
	<b>UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO</b> <b>FACULTAD DE INGENIERÍA</b> <b>Microprocesadores y Microcontroladores</b>	Semestre: 2020-2	
	<b>Prof. Dr. Saúl De La Rosa Nieves</b>	Página 1 de 21	

TAREA-EXAMEN 1	
<b>Título:</b>	<b>Desarrollo del control inalámbrico de mensajes desplegables</b>
<b>Fecha:</b>	18-06-2021
<b>Preparado por:</b>	Fiel Muñoz Teresa Elpidia
<b>Evaluación:</b>	

## I. Planteamiento del proyecto



Diseñar un control por medio de una aplicación en un teléfono inteligente para configurar vía BT la hora de un reloj en tiempo real controlado por el microcontrolador TM4C1294 y la hora y fecha se mostrará en un “Display de 7 segmentos” de 4 dígitos, a la vez se agregó que reproduzca una señal senoidal con frecuencia variable utilizando protocolo SPI y que se pueda ajustar la frecuencia de esta señal por un comando BT.

## II. Requerimientos del proyecto

### *Requerimientos de hardware*

1. Como unidad de control del “Display” se utilizará el microcontrolador TM4C1294NCPDT.
2. La selección de los mensajes desplegables se realizará desde un teléfono inteligente mediante una aplicación diseñada con APP Inventor.
3. La recepción de los comandos del teléfono inteligente en el microcontrolador se realizará mediante un puerto UART conectado con un módulo Bluetooth HC-05.
4. Se utilizará el Reloj en Tiempo Real DS1307

## III. Determinación de Requerimientos del proyecto

	<b>UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO</b> <b>FACULTAD DE INGENIERÍA</b> <b>Microprocesadores y Microcontroladores</b>	Semestre: 2020-2	
	<b>Prof. Dr. Saúl De La Rosa Nieves</b>	Página 2 de 21	

Este proyecto requiere el dominio y la aplicación de los conceptos de UART, SPI e I2C vistos en la materia de microcontroladores y repasadas en esta clase, para lo siguiente requerimos que el microcontrolador sea capaz de realizar estos protocolos de comunicación y que a su vez los realice a la vez.

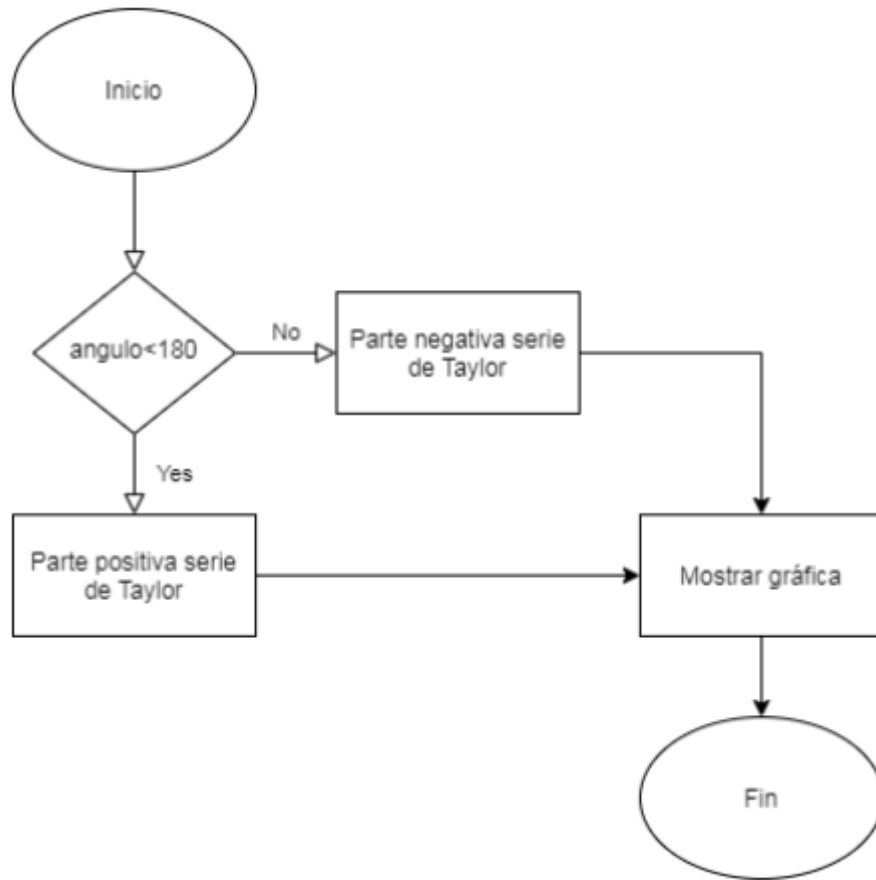
Para los requerimientos se necesita también la creación de una aplicación bluetooth con comandos para poder ingresar la hora (minutos, y horas) y que tenga por lo menos una forma de poder seleccionar cómo ir ingresando estos datos y cómo decirle al programa que los almacene.

Con la parte de SPI se requieren también dos comandos extras para las frecuencias a manejar en el potenciómetro digital.

## IV. Diseño conceptual

El diseño conceptual abordará los métodos a seguir para resolver estos problemas a nivel de pseudocódigo, esto se realizará con la creación de diagramas de flujo para cada etapa que se considere relevante.

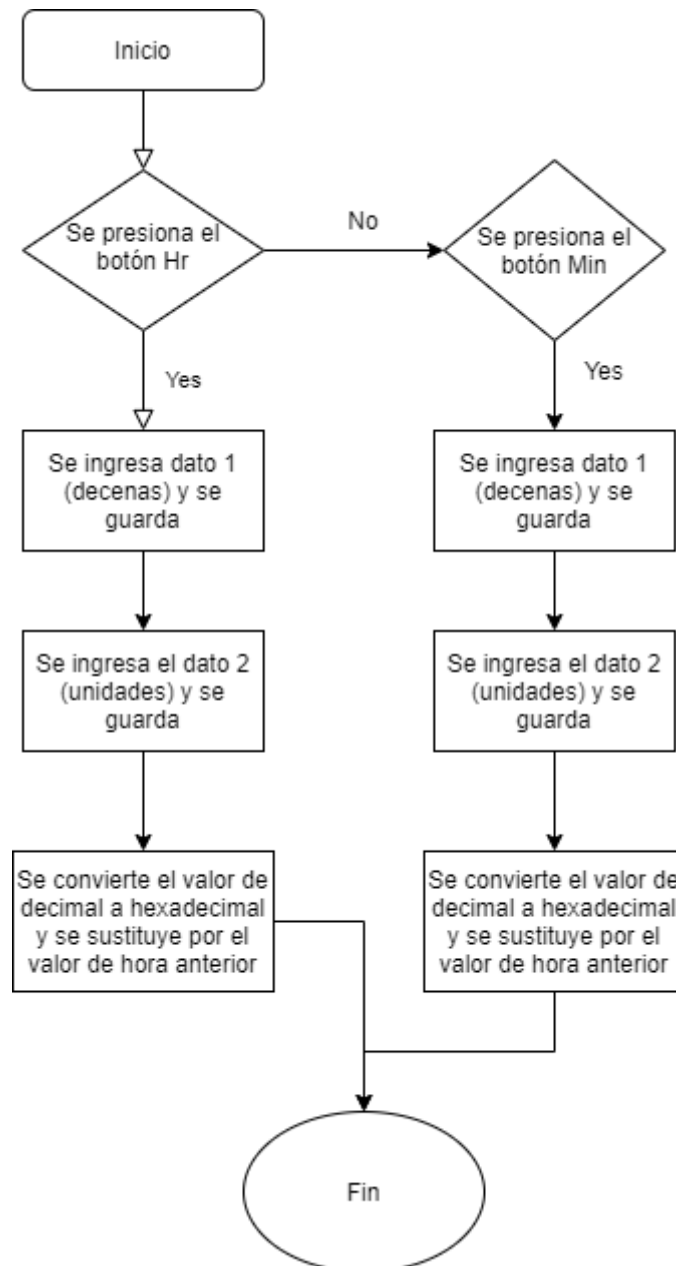
### *Modelado de señal senoidal*



*Muestra de datos en display 7 segmentos*



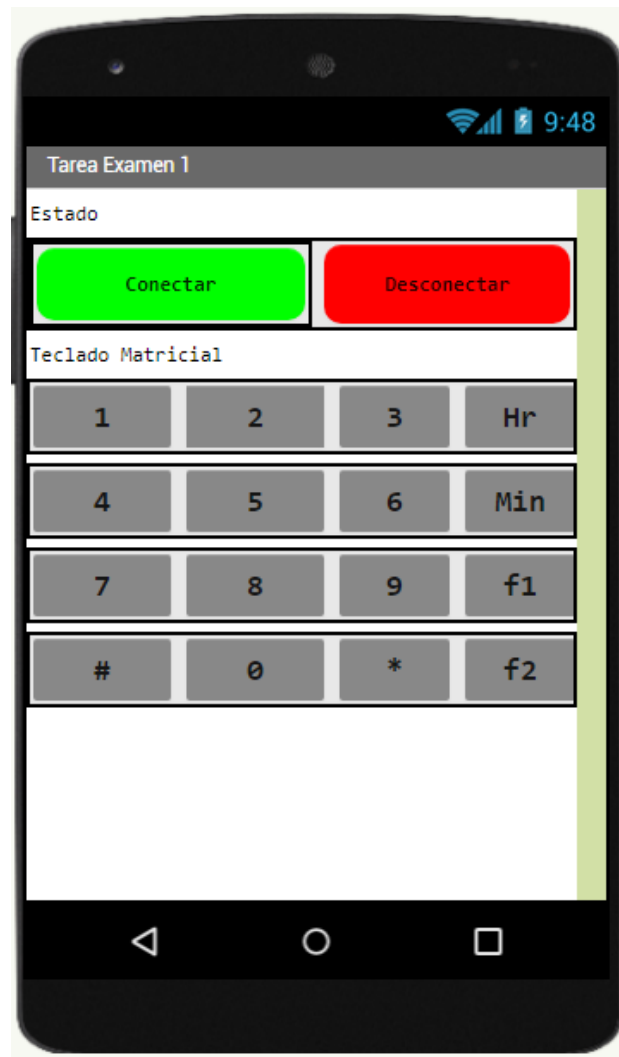
*Ajuste de hora o minutos*



## V. Diseño de detalle

Para el diseño de detalle se mostrará el código implementado para resolver el proyecto, aquí agregaré la parte de la aplicación bluetooth y que significa cada botón, la muestra de todo el programa está condicionada a recibir o no los comandos y cada uno hace algo distinto.

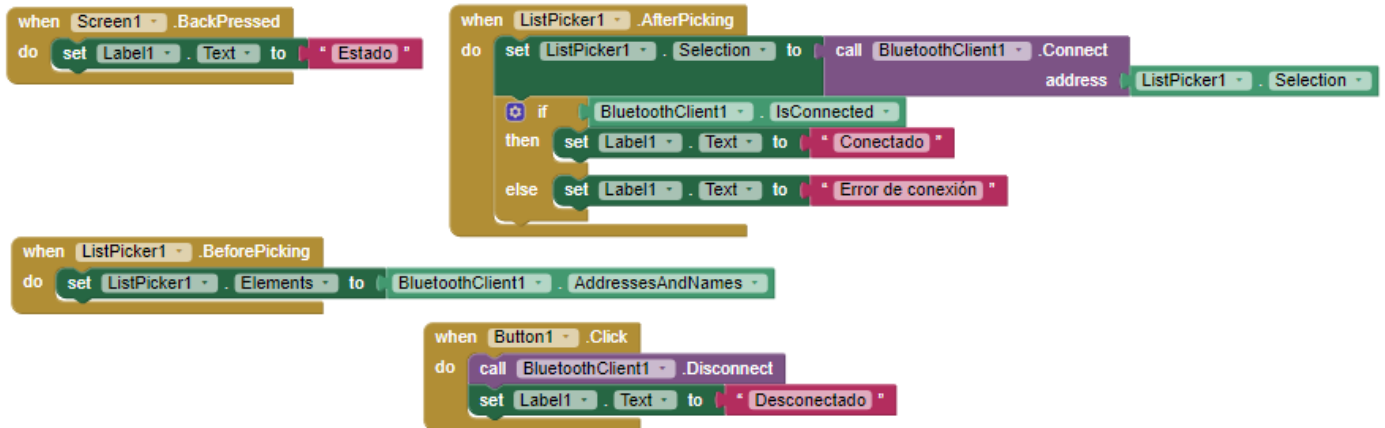
### *Layout de la aplicación*



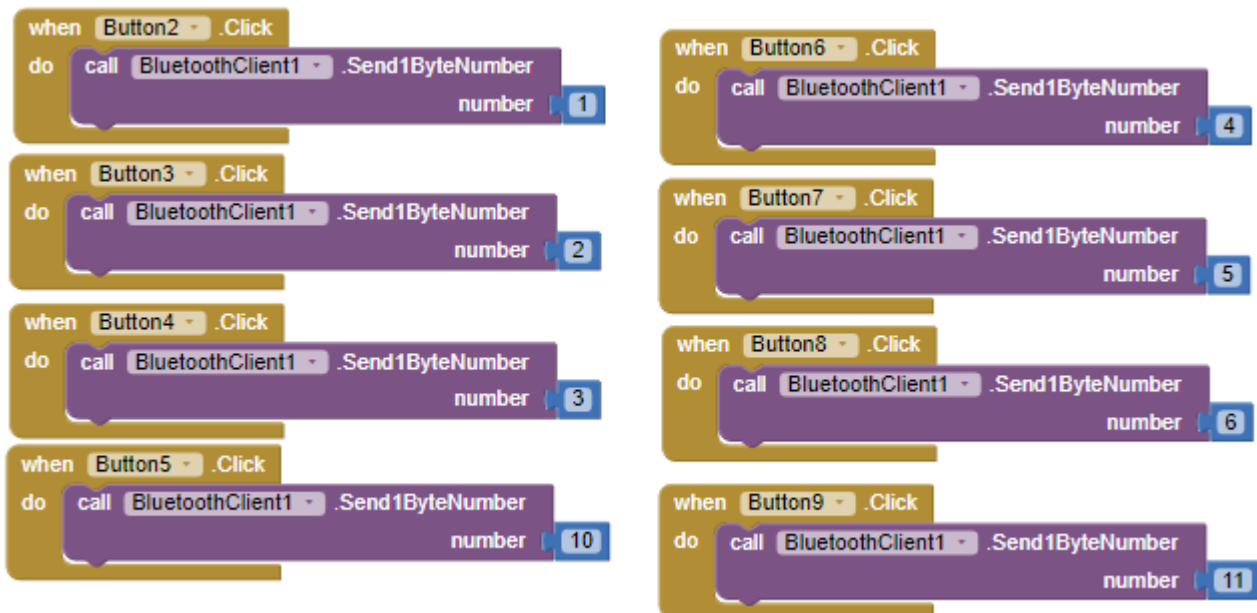
Los valores numéricos son para poder ajustar la hora, el botón de hora es para que se puedan comenzar a ingresar los valores numéricos para la hora y lo mismo con los minutos, f1 y f2 son frecuencias establecidas, f1 de 0.5 Hz y f2 de 0.1 Hz, los demás botones del teclado matricial no tienen un uso.

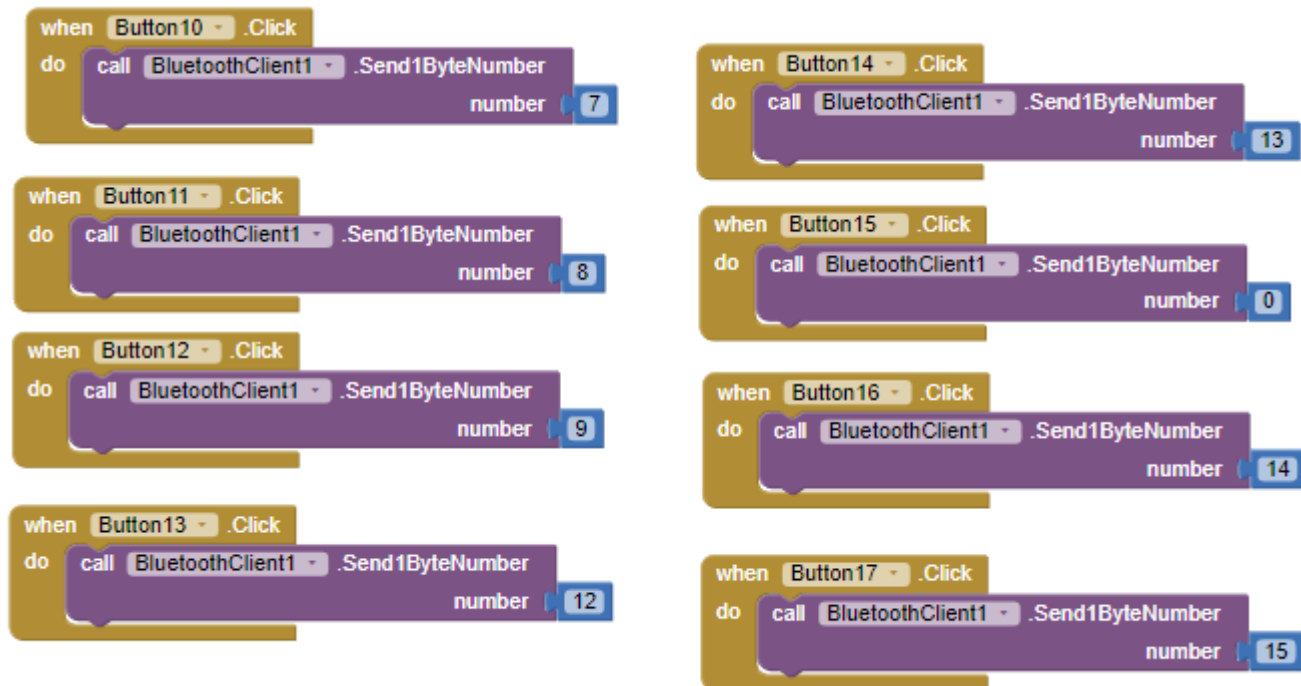
### ***Código AppInventor***

Conexiones bluetooth



### Asignación de botones





## Código C comentado

```
/*
 * rtc.c
 *
 * Created on: 17 jun. 2021
 * Author: Teresa Fiel
 */

/**
 * PROGRAMA QUE CONFIGURA EL PERIFERICO I2C
 * EN EL TIVA TM4C1294 COMO MAESTRO Y ESCLAVO EL RTC DS1307
 */

//Librerias
#include<stdint.h>
#include<stdbool.h>
#include "inc/tm4c1294ncpdt.h"

//-----DISPLAY 7 SEGMENTOS-----

#define GPIO_PORTM_DATA_R (*(volatile unsigned long *)0x400633FC) // Registro de Datos Puerto M
#define GPIO_PORTM_DIR_R (*(volatile unsigned long *)0x40063400) // Registro de Dirección Puerto M
#define GPIO_PORTM_DEN_R (*(volatile unsigned long *)0x4006351C) // Registro de Habilitación Puerto M
#define GPIO_PORTM_PDR_R (*(volatile unsigned long *)0x40063514) // Registro de Pull-Down Puerto M

#define GPIO_PORTD_DATA_R (*(volatile unsigned long *)0x4005B3FC) // Registro de Datos Puerto D
#define GPIO_PORTD_DIR_R (*(volatile unsigned long *)0x4005B400) // Registro de Dirección Puerto D
#define GPIO_PORTD_DEN_R (*(volatile unsigned long *)0x4005B51C) // Registro de Habilitación Puerto D
#define GPIO_PORTD_PUR_R (*(volatile unsigned long *)0x4005B510) // Registro de Pull-Up Puerto D

#define GPIO_PORTC_AFSEL_R (*(volatile uint32_t *)0x4005A420)
#define GPIO_PORTC_DEN_R (*(volatile uint32_t *)0x4005A51C)
#define GPIO_PORTC_AMSEL_R (*(volatile uint32_t *)0x4005A528)
#define GPIO_PORTC_PCTL_R (*(volatile uint32_t *)0x4005A52C)

#define UART7_DR_R (*(volatile uint32_t *)0x40013000)
#define UART7_FR_R (*(volatile uint32_t *)0x40013018)

#define UART_FR_TXFF 0x00000020 // FIFO TX UART llena
#define UART_FR_RXFE 0x00000010 // FIFO RX UART vacía
```





```
#define UART7_IBRD_R      (*((volatile uint32_t *)0x40013024)) //Divisor de entero de Baud Rate UART (p.1184)
#define UART7_FBRD_R      (*((volatile uint32_t *)0x40013028)) //Divisor de fracción de Baud Rate UART (p.1185)
#define UART7_LCRH_R      (*((volatile uint32_t *)0x4001302C)) // Control de línea UART (p.1186)

#define UART_LCRH_WLEN_8   0x00000060 // palabra de 8 bits
#define UART_LCRH_FEN      0x00000010 // Habilitación de la FIFO de la UART

#define UART7_CTL_R        (*((volatile uint32_t *)0x40013030)) //Control UART (p.1188)
#define UART7_CC_R         (*((volatile uint32_t *)0x40013FC8)) // Configuración de control (p.1213)
#define UART_CC_CS_M       0x0000000F // UART fuente del reloj de Baud Rate
#define UART_CC_CS_SYSCLK   0x00000000 // Sistema de reloj (basado en fuente de reloj y factor de división)
// source and divisor factor)
#define UART_CC_CS_PIOSC    0x00000005 // PIOSC
#define SYSCCTL_ALTKCFG_R   (*((volatile uint32_t *)0x400FE138)) //Configuración de reloj alterno (p.280)
#define SYSCCTL_ALTKCFG_ALTCLK_M \
    0x0000000F // Fuente alternativa de reloj \
#define SYSCCTL_ALTKCFG_ALTCLK_PIOSC \
    0x00000000 // PIOSC

//-----Inicialización del UART-----
// Inicializa UART7 para 115,200 bauds (reloj de 16 MHz del PIOSC),
// palabra de 8 bit, no hay bits de paridad, un stop bit, los FIFOs habilitados,
// no hay interrupciones

//--VARIABLES--

void SSI0_init (void);
void SSI0_sendData (uint16_t dat);
void pot_setVal (uint8_t slider);
void SysTick_Init(void);
void SysTick_Wait(uint32_t retardo);

uint32_t adc_result;
uint8_t y = 0x00; // Valor inicial.
float x=0; //angulo
int ascendente = 1;
float val;
int val_int;
int signaltype=0;
uint32_t data;
int datin=0;
float frecuencia=0.3;

int recorrido=0;
int decenasd[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int unidadesd[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int segmentosd[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int segmentosu[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int dig1=0x00;
int dig2=0x00;
int digdec=0x00;
int dec1=0x00;
int dec2=0x00;
int dec3=0x00;

//REGISTROS DE RELOJ
#define SYSCCTL_RCGCGPIO_R      (*((volatile uint32_t *)0x400FE608)) //Reloj del puerto
#define SYSCCTL_RCGCI2C_R      (*((volatile uint32_t *)0x400FE620)) //Reloj de I2C
#define SYSCCTL_PRGPIO_R       (*((volatile uint32_t *)0x400FEA08)) //Bandera de "Peripheral Ready"

//REGISTROS DEL PUERTO B
#define GPIO_PORTB_DATA_R      (*((volatile uint32_t *)0x400593FC)) //Para los datos del puerto
#define GPIO_PORTB_DIR_R       (*((volatile uint32_t *)0x40059400)) //Para seleccionar función
#define GPIO_PORTB_AFSEL_R     (*((volatile uint32_t *)0x40059420)) //Para seleccionar función alterna
#define GPIO_PORTB_ODR_R       (*((volatile uint32_t *)0x4005950C)) //Para activar el Open Drain
#define GPIO_PORTB_DEN_R       (*((volatile uint32_t *)0x4005951C)) //Para activar función digital
#define GPIO_PORTB_PCTL_R      (*((volatile uint32_t *)0x4005952C)) //Para el control del puerto

//REGISTROS DEL MÓDULO I2C
#define I2C0_MSA_R              (*((volatile uint32_t *)0x40020000)) //I2C Master Slave Address
#define I2C0_MCS_R              (*((volatile uint32_t *)0x40020004)) //I2C Master Control Status
```



```
#define I2C0_MDR_R      (*((volatile uint32_t *)0x40020008)) //I2C Master Data Register
#define I2C0_MTPR_R     (*((volatile uint32_t *)0x4002000C)) //I2C Master Time Period
#define I2C0_MCR_R      (*((volatile uint32_t *)0x40020020)) //I2C Master Congirutation Register

#define SYSCTL_RCGCGPIO_R  (*((volatile uint32_t *)0x400FE608)) // Control de reloj GPIO (p.382)
#define SYSCTL_RCGCUART_R  (*((volatile uint32_t *)0x400FE618)) // Control de reloj UART (p.388)
#define SYSCTL_PRGPIO_R    (*((volatile uint32_t *)0x400FEA08)) // Estado de GPIO (p.499)
#define SYSCTL_PRGPIO_R0   0x00000004 // Puerto GPIO C listo
#define SYSCTL_PRUART_R    (*((volatile uint32_t *)0x400FEA18)) // Estado del UART (p.505)
#define SYSCTL_PRUART_R0   0x00000080 // UART Module 7 del UART listo

/*
El registro I2C Master Control/Status (I2C_MCS_R) tiene:
-Modo READ-ONLY DATUS: los 7 bits menos significativos son:
    7:Clock Time Out Error    6:BUS BUSY        5:IDLE
    4:Arbitration Lost        3:DataAck         2:AdrAck
    1>Error                   0:CONTROLLER BUSY

-Modo WRITE-ONLY CONTROL Los 6 bits menos significativos son:
    6:BURST    5:QuickCommand  4:High Speed Enable
    3:ACK      2:STOP          1:START
    0:RUN

*/
#define I2C_MCS_ACK 0x00000008 //Transmitter Acknowledge Enable
#define I2C_MCS_DATAACK 0x00000008 // Data Acknowledge Enable
#define I2C_MCS_ADRACK 0x00000004 // Acknowledge Address
#define I2C_MCS_STOP 0x00000004 // Generate STOP
#define I2C_MCS_START 0x00000002 // Generate START
#define I2C_MCS_ERROR 0x00000002 // Error
#define I2C_MCS_RUN 0x00000001 // I2C Master Enable
#define MAXRETRIES 5 // number of receive attempts before giving up

/**Direcciones del DS1307
int AdreDS1307 =0x068;///Dirección del RTC DS1307
int AdreSec= 0x00;
int AdreMin=0x01;

/*El cálculo del Time Period Register (TPR) se especifica en la página 1284
Asumiendo un reloj de 16 MHz y un modo de operación estándar (100 kbps):
*/
int TPR = 7;

// Variables para manejar los valores del RTC
uint8_t segundos, minutos, horas, dia, fecha, mes, anio;
uint8_t error;
uint32_t i;

/** Función que inicializa los relojes, el GPIO y el I2C0 **
void I2C_Init(void){
//CONFIGURACIÓN DE LOS RELOJ
SYSCTL_RCGCI2C_R |= 0x0001; // Activamos el reloj de I2C0 [I2C9 I2C8 I2C7 ... I2C0]<--Mapa de RCGCI2C
SYSCTL_RCGCGPIO_R |= 0x7EFF; // Activamos el reloj GPIO_PORTB mientras se activa el reloj de I2C0
while((SYSCTL_PRGPIO_R&0x0002) == 0){};//Espero a que se active el reloj del puerto B

//CONFIGURACIÓN DE LOS GPIOs
/*Acorde con la tabla "Signals by function" de la p. 1808:
el PIN 2 del puerto B (PB2) es el I2C0SCL del I2C0, y
el PIN 3 del puerto B (PB3) es el I2C0SDA del I2C0
*/
GPIO_PORTB_AFSEL_R |= 0x0C; // Activo la función alterna del PB2 y PB3
GPIO_PORTB_ODR_R |= 0x08; // Activo el OPEN DRAIN para el PB3, ya que el PB2 ya tiene uno por preconfig.
GPIO_PORTB_DIR_R |= 0x0C; //Activo al PB2 y al PB3 como OUTPUT
GPIO_PORTB_DEN_R |= 0x0C; //Activo la función digital de PB3 y PB2

GPIO_PORTD_DIR_R |= 0x0F; // Salidas de PD0-PD3
GPIO_PORTD_DEN_R |= 0x0F; // Habilita PD0-PD3
GPIO_PORTD_PUR_R |= 0x0F; // Habilita resistencias de pull-up para dígitos de habilitación 7 segmentos
```



```
/*
Así como el registro AFSEL indica que se ejecutará una función externa, en el registro PCTL
debemos indicar qué función alterna se realizará acorde con la tabla 26-5 de la p.1808 e indicarlo
en el correspondiente PCMN (uno por cada bit del puerto) del registro PCTL
*/
GPIO_PORTB_PCTL_R|=0x00002200;

//CONFIGURACIÓN DEL MODULO I2C0
I2C0_MCR_R = 0x00000010; // Habilitar función MASTER para el I2C0
I2C0_MTPR_R = TPR; // Se establece una velocidad estándar de 100kbps
}

// ** Función esperar **
int esperar(){
    while(I2C0_MCS_R&0x00000001){}; //Espero a que la transmisión acabe
    if(I2C0_MCS_R&0x00000002==1){ //¿Hubo error?
        error=1;
        return error;
    };
    return 0;
}

/** Función para configurar al esclavo RTC DS1307 **

void CargarFecha(){
    /*
    Programar: Jueves 31 de octubre del 2019, a las 9:40:00 pm

    El mapa de memoria del DS1207 es el siguiente:
    DIRECCIÓN  FUNCIÓN  BIT7  BIT6  BIT5  BIT4  BIT3  BIT2  BIT1  BIT0
    00h  Segundos  0    0    0    0    0    0    0    0
    01h  Minutos  0    0    0    0    0    0    0    0
    02h  Horas    0    0    0    1    0    0    0    0
    03h  Día      0    0    1    1    0    0    0    1
    04h  Fecha    0    0    0    0    0    1    0    0
    05h  Mes      0    0    0    1    0    0    0    0
    06h  Año      0    0    0    1    1    0    0    1
    07h  Control  0    0    0    0    0    0    1    1
    08h-3Fh  RAM 56x8
    */
    //Por lo tanto

    int segundos=0x00, minutos=0x40, horas=0x09, dia=0x05, fecha=0x11, mes=0x06, anio=0x21;
    while(I2C0_MCS_R&0x00000001){}; // wait for I2C ready
    //Para transmitir
    I2C0_MSA_R=(AdreDS1307<<1)&0xFE; //Cargo la dirección del DS1307 e indico "SEND", es decir, el Slave va a recibir
    I2C0_MDR_R=AdreSec&0xFF; //Envío la Subdirección( dirección del registro interno "segundos") al DS1307
    I2C0_MCS_R=(I2C0_MCS_RUN|I2C0_MCS_START); // Condición de START y corro
    esperar();
    for(i=0;i<300;i++){ //Delay

    I2C0_MDR_R=segundos; //Envío el valor de "segundos"
    I2C0_MCS_R=(I2C0_MCS_RUN);
    esperar();
    for(i=0;i<300;i++){ //Delay

    I2C0_MDR_R=minutos; //Envío el valor de "minutos"
    I2C0_MCS_R=(I2C0_MCS_RUN); //Inicio la transmisión 1
    esperar();
    for(i=0;i<300;i++){ //Delay

    I2C0_MDR_R=horas; //Envío el valor de "horas"
    I2C0_MCS_R=(I2C0_MCS_RUN); //Inicio la transmisión 2
    esperar();
    for(i=0;i<300;i++){ //Delay

    I2C0_MDR_R=dia; //Envío el valor de "dia"
    I2C0_MCS_R=(I2C0_MCS_RUN); //Inicio la transmisión 4
```



```
esperar();
for(i=0;i<300;i++){ //Delay

I2C0_MDR_R=fecha; //Envio el valor de "fecha"
I2C0_MCS_R=(I2C_MCS_RUN); //Inicio la transmisión 5
esperar();
for(i=0;i<300;i++){ //Delay

I2C0_MDR_R=mes; //Envio el valor de "mes"
I2C0_MCS_R=(I2C_MCS_RUN); //Inicio la transmisión 6
esperar();
for(i=0;i<300;i++){ //Delay

I2C0_MDR_R= anio; //Envio el valor de "año"
I2C0_MCS_R=(I2C_MCS_STOP|I2C_MCS_RUN); //Inicio la ultima transmisión y STOP
esperar();
for(i=0;i<300;i++){ //Delay
}

void leerFecha(){
    while(I2C0_MCS_R&0x00000001){}; // wait for I2C ready

    //Para actualizar registro para iniciar la lectura
    I2C0_MSA_R=(AdreDS1307<<1)&0xFE; //Cargo la dirección del DS1307 e indico "SEND", es decir, el Slave va a
    recibir
    I2C0_MDR_R=AdreSec&0xFF; //Envio la Subdirección( dirección del registro interno "segundos") al DS1307
    I2C0_MCS_R=(I2C_MCS_START|I2C_MCS_RUN); // Condición de START, y corro
    esperar();
    for(i=0;i<300;i++){ //Delay

    //Para recibir información
    I2C0_MSA_R=(AdreDS1307<<1)&0xFE; //La dirección del DS1307 en el Master Slave Address
    I2C0_MSA_R|=0x01; //Indico "RECEIVE", es decir, el Slave va a transmitir
    I2C0_MCS_R=(I2C_MCS_START|I2C_MCS_RUN|I2C_MCS_ACK); // Condición de RESTART, corro, ack
    esperar();
    for(i=0;i<300;i++){ //Delay
    segundos=(I2C0_MDR_R&0xFF); //El Master lee lo que envía el DS1307

    I2C0_MCS_R=(I2C_MCS_RUN|I2C_MCS_ACK); // corro, ack
    esperar();
    for(i=0;i<300;i++){ //Delay
    minutos=(I2C0_MDR_R&0xFF); //El Master lee lo que envía el DS1307

    I2C0_MCS_R=(I2C_MCS_RUN|I2C_MCS_ACK); //corro, ack
    esperar();
    for(i=0;i<300;i++){ //Delay
    horas=(I2C0_MDR_R&0xFF); //El Master lee lo que envía el DS1307



    I2C0_MCS_R=(I2C_MCS_RUN|I2C_MCS_ACK); // corro, ack
    esperar();
    for(i=0;i<300;i++){ //Delay
    dia=(I2C0_MDR_R&0xFF); //El Master lee lo que envía el DS1307

    I2C0_MCS_R=(I2C_MCS_RUN|I2C_MCS_ACK); // corro, ack
    esperar();
    for(i=0;i<300;i++){ //Delay
    fecha=(I2C0_MDR_R&0xFF); //El Master lee lo que envía el DS1307

    I2C0_MCS_R=(I2C_MCS_RUN|I2C_MCS_ACK); // corro, ack
    esperar();
    for(i=0;i<300;i++){ //Delay
    mes=(I2C0_MDR_R&0xFF); //El Master lee lo que envía el DS1307

    I2C0_MCS_R=(I2C_MCS_RUN|I2C_MCS_ACK); // corro, ack
    esperar();
    for(i=0;i<300;i++){ //Delay
    anio=(I2C0_MDR_R&0xFF); //El Master lee lo que envía el DS1307

    I2C0_MCS_R=(I2C_MCS_STOP|I2C_MCS_RUN); //corro, alto
}
```

	<b>UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO</b> <b>FACULTAD DE INGENIERÍA</b> <b>Microprocesadores y Microcontroladores</b>		Semestre: 2020-2	 <b>INGENIERÍA</b> <b>ELÉCTRICA ELECTRÓNICA</b>
	<b>Prof. Dr. Saúl De La Rosa Nieves</b>	<b>Grupo 3</b>	Página 13 de 21	

/\*\* PROGRAM PRINCIPAL \*\*/

```

void main(){

    SysTick_Init();
    SSI0_init(); // Función que habilita el SPI-GPIO's-ADC
    int delay ;
    int m;
    I2C_Init(); //Función que inicializa los relojes, el GPIO y el I2C0

    //Inicializo Slave
    while(I2C0_MCS_R&0x00000001){}; // espera que el I2C esté listo

    //Para transmitir

    CargarFecha(); // Función para configurar al esclavo RTC DS1307

    SYSTCL_RCGCUART_R |= 0x00000080; // activa el reloj para el UART7 (p.388)
    SYSTCL_RCGCGPIO_R |= 0x00007EFF; // active el reloj para el Puerto C (p.382)

    while((SYSTCL_PRUART_R&SYSTCL_PRUART_R0) == 0){}; // Se espera a que el reloj se estabilice (p.505)
    UART7_CTL_R &= ~0x00000001; // se deshabilita el UART (p.1188)

    UART7_IBRD_R = 104; // IBRD = int(16,000,000 / (16 * 115,200)) = int(8.681) (p.1184)
    UART7_FBRD_R = 10; // FBRD = round(0.6806 * 64) = 44 (p. 1185)
    // Palabra de 8 bits (sin bits de paridad, un bit stop, FIFOs) (p. 1186)

    UART7_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);

    // UART toma su reloj del la fuente alterna como se define por SYSTCL_ALTCLKCFG_R (p. 1213)

    UART7_CC_R = (UART7_CC_R&~UART_CC_CS_M)+UART_CC_CS_PIOSC;

    // La fuente de reloj alterna es el PIOSC (default)(P. 280)

    SYSTCL_ALTCLKCFG_R = (SYSTCL_ALTCLKCFG_R&~SYSTCL_ALTCLKCFG_ALTCLK_M)+SYSTCL_ALTCLKCFG_ALTCLK_PIOSC; //

    // alta velocidad deshabilitada;divide el reloj por 16 en lugar de 8 (default)(1208)

    UART7_CTL_R &= ~0x00000020;
    UART7_CTL_R |= 0x00000001; // habilita el UART (p.1208)

    // Se espera a que el reloj se estabilice

    while((SYSTCL_PRGPIO_R&0x00000004) == 0){};
    GPIO_PORTC_AFSEL_R |= 0x30; // habilita función alterna en PC5-4
    GPIO_PORTC_DEN_R |= 0x30; // habilita digital I/O en PC5-4

    // configura PC5-4 como UART

    GPIO_PORTC_PCTL_R = (GPIO_PORTC_PCTL_R&0xFF00FFFF)+0x00110000;
    GPIO_PORTC_AMSEL_R &= ~0x30; // deshabilita la funcionabilidad analogica de PC

    GPIO_PORTD_DATA_R=0xFF; //Apaga todos los puertos de segmentos para que no se traslape su muestra

    GPIO_PORTM_DIR_R |= 0xFF; // Salidas de PM0-PM7
    GPIO_PORTM_DEN_R |= 0xFF; // Habilita PM0-PM7
    GPIO_PORTM_PDR_R |= 0xFF; // Habilita resistencias de pull-down para salidas del display 7 segmentos

    do{
    if((UART7_FR_R&0x00000010) != 0) // Se espera a que el FIFO TX este vacio
    {
        leerFecha();

        switch(datin)
        {
            case 0x0C:
                frecuencia=0.5;
                break;

```



```
case 0x0F:
    frecuencia=0.1;
break;
case 0x01:

break;
case 0x02:

break;
}

conversionD();

//senoidal
signaltype=4;
delay = ((1/frecuencia)*4000000)/32;
for (m = 0; m < 180; m=m+10){
    x=(2*3.14159265358979*m)/360; //grados a radianes
    // aproximacion con serie de taylor
    val= x - ((x*x*x)/(2*3)) + ((x*x*x*x*x)/(2*3*4*5)) - ((x*x*x*x*x*x*x)/(2*3*4*5*6*7)) +
    ((x*x*x*x*x*x*x*x*x)/(2*3*4*5*6*7*8*9)) - ((x*x*x*x*x*x*x*x*x*x)/(2*3*4*5*6*7*8*9*10*11)) ;
    val=(val*100);
    val_int = val+150;
    pot_setVal(val_int);
    ADC0_PSSI_R = 0x0008; // Inicia conversión del SS3
    while ((ADC0_RIS_R & 0x08)==0); // Espera a que SS3 termine conversión (polling)
    adc_result = (ADC0_SSIFIF03_R & 0xFFF); // Resultado en FIF03 se asigna a variable "result"
    ADC0_ISC_R = 0x0008; // Limpia la bandera RIS del ADC0
    SysTick_Wait(delay);
} //end FOR
for (m = 180; m > 0; m=m-10){
    x=(2*3.14159265358979*m)/360; //grados a radianes
    // aproximacion con serie de taylor
    val= x - ((x*x*x)/(2*3)) + ((x*x*x*x*x)/(2*3*4*5)) - ((x*x*x*x*x*x*x)/(2*3*4*5*6*7)) +
    ((x*x*x*x*x*x*x*x*x)/(2*3*4*5*6*7*8*9)) - ((x*x*x*x*x*x*x*x*x*x)/(2*3*4*5*6*7*8*9*10*11)) ;
    val=(-(val*100));
    val_int = val+150;
    pot_setVal(val_int);
    ADC0_PSSI_R = 0x0008; // Inicia conversión del SS3
    while ((ADC0_RIS_R & 0x08)==0); // Espera a que SS3 termine conversión (polling)
    adc_result = (ADC0_SSIFIF03_R & 0xFFF); // Resultado en FIF03 se asigna a variable "result"
    ADC0_ISC_R = 0x0008; // Limpia la bandera RIS del ADC0
    SysTick_Wait(delay);
} //END for
}else
{
    data=UART7_DR_R&0xFF; //Se guarda el primer valor
    datin=data;
}
}while(error!=1);
}

void SSI0_init (void) {
    SYSCCTL_RCGCSSI_R = SYSCCTL_RCGCSSI_R0; // Activa reloj al SSI0
    while ((SYSCCTL_PRSSI_R & SYSCCTL_PRSSI_R0) == 0); // Espera a que este listo
    SYSCCTL_RCGCGPIO_R |= 0x7EFF; // Activa reloj del GPIO A/K
    while ((SYSCCTL_PRGPIO_R & SYSCCTL_PRGPIO_R0) == 0); // Espera a que este listo
    SYSCCTL_RCGCADR_R = 0x01; // 6) Habilita reloj para ADC0
    while((SYSCCTL_PRADC_R&0x01)==0); // Espera a reloj este listo

    //PORT A
    GPIO_PORTA_AHB_AFSEL_R |= 0x3C; // Selecciona la función alterna de PA[2:5].
    GPIO_PORTA_AHB_PCTL_R = (GPIO_PORTA_AHB_PCTL_R & 0xFF0000FF) | 0x00FFFF00; // Configura las terminales de PA a su
    función de SSI0.
    GPIO_PORTA_AHB_AMSEL_R = 0x00; // Deshabilita la función analógica
    GPIO_PORTA_AHB_DIR_R = (GPIO_PORTA_AHB_DIR_R & ~0x3C) | 0x1C; // Configura al puerto como salida
    GPIO_PORTA_AHB_DEN_R |= 0x3C; // Habilita la función digital del puerto
    //PORT E
    GPIO_PORTE_AHB_DIR_R = 0x00; // 2) PE4 entrada (analógica)
    GPIO_PORTE_AHB_AFSEL_R |= 0x10; // 3) Habilita Función Alterna de PE4
    GPIO_PORTE_AHB_DEN_R = 0x00; // 4) Deshabilita Función Digital de PE4
```



```
GPIO_PORTE_AHB_AMSEL_R |= 0x10; // 5) Habilita Función Analógica de PE4
//ADC
ADC0_PC_R = 0x01; // 7) Configura para 125Ksamp/s ( 1 octavo de la frecuencia de conversión configurada)(p.1159)
ADC0_SSRI_R = 0x0123; // 8) SS3 con la más alta prioridad (p.1099)
ADC0_ACTSS_R = 0x0000; // 9) Deshabilita SS3 antes de cambiar configuración de registros (p. 1076)
ADC0_EMUX_R = 0x0000; // 10) iniciar muestreo por software (p.1091)
ADC0_SSEMUX3_R = 0x00; // 11) Entradas AIN(15:0) (p.1146)
ADC0_SSMUX3_R = (ADC0_SSMUX3_R & 0xFFFFFFF0) + 9; // canal AIN9 (PE4) (p.1141)
ADC0_SSCTL3_R = 0x0006; // 12) Si: sensor de temperatura, Habilitación de INR3, Fin de secuencia; No:muestra diferencial (p.1142)
ADC0_IM_R = 0x0000; // 13) Deshabilita interrupciones de SS3(p. 1081)
ADC0_ACTSS_R |= 0x0008; // 14) Habilita SS3 (p.1077)
SYSCTL_PLLFREQ0_R |= SYSCTL_PLLFREQ0_PLLPWR; // encender PLL
while((SYSCTL_PLLSTAT_R&0x01)==0); // espera a que el PLL fije su frecuencia
SYSCTL_PLLFREQ0_R &= ~SYSCTL_PLLFREQ0_PLLPWR; // apagar PLL
// Se recomienda Limpiar la bandera RIS del ADC0
ADC0_ISC_R = 0x0004;
//PSI
SSI0_CR1_R = 0x00; // Selecciona modo maestro/deshabilita SSI0. (p. 1247)
SSI0_CPSR_R = 0x02; // preescalador (CPSDVSR) del reloj SSI (p. 1252)
// configura para Freescale SPI; 16bit; 4 Mbps; SPO = 0; SPH = 0 (p. 1245)
SSI0_CR0_R = (0x0100 | SSI_CR0_FRF_MOTO | SSI_CR0_DSS_16) & ~(SSI_CR0_SPO | SSI_CR0_SPH);
SSI0_CR1_R |= SSI_CR1_SSE; // Habilita SSI0.
} //END SSI0_init
void SSI0_sendData (uint16_t dat) {
    // Envía dato de 16-bit
    while ((SSI0_SR_R & SSI_SR_BSY) != 0); // espero si el bus está ocupado
    SSI0_DR_R = dat; // envia dato.
}
void pot_setVal(uint8_t slider) {
    //Combine el valor del control deslizante con el código de comando de escritura.
    // Estructura del mensaje SPI: [comando (8-bits)][deslizador (8-bits)]
    SSI0_sendData(0x1100 | slider);
}
void SysTick_Init(void){ NVIC_ST_CTRL_R = 0;
    NVIC_ST_RELOAD_R = NVIC_ST_RELOAD_M;
    NVIC_ST_CURRENT_R = 0; NVIC_ST_CTRL_R = 0x00000001;
}
void SysTick_Wait(uint32_t retardo){
    //ECUACION
    //retardo=T[s]*4 000 000
    NVIC_ST_RELOAD_R= retardo-1; //número de cuentas por esperar
    NVIC_ST_CURRENT_R = 0; while((NVIC_ST_CTRL_R & 0x00100000)==0){} //espera hasta que la bandera COUNT sea valida
}
void conversionD()//Función que convierte lectura de 12C a conversión en 7S para 14 dígitos
{
    int digitoRTC;
    int sevens;
    int sevenar;
    int j=0;
    int k=0;
    int n=0;

    for(j=0;j<=6;j++)
    {
        if(j==0)
        {
            segundos=convertidorsin(segundos);//Convierte dato
            digitoRTC=segundos;//Iguala variable
        }
        if(j==1)
        {
            minutos=convertidorsin(minutos);//Convierte dato
            digitoRTC=minutos;//Iguala variable
        }
        if(j==2)
        {
            horas=convertidorsin(horas);//Convierte dato
            digitoRTC=horas;//Iguala variable
        }
    }
}
```





```
if(j==3)
{
    dia=convertidorsin(dia);//Convierte dato
    digitoRTC=dia;//Iguala variable
}
if(j==4)
{
    digitoRTC=fecha;//Iguala variable, como es menor a 9 no se necesita convertir
}
if(j==5)
{
    mes=convertidorsin(mes);//Convierte dato
    digitoRTC=mes;//Iguala variable
}
if(j==6)
{
    anio=convertidorsin(anio);//Convierte dato
    digitoRTC=anio;//Iguala variable
}

dig1=digitoRTC&0x0F;//Obtiene digito1
dig2=digitoRTC&0xF0;//Obtiene digito2
digdec=dig2+dig1;//Conversión a decimal
dec3=digdec/100;//Obtiene unidades
dec2=(digdec-dec3*100)/10;//Obtiene decenas
dec1=(digdec-dec3*100-dec2*10);//Obtiene centenas
unidadesd[n]=dec1;//Almacena unidades
decenasd[n]=dec2;//Almacena decenas
n=n+1;//Incremento
}
for(j=0;j<=13;j++)//For para ir llenando los arreglos
{
    for(k=0;k<=1;k++)
    {
        if(k==0)
        {
            sevens=decenasd[j];//Almacena en el arreglo para ser mostrado
        }
        if(k==1)
        {
            sevens=unidadesd[j];//Almacena en el arreglo para ser mostrado
        }
        if(sevens==1)
        {
            sevenar=0x30;//Almacena en el arreglo para ser mostrado
        }
        if(sevens==2)
        {
            sevenar=0x6D;//Almacena en el arreglo para ser mostrado
        }
        if(sevens==3)
        {
            sevenar=0x79;//Almacena en el arreglo para ser mostrado
        }
        if(sevens==4)
        {
            sevenar=0x33;//Almacena en el arreglo para ser mostrado
        }
        if(sevens==5)
        {
            sevenar=0x5B;//Almacena en el arreglo para ser mostrado
        }
        if(sevens==6)
        {
            sevenar=0x5F;//Almacena en el arreglo para ser mostrado
        }
        if(sevens==7)
        {
            sevenar=0x70;//Almacena en el arreglo para ser mostrado
        }
        if(sevens==8)
```





```
{
    sevenar=0x7F;//Almacena en el arreglo para ser mostrado
}
if(sevens==9)
{
    sevenar=0x7B;//Almacena en el arreglo para ser mostrado
}
if(sevens==0)
{
    sevenar=0x7E;//Almacena en el arreglo para ser mostrado
}
if(k==0)
{
    segmentosd[j]=sevenar;//Almacena en el arreglo para ser mostrado
}
if(k==1)
{
    segmentosu[j]=sevenar;//Almacena en el arreglo para ser mostrado
}
}
}
for(i=0;i<=100;i++)
{
    GPIO_PORTD_DATA_R=0xFF;//Apaga puerto para que se muestre solo el que se requiere

    for(i=0;i<=10000;i++)
    {
        GPIO_PORTD_DATA_R=0xFE;//Habilita dígito 4
        GPIO_PORTM_DATA_R=segmentosu[recorrido];;//Enciende dígito 4
        GPIO_PORTM_DATA_R=0x00;//Apaga los 7 segmentos
        GPIO_PORTD_DATA_R=0xFD;//Habilita dígito 3
        GPIO_PORTM_DATA_R=segmentosd[recorrido];;//Enciende dígito 3
        GPIO_PORTM_DATA_R=0x00;//Apaga los 7 segmentos
        GPIO_PORTD_DATA_R=0xFB;//Habilita dígito 2
        GPIO_PORTM_DATA_R=segmentosu[recorrido+1];;//Enciende dígito 2
        GPIO_PORTM_DATA_R=0x00;//Apaga los 7 segmentos
        GPIO_PORTD_DATA_R=0xF7;//Habilita dígito 1
        GPIO_PORTM_DATA_R=segmentosd[recorrido+1];;//Enciende dígito 1
        GPIO_PORTD_DATA_R=0xFF;//Apaga puerto para que se muestre solo el que se requiere
        GPIO_PORTM_DATA_R=0x00;
    }
    recorrido=recorrido+2;
    if(recorrido>=8)
    {
        recorrido=0;
    }
}
}

int convertidorsin(int digitc)//Función que convierte el ciclo de 0-90 a 0-60
{
    if(digitc<=9)
    {
        digitc=digitc;//Conversión de datos
    }
    else if(digitc<=25)
    {
        digitc=digitc-6;//Conversión de datos
    }
    else if(digitc<=41)
    {
        digitc=digitc-12;//Conversión de datos
    }
    else if(digitc<=57)
    {
        digitc=digitc-18;//Conversión de datos
    }
    else if(digitc<=73)
    {

```

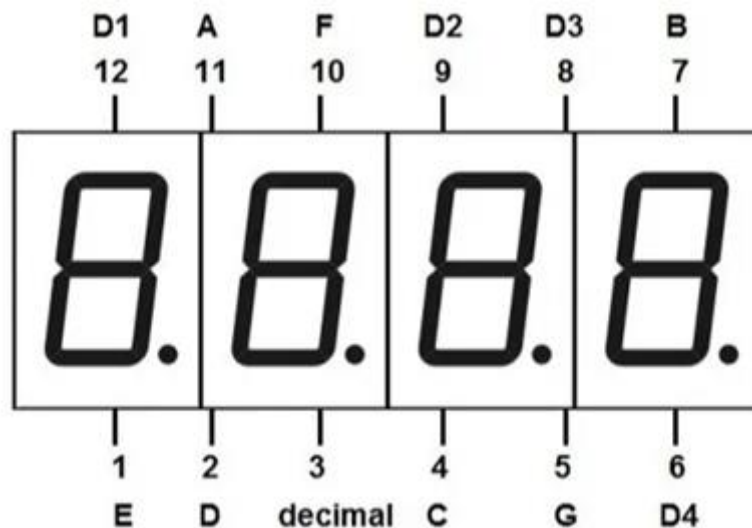


```
    digitc=digitc-24;//Conversión de datos
}
else if(digitc<=90)
{
    digitc=digitc-30;//Conversión de datos
}
return digitc;
}
```

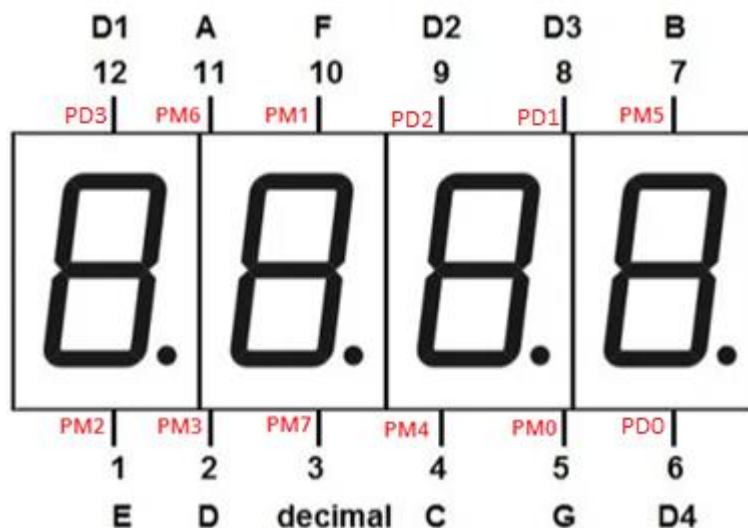
## VI. Construcción

### Conexión al display 7 segmentos

Para la conexión del display 7 segmentos se utilizó el puerto M y el puerto D, un total de 12 pines conectados, 6 y 4 respectivamente. En este caso el display 7 segmentos tiene la siguiente configuración:

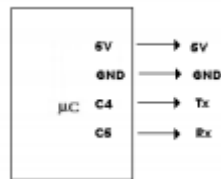


Para realizar las conexiones utilicé el puerto D para la habilitación de los dígitos y el puerto M para los segmentos por lo que la conexión realizada al microcontrolador fue la siguiente.

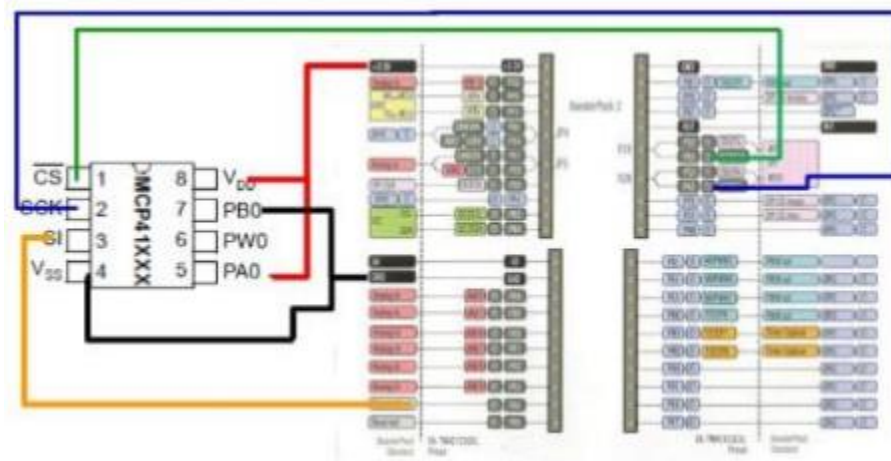




### Conexión módulo bluetooth



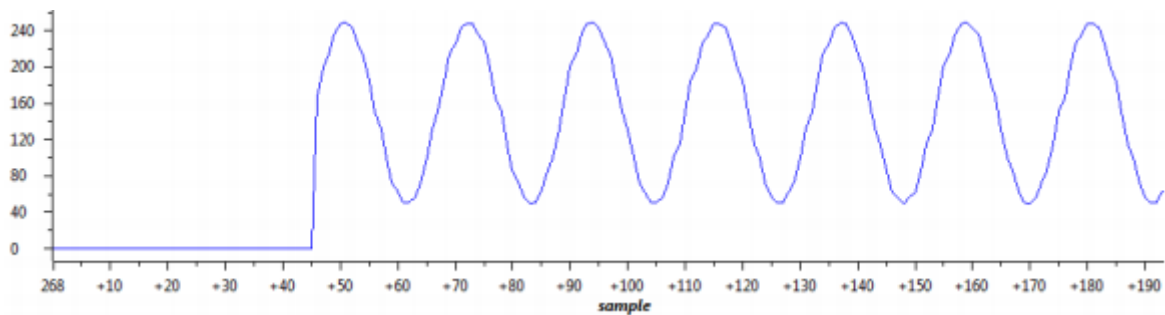
### Conexión SPI



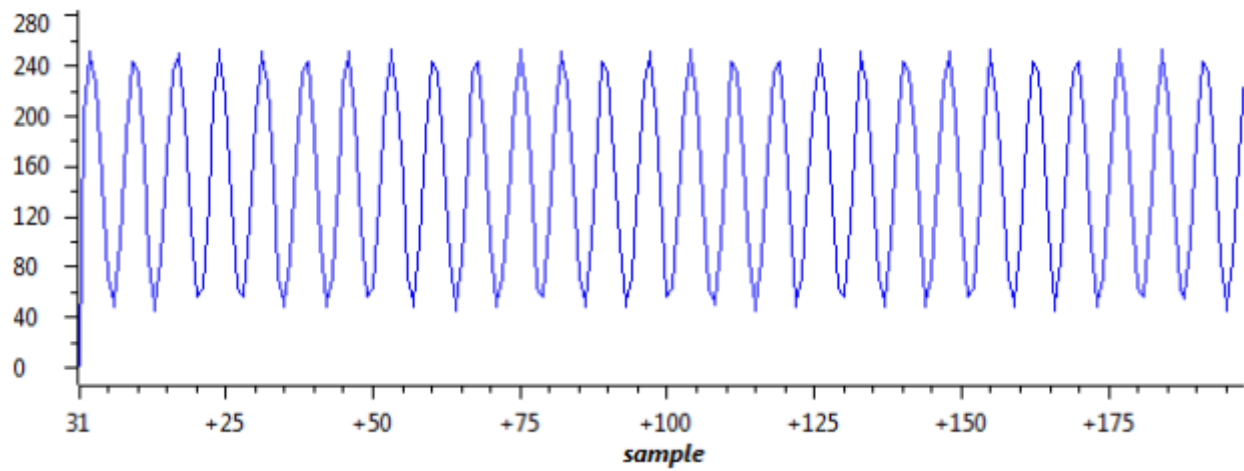
## VII. Resultados y – Conclusiones

Señales senoidales de salida:

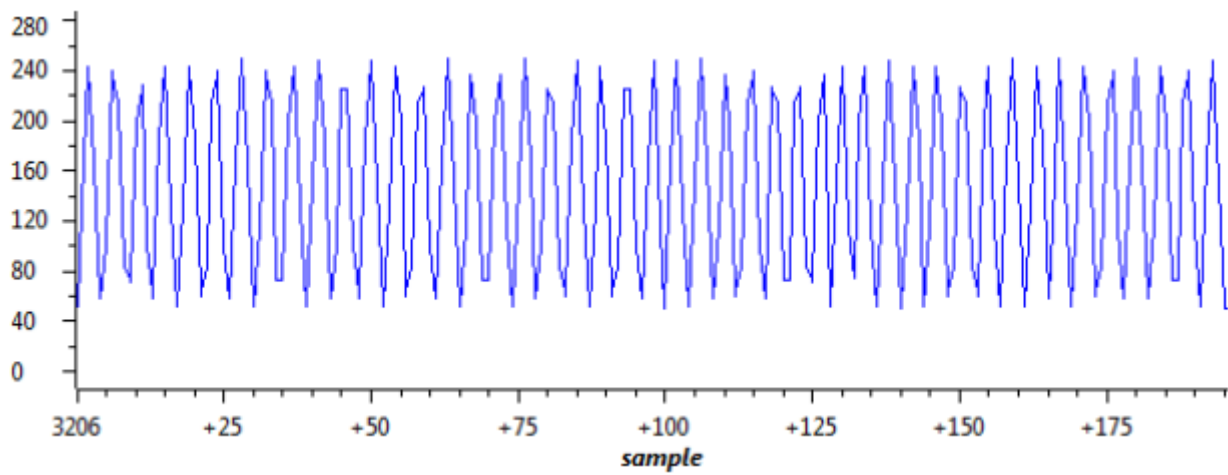
0.1Hz:



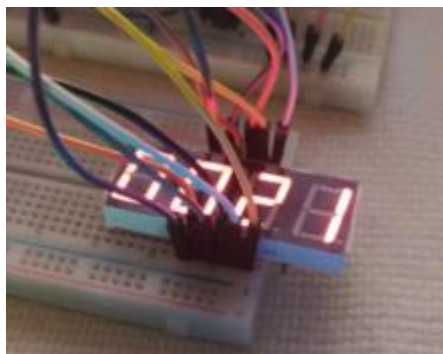
0.3 Hz:



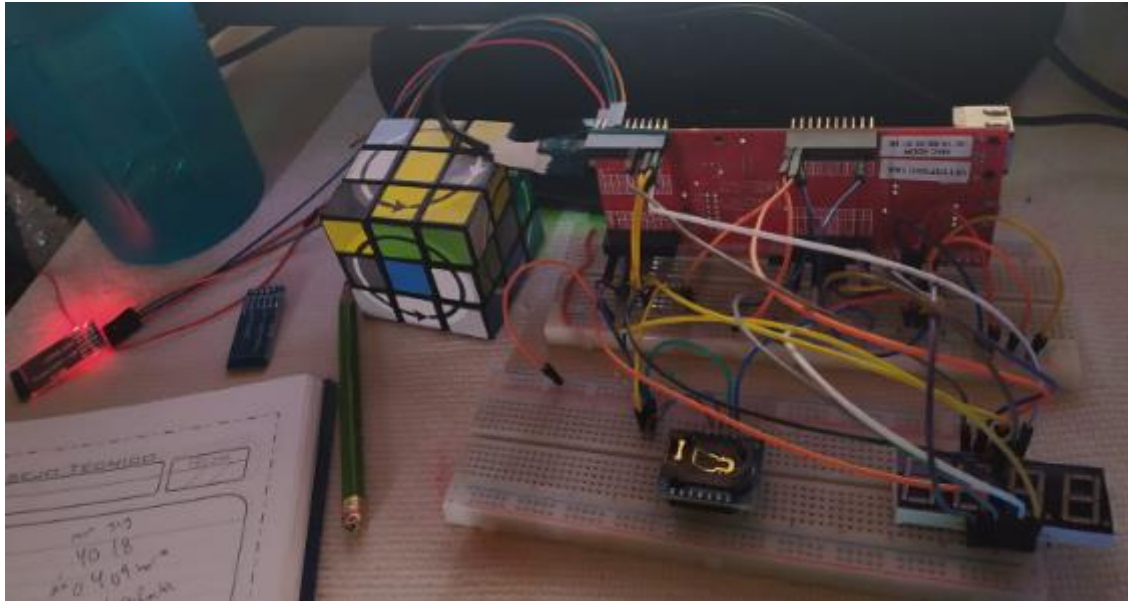
0.5 Hz:



Muestra del año en curso en el display 7 segmentos:



Proyecto alambrado:



En conclusión pudimos acoplar tres protocolos de comunicación en un mismo periodo con sus limitantes, esto porque para que todo funcione a la par perfectamente se necesitan interrupciones para poder mostrar los datos del display y a la vez poder recibir datos del SPI e I2C al momento, si pones la etapa de muestra de datos bastante grande si se modifica un poco la señal senoidal y se muestra recortada en algunas zonas por el tiempo que utiliza, fuera de eso fue una buena práctica para recordar lo visto en la materia anterior.