    count_error = 0;

    for (int i = 0; i < BUFFER_LENGTH; i++)
    {
        if (buffer_tx[i] != receiveData[i])
        {
            count_error++;
        }
        else
        {
            //
        }
    }
    if (count_error)
    {
        LED_RED_TOGGLE();
    }
    else
    {
        LED_GREEN_TOGGLE();
    }
}
```

### 5.3. Script en Python

```
#modulos
import serial
#from serial.serialutil import EIGHTBITS, PARITY_NONE, STOPBITS_ONE, Timeout
#abrir puerto
ser = serial.Serial(port='COM3',
                     baudrate=115200,
                     bytesize=serial.EIGHTBITS,
                     parity=serial.PARITY_NONE,
```

```
stopbits=serial.STOPBITS_ONE)

#ser=serial.serial_for_url('loop://',timeout=1)

#INTERFACE
sep= "//" + "-"*75 + "//\n"
print(sep)
print('\t\tPROYECTO FINAL ED3\n')
print('Escriba un comando para seguir, se pueden consultar los que'
estan\ndisponibles escribiendo "cmd".\n')
print(sep)

#bandera de finalizacion
exit_flag=False

#mientras se requiera un comando
while exit_flag == 0:
    #limpiar buffers
    ser.flushInput
    ser.flushOutput

    #ingresar comando
    cmd = input("Ingrese un comando >").strip().lower()

    #si cmd, imprimir comandos
    if cmd=='cmd':
        print('\nComandos disponibles:\n')
        print('\t* "exit"    -> cerrar el programa\n')
        print('\t* "onradio" -> iniciar transmision\n')
        print('\t* "getkey"   -> obtener clave de validacion actual del receptor\n')
        print('\t* "setkey"   -> modificar clave de validacion del
receptor\n')

    #si exit, finalizar
    elif cmd=='exit':
        exit_flag=True

    #si onradio, activar transmision
    elif cmd=='onradio':
        #enviar comando al micro, es necesario pasarlo a bytes con encode
        ser.write("o\n".encode())

    #si offradio, finalizar transmision
    elif cmd=='offradio':
```



UNC

Universidad  
Nacional  
de Córdoba

FCEFyN

```
ser.write("f\n".encode())

#si getkey, pedir al micro la llave actual
elif cmd=='getkey':
    #pedir
    ser.write("g\n".encode())
    #escuchar 1 byte
    currentKey=ser.read(1)
    print(f"La clave actual es: {currentKey.hex()}\n")

#si setkey, reconfigurar la llave del micro
elif cmd=='setkey':
    newKey=input('Ingrese la nueva clave de validación (1 Byte en HEX) >> ')

    #si fuera de rango, error
    if int(newKey,16) > (2**8)-1:
        print('Error, la clave no es valida\n')
    #sino, cambiar llave
    else:
        ser.write('s'.encode())
        ser.write(bytes.fromhex(newKey))
        #esperar byte de confirmacion
        readyToLoad=ser.read(1)

        if readyToLoad:
            #pasar a bytes y mandar la nueva llave
            print("Clave cargada con exito.\n")
        else:
            print('Ocurrio un error cargando la nueva clave\n')

    #si no existe comando
    else:
        print('\t*COMANDO INVALIDO*\n')

#cerrar puerto
ser.close()
```