Microcontroladores y Electrónica de potencia

Trabajo Integrador:

Domótica

Alumno: Stefania Giannina Arias Farias

Legajo: 9668

Tabla de contenido

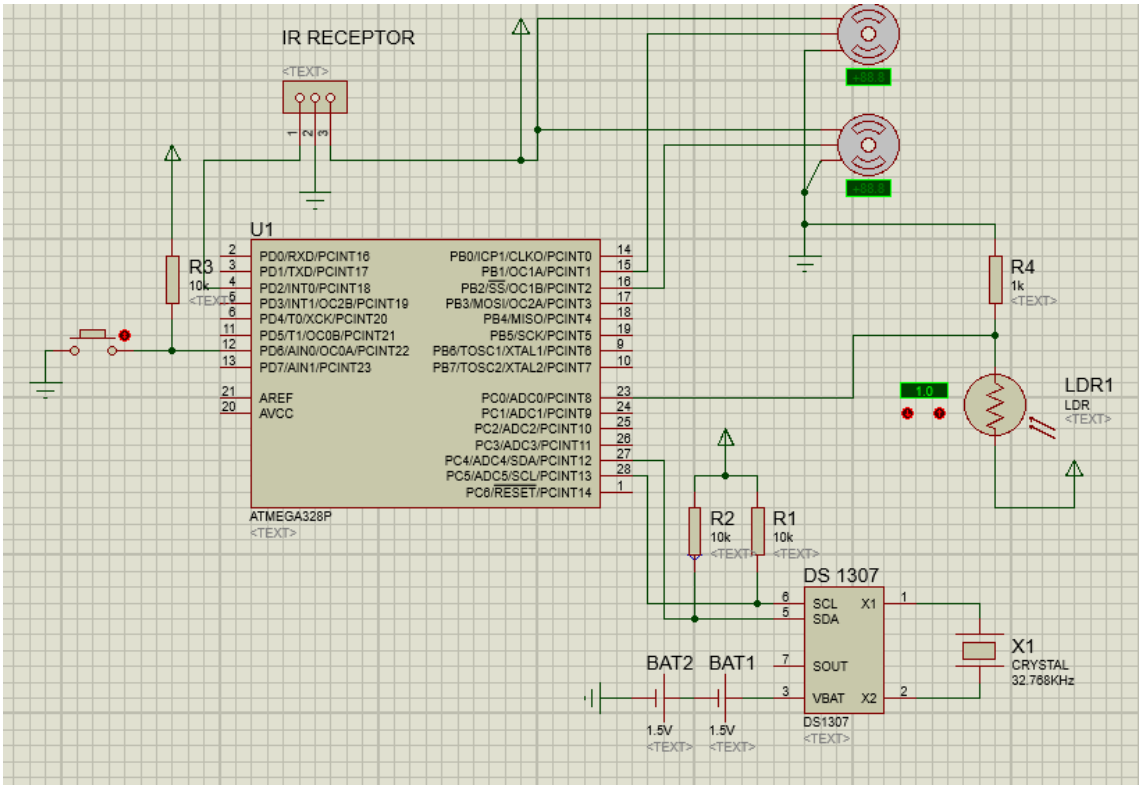
Introducción

El sistema desarrollado a lo largo de este informe consiste en la automatización de una casa con la utilización de un sensor LDR , un sensor de tiempo real DS1307 con protocolo i2c, un receptor infrarojo y un pulsador simulando la entrada a la casa. Todo esto será controlado por el Atmega328p, el cual viene integrado en la placa ArduinoUno.

El propósito de este proyecto es cumplimentar una instancia evaluativa que consiste en la realización de un proyecto que integre los conceptos aprendidos a lo largo de la materia. Las herramientas utilizadas fueron: interrupciones, comunicación serie UART, PWM, i2c, conversión analógica-digital.

Asi también, se realizó debido al amplio desarrollo en el campo de la domotica y grandes posibilidades de desarrollo a futuro.

Esquema Tecnológico

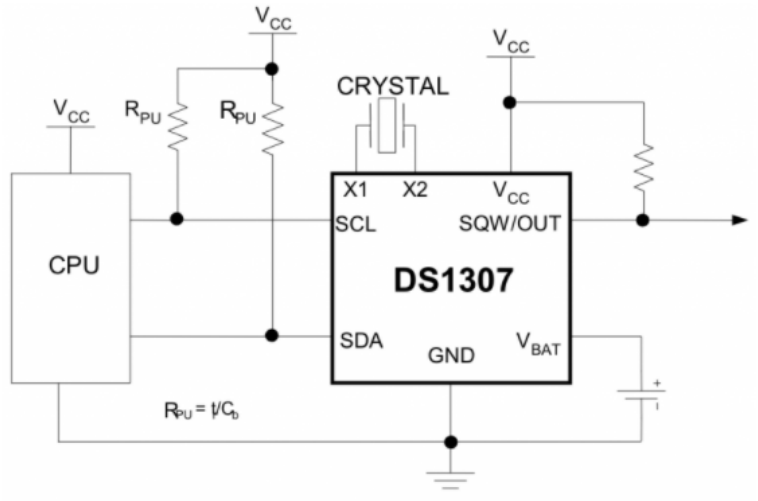


Detalle de módulos

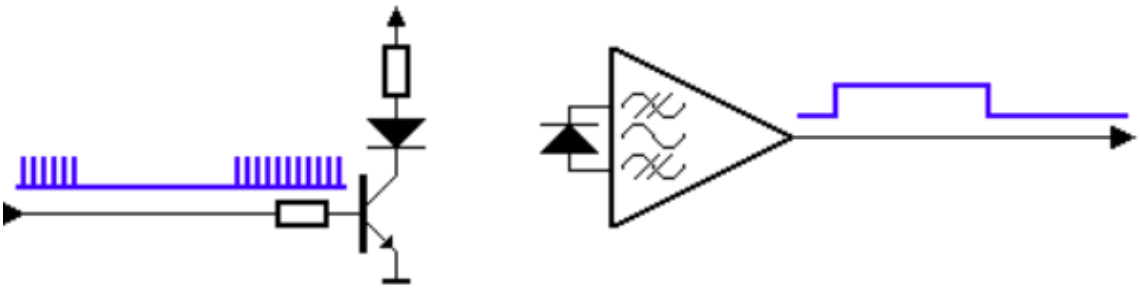
- **DS1307:** Es un circuito electrónico especializado cuya función es mantener la hora y fecha actual en un sistema informático (ya sea con microcontrolador u otro tipo de CPU). Se caracteríza por tener un bajo consumo de energía y también normalmente su propia fuente de alimentación auxiliar. Al recurrir a este tipo de circuitos integrados es de esperar que se obtenga una mejor precisión en la cuenta del tiempo. Un ejemplo de dispositivos que incluyen relojes en tiempo real son las computadoras personales (PC).

  - *Para la comunicación a través de I2C, el DS1307 requiere de dos resistencias pull-up en los pines SDA y SCL. Si vamos a utilizar el pin SQW/OUT, este también requiere una resistencia pull-up para funcionar correctamente.*
  - *El DS1307 requiere un cristal de 32.768 Khz, este valor viene dado por el hecho de que 2^15 = 32,768. Esto quiere decir que la frecuencia es divisible binariamente para generar un segundo exacto. El cristal ya se incluye si compramos algún módulo con el DS1307.*

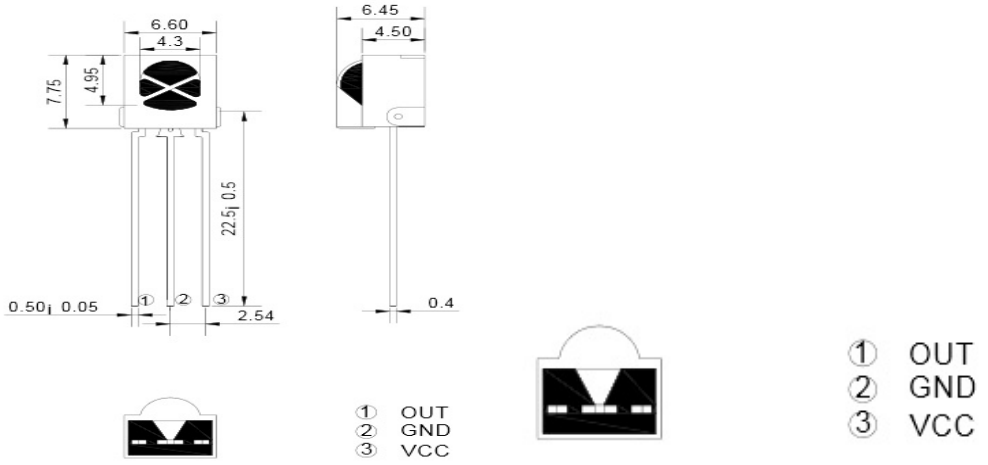
- *El DS1307 requiere dos fuentes de alimentación: Por una parte, requiere alimentación de 5 volts que opera mientras el circuito esta encendido y funcionando y otra fuente de poder que proviene de una batería de litio (tipo reloj) que mantiene funcionando el reloj/calendario mientras la alimentación principal NO esta disponible. El cambio entre ambas fuentes de alimentación es gestionado por el DS1307 de manera automática.*



→ **VS1838B:** es un receptor infrarojo que recibe un tren de ondas estable, el cual el receptor traduce como un bajo.



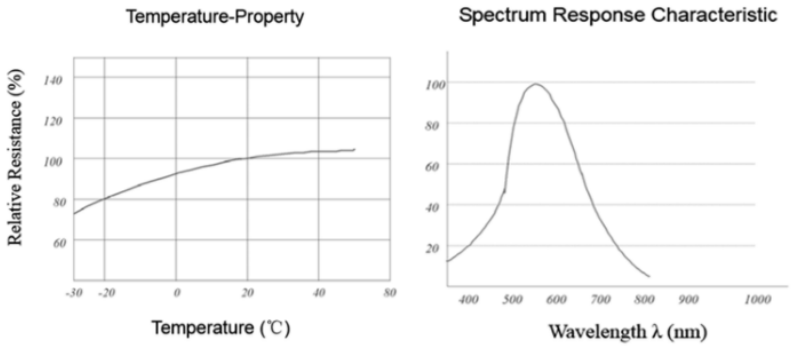
Conexión a la breadboard:



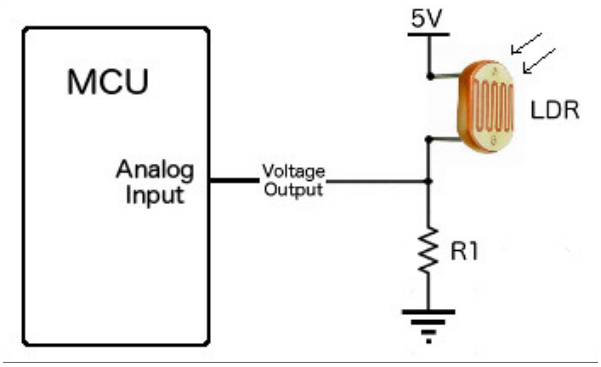
- **Sensor LDR:** Componente cuya resistencia varía sensiblemente con la cantidad de luz percibida. La relación entre la intensidad lumínica y el valor de la resistencia no es lineal. Se utiliza ampliamente para medir la iluminación en dispositivos electrónicos que requieren un precio agresivo. Su comportamiento es el siguiente:

  Mas luz = menor resistencia eléctrica

  Menos luz = mayor resistencia eléctrica

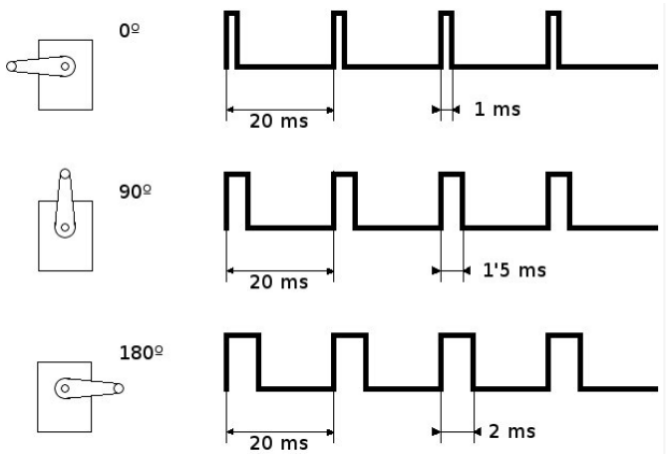


  Conexión a la breadboard:

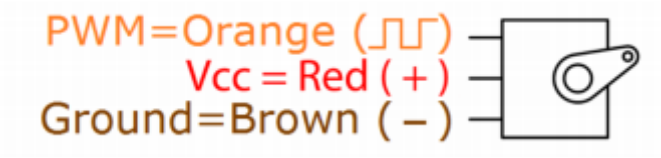


- **Servomotores SG90:** Un servomotor es un dispositivo que dispone en su interior de un motor de corriente continua, un reductor y un circuito de control, esto le proporciona la capacidad de ubicarse en cualquier posición dentro de su rango de operación y mantenerse estable en dicha posición.

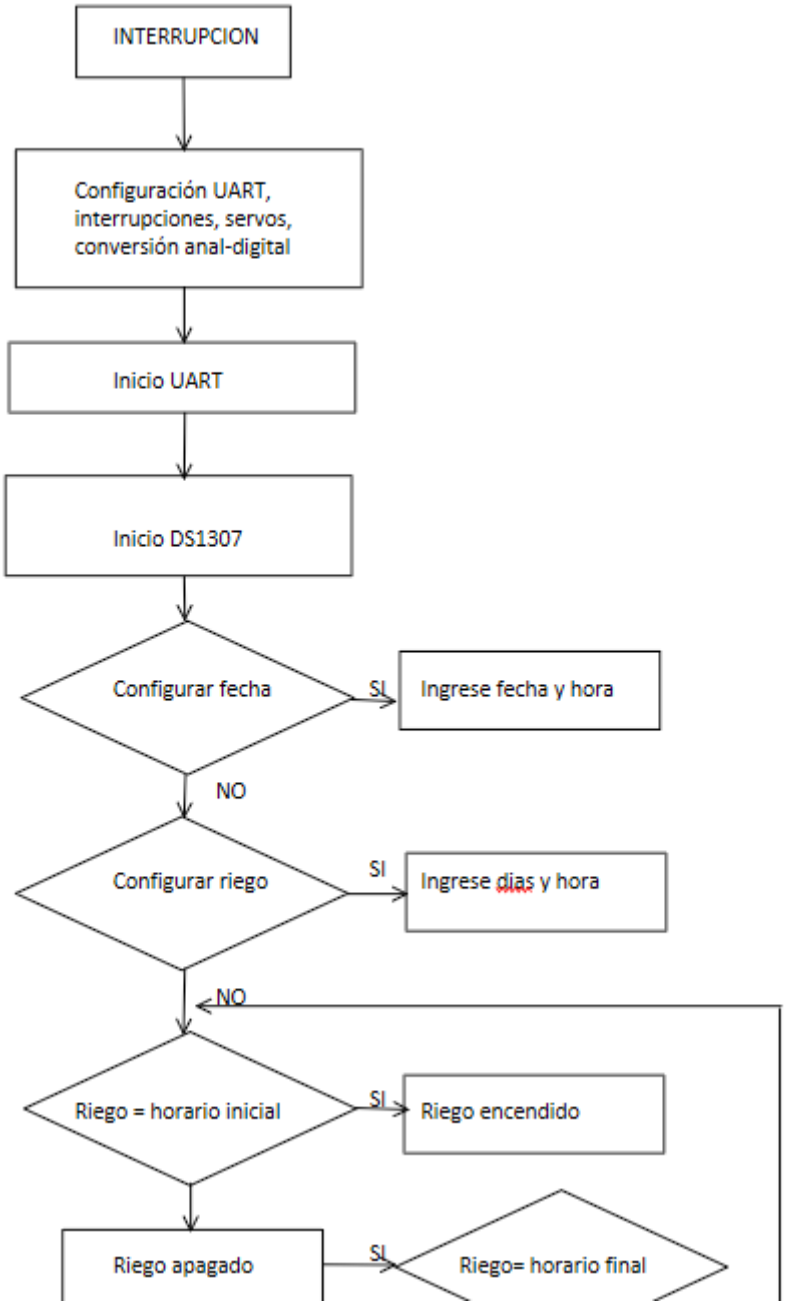
  El funcionamiento delos servomotores es a través de señal de pulso a una frecuencia de 50Hz (periodo de 20ms) y para poder controlar la posición del servomotor se necesita un ciclo de trabajo positivo entre 0.5ms y 2.5ms según indique el fabricante. A continuación, se muestra una imagen que detalla lo explicado.

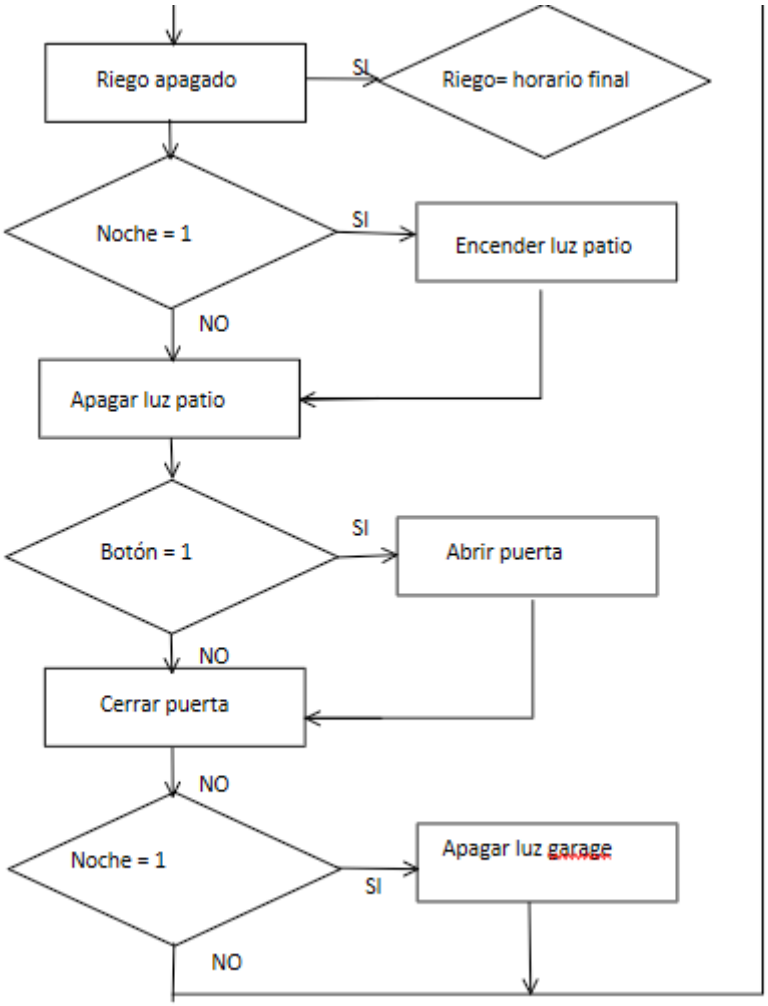


Conexión a la breadboard:

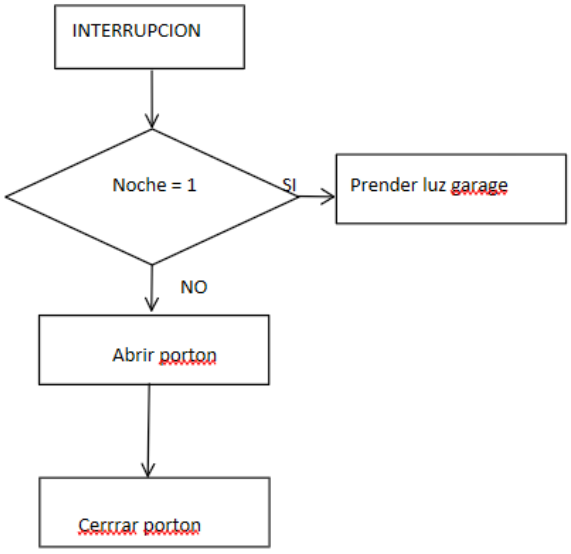


Funcionamiento general


INTERRUPCION
Configuración UART, interrupciones, servos, conversión anal-digital
Inicio UART
Inicio DS1307
Configurar fecha
SI
Ingrese fecha y hora
NO
Configurar riego
SI
Ingrese dias y hora
NO
Riego = horario inicial
SI
Riego encendido
Riego apagado
SI
Riego= horario final



Rutina de interrupción



Programación

I. Configuracion de PWM

PWM quiere decir modulación por ancho de pulso; cuando se tiene una onda rectangular de un periodo fijo o de una frecuencia fija, a la parte de la onda rectangular que está a mayor nivel o en alto se le llama pulso el cual tendrá un ancho, mediante el PWM se modifica el ancho de ese pulso. El uso del PWM tiene mucha importancia en el control de dispositivos en diversos modos como son el control de velocidad de motores, control de iluminación, control de temperatura, de transistores, entre otros.
Para manejar los PWM se eligio el timer 1 ya que es de 16 bits, y la configuración de los PWM en modo rápido. Para configurarlos se procedió de la siguiente manera:
Se habilitaron los pines PINB1 y PINB2 como salidas.

Timer / Counter1 tiene 2 salidas, OC1A y OC1B. Dado que estas dos salidas se ejecutan en el mismo temporizador y los generadores de forma de onda, tanto OC1A como OC1B están sincronizados, también es capaz de funcionar en 3 modos, el modo PWM rápido, el modo PWM corregido de fase y el modo de corrección de fase y frecuencia. Cada uno de estos modos puede invertirse o no invertirse. Al igual que Timer / Counter0 Timer / Counter1 tiene varias opciones para controlar el valor TOP de PWM a diferencia de Timer / Counter0, sin embargo, el valor TOP puede ser fijo, almacenado en el registro OCR1A o en el registro ICR1

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10

Registro de control de temporizador / contador 1 A

COM1A1 COM1B1	COM1A0 COM1B0	DESCRIPCIÓN
0	0	Operación de puerto normal, OC1A / OC1B desconectado.
0	1	Modo 9,11,14,15 solamente: habilite OCR1A solamente (OC1B desconectado)
1	0	Modo ninguno invertido (ALTO en la parte inferior, BAJO en la coincidencia)
1	1	Modo invertido (BAJO en la parte inferior, ALTO en la coincidencia)

Se aplica solo a los modos PWM

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Registro de control de temporizador / contador 1 B

Se elige modo no invertido para eso se ponen alto los pines COM1A1 y COM1B1.

CS12	CS11	CS10	DESCRIPCIÓN
0	0	0	Timer / Counter2 desactivado
0	0	1	Sin preescalado
0	1	0	Reloj / 8
0	1	1	Reloj / 64
1	0	0	Reloj / 256
1	0	1	Reloj / 1024
1	1	0	Fuente de reloj externa en el pin T1, Reloj en el borde descendente
1	1	1	Fuente de reloj externa en el pin T1, reloj en el borde ascendente

Bits CS

Se elige un prescaler de 64 bit por lo que se ponen en alto CS11 y CS10.

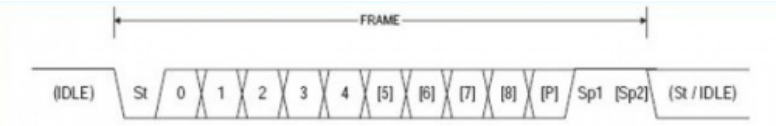
MODO	WGM13	WGM12	WGM11	WGM10	DESCRIPCIÓN	PARTE SUPERIOR
0	0	0	0	0	Normal	0xFFFF
1	0	0	0	1	PWM, fase corregida, 8 bits	0x00FF
2	0	0	1	0	PWM, fase corregida, 9bit	0x01FF
3	0	0	1	1	PWM, fase corregida, 10 bits	0x03FF
5	0	1	0	1	PWM rápido, 8 bits	0x00FF
6	0	1	1	0	PWM rápido, 9 bits	0x01FF
7	0	1	1	1	PWM rápido, 10 bits	0x03FF
8	1	0	0	0	PWM, fase y frecuencia corregidas	ICR1
9	1	0	0	1	PWM, fase y frecuencia corregidas	OCR1A
10	1	0	1	0	PWM, fase correcta	ICR1
11	1	0	1	1	PWM, fase correcta	OCR1A
14	1	1	1	0	PWM rápido	ICR1
15	1	1	1	1	PWM rápido	OCR1A

Bits del modo generador de forma de onda (abreviado)

Se elige PWM modo rápido con ICR1 como parte superior del contador, ponemos en alto WGM13, WGM12, WGM11.
Los valores de ICR1, OCR1A, OCR1B se obtienen a partie de los siguientes cálculos.

II. Configuración de UART

USART (Universal Synchronous/Asynchronous Receiver Transmitter ó Transmisor-Receptor Síncrono/Asíncrono Universal) es un protocolo empleado en comunicaciones duales, es decir que está en la capacidad de recibir y transmitir simultáneamente. Los datos son transmitidos de manera serial, lo que significa que sólo un bit es transferido por el canal al tiempo. Las interfaces seriales son sencillas y baratas de implementar, motivo por el cual fueron el sistema más común de comunicación electrónica hasta la aparición del protocolo USB.



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

El formato de envío en el protocolo se basa en ventanas o *frames*, cada *frame* posee los siguientes elementos:

- Un bit de parada
- Bits de datos (5,6,7,8 o 9 según la configuración)
- Uno o dos bits de parada
- Las ventanas pueden incluir bits de paridad para la detección y corrección de errores

Con el propósito de establecer una comunicación asincrónica entre dos agentes es necesario acordar la tasa de transmisión de bytes (baudios) entre ambas partes.

En primer lugar, se presentan las respectivas definiciones e inclusiones, como la frecuencia del reloj, la tasa de baudios y la inclucion de las librerias <avr/io.h> y <util/delay.h>.

Luego se procede a inicializar el USART:

Name: UCSR0B
Offset: 0xC1
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

Name: UCSR0C
Offset: 0xC2
Reset: 0x06
Property: -

Bit	7	6	5	4	3	2	1	0
	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 / UDORD0	UCSZ00 / UCPHA0	UCPOL0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	1	0

Se ponen en alto los pines RXEN0, TXEN0,USBS y UCSZ00 de 8 bit. Se procede a crear las funciones para enviar y rebicir un byte. En los cuales se hace una transmicion de dato o un retorno del dato recibido.

III. Configuración de interrupción externas

Una interrupción es un evento que requiere que la CPU detenga la ejecución normal del programa correspondiente y realice un servicio particular. Una interrupción puede ser generada internamente o externamente. Una interrupción externa es generada cuando el dispositivo detecta un evento referente a una señal de entrada. Una interrupción interna puede ser generada por la circuitería interna del dispositivo o por errores en software.

Se usan para coordinar actividades de entrada/salida y prevenir que la CPU se atasque durante un proceso de transferencia de datos, proveer vías de salida manejables cuando ocurre un error de software y realizar tareas rutinarias.
La CPU provee un servicio de interrupción al ejecutar un programa llamado Rutina de Servicio de Interrupción (ISR). Una interrupción corresponde a un evento especial para la CPU. Luego de proveer el servicio de una interrupción, la CPU debe resumir la ejecución normal del programa. Para este propósito, se debe almacenar el valor del contador de programa y el estado de información del programa antes de ejecutar la rutina de interrupción y posteriormente restaurar el estado de la CPU con la información previamente almacenada.
Para habilitar las interrupciones debemos definir los registros EICRA y EMISK .
Con EICRA definimos a qué tipo de flanco responderá la interrupción y con EMISK que pin recibirá la interuupción, ya que hay 2 pines para interrupción externa INT0 y INT1.

Name: EICRA
Offset: 0x69
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
					ISC11	ISC10	ISC01	ISC00
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Value	Description
00	The low level of INT0 generates an interrupt request.
01	Any logical change on INT0 generates an interrupt request.
10	The falling edge of INT0 generates an interrupt request.
11	The rising edge of INT0 generates an interrupt request.

Name: EIMSK
Offset: 0x3D
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x1D

Bit	7	6	5	4	3	2	1	0
							INT1	INT0
Access							R/W	R/W
Reset							0	0

IV. Convertidos analógico- digital ADC

En primer lugar se realiza la habilitación del conversor análogo digital por medio del registro ADMUX. En la asignación realizada se observa el establecimiento del valor de referencia como el valor de alimentación del dispositivo (VCC). El registro ADCSRA es empleado para la habilitación de un prescaler que será empleado como fuente de reloj para el conversor. Esta operación se debe realizar para realizar una lectura más óptima a menor frecuencia. La modificación de la bandera ADEN es empleada para la habilitación del prescaler. Las banderas ADPSX son empleadas para definir los factores de división de la frecuencia. Para más información acerca de este registro consultar la hoja de datos del dispositivo.

Name: ADMUX
Offset: 0x7C
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Table 28-3. ADC Voltage Reference Selection

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin
10	Reserved
11	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Table 28-4. Input Channel Selection

MUX[3:0]	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5

Para leer los valores del sensor la función implementada realiza modificaciones sobre el registro ADMUX. Este registro es empleado para seleccionar el canal del ADC que será empleado en la lectura, definir el voltaje de referencia, entre otras funcionalidades. Adicionalmente, se observa la modificación del registro ADCSRA, empleado para definir el comportamiento del conversor. En este caso, la asignación realizada es empleada para habilitar la lectura. Finalmente, se observa una rutina de retardo que involucra el mismo registro, esta es empleada para esperar a que finalice la conversión previamente inicializada. Por último, el valor de la lectura almacenado en el registro ADC es retornado por la función.

Name: ADCSRA
Offset: 0x7A
Reset: 0x00
Property: -

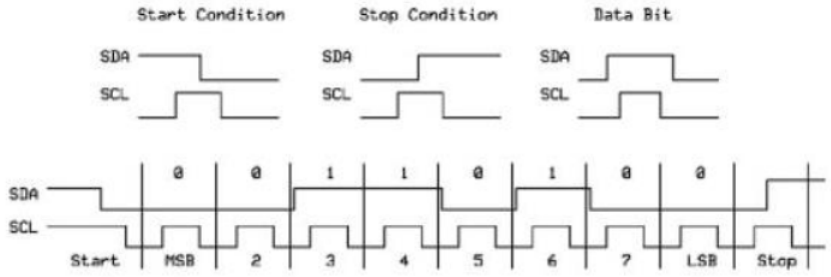
Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

V. I2c

El protocolo tiene las siguientes características:

Naturaleza sincrónica: La transferencia de datos siempre es iniciada por el dispositivo maestro. Una señal de reloj (SCL) sincroniza la transmisión de los datos. La frecuencia de reloj puede variar sin distorsionar los datos. Los datos transmitidos simplemente se modificarán de acuerdo a los cambios en la frecuencia de reloj.
Modelo de maestro y esclavo: El dispositivo maestro controla la línea de reloj (SCL). Esta línea determina la temporalidad de todas las transferencias de datos en el bus I2C.
Transferencia de datos bidireccional: Los datos a través del bus I2C pueden viajar en cualquier dirección.
Método de interfaz serial: El protocolo I2C emplea únicamente las señales SCL y SDA. La señal SCL corresponde a la señal serial de reloj, la señal SDA es conocida como señal de datos seriales. Esta señal es la encargada de la transmisión de las direcciones y los datos.
Hay tres tipos de condición manejadas por un bus I2C en un proceso de comunicación:

Condición de inicio: Es la condición que da lugar a todo el proceso de comunicación. Se caracteriza por el cambio de la señal SDA de alto a bajo mientras la señal de SCL tiene un nivel lógico alto.
Condición de parada: Es la condición que termina con el proceso de comunicación I2C. Se caracteriza por el cambio de la señal SDA de bajo a alto mientras la línea SCL tiene un nivel lógico alto.
Transferencia de datos: Luego de la condición de inicio, un bit de datos se considera válido cuando permanece estable durante el periodo en el cual la señal SCL tiene un nivel lógico alto. Los cambios en el estado de la señal SDA deben ser realizados mientras la señal SCL tiene un nivel lógico bajo.
Estas tres condiciones son descritas en la siguiente figura, la cual ilustra las diferencias correspondientes. La señal SDA cambia mientras la señal SCL tiene un nivel lógico alto para las condiciones de inicio y parada. Por otro lado, la señal SDA cambia mientras la señal SCL permanece con un nivel lógico bajo para la transmisión de bits de datos. La siguiente figura muestra la dinámica de estados previamente descrita y la transmisión completa de un valor (34h) a través del bus incluyendo las condiciones de inicio y parada.



En particular, se observa la asignación a los registros TWBR y TWSR con el propósito de establecer la frecuencia de reloj correspondiente al pin SCL (señal que emplea el maestro para la sincronización con los esclavos asociados). El registro TWCR corresponde al registro de control de la interfaz I2C y será empleado a lo largo de las rutinas implementadas para definir las acciones que deben ser ejecutadas por el módulo de control. En este caso, la asignación correspondiente se realiza para habilitar el módulo I2C.

Con el registro TWBR estamos generando la señal de clock del SCL en el modo maestro y con TWSR configura el prescaler igual a 1.

Name: TWBR
Offset: 0xB8
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Name: TWSR
Offset: 0xB9
Reset: 0xF8
Property: -

Bit	7	6	5	4	3	2	1	0
	TWS4	TWS3	TWS2	TWS1	TWS0		TWPS1	TWPS0
Access	R	R	R	R	R		R/W	R/W
Reset	1	1	1	1	1		0	0

El registro TWCR se usa para controlar la operación del i2c, activa el protocolo, lo inicia en modo maestro aplicando una condición de inicio al bus, para recibir datos y para detener la transmicion, también indica colision de escritura.

Name: TWCR
Offset: 0xBC
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN		TWIE
Access	R/W	R/W	R/W	R/W	R/W	R/W		R/W
Reset	0	0	0	0	0	0		0

Etapas de montaje y ensayos realizados

Se realizaron pruebas en proteus simulando el correcto funcionamiento de los servomotores y el DS1307.

Se probo cada sensor por separado con un código individual para cada uno filmando su correcto funcionamiento.

Tambien se realizaron pruebas con todos los dispositivos montados, con algunas fallas, las cuales corrigió el profesor.

Se tuvo que corregir el valor del OCR1A y OCR1B ya que el rango de trabajo real del servomotor sg90 oscila entre los 0,5 ms hasta los 2,5 ms para una frecuencia PWM de 50 Hz. Se alimentaron los servos por separado ya que causaba ruido en el circuito y hacia saltar interrupciones, por lo que también hubo que estabilizar el receptor ir.

Resultados, especificaciones finales

No se llego a alcanzar la totalidad de las metas, pero el programa hace lo que se le que se planteo en un principio con algunas modificaciones.

El mayor consumo de recursos lo posee al tener que imprimir todos los valores que se leen por uart, tiene presicion media , buena velocidad de respuesta y buena autonomía.

Conclusiones

El desarrollo de este proyecto fue, en parte, un gran desafió a realizar ya que no conocía las diferencias que había entre picc y avr, esto también me permitio adentrarme un poco mas en los conocimientos de la materia.

También me ha dejado ver las diferencias que hay al hacer el proyecto de forma física, ya que en proteus es muy raro encontrar ruido en las señales.

En lo que respecta a mejoras, este proyecto, aun tiene muchas posibilidades de mejorar y crecer. Pueden agregarse mas sensores, como uno de humedad y temperatura para controlar los aires acondicionados y diseñar una interfaz remota que interactue con la casa

Referencias

http://www.coffeebrain.org/wiki/index.php?title=Tutoriales_ATMEGA

https://sites.google.com/site/qeewiki/books/avr-guide

http://microcontroladores-mrelberni.com/lcd-avr/

http://www.electronicwings.com/avr-atmega/interfaces

Github

Datasheet de los sensores

Anexos

MAIN.C

```
#define F_CPU 16000000
#define BAUD 9600
#define LTHRES 500 // limite del dia y la noche

#include <stdlib.h>
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "i2c/i2cmaster.h"
#include "rtc/rtc.h"

void mi_UART_Init( unsigned int);
uint8_t mi_putc(char);
uint8_t mi_getchar(void);
#define getc() mi_getc()
#define putc(x) mi_putc(x)
/********************/
int NOCHE=0;
int Porton=0, Puerta=0;
/********UART********/
void mi_UART_Init( unsigned int ubrr)
{
	UBRR0 = F_CPU/16/ubrr-1;			// Configura baudrate. Ver
en sección UART de datasheet
	UCSR0B = (1<<RXEN0)|(1<<TXEN0);			// Habilita bits TXEN0 y
RXEN0
	UCSR0C = (1<<USBS0)|(3<<UCSZ00);	// USBS0=1 2 bits stop, UCSZxx=3
8 bits
}
uint8_t mi_putc(char c)
{
	while(!(UCSR0A & (1<<UDRE0)) ); // Espera mientras el bit UDRE0=0 (buffer
de transmisión ocupado)
	UDR0 = c;					// Cuando se desocupa, UDR0
puede recibir el nuevo dato c a trasmitir
	return 0;
}

uint8_t mi_getc()
{
	while ( !(UCSR0A & (1<<RXC0)) );//Espera mientras el bit RXC0=0 (recepción
incompleta)
	return UDR0;				//Cuando se completa, se lee UDR0
}
uint8_t enviarCadena(char* cadena){ // Imprime una cadena de caracteres
	while(*cadena !=0x00){
```

```
                putc(*cadena);
                cadena++;
        }
        return 0;
}
/********ADC********/
void adc_init(){
        ADMUX |=(1<<REFS0);//setting the reference of ADC
        ADCSRA |=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
uint16_t adc_read(uint8_t ch)
{
        ADMUX &= 0xf0;
        ADMUX |= ch;
        ADCSRA |= (1<<ADSC);
        while(ADCSRA & (1<<ADSC));
        return (ADC);
}
/********LDR********/
void LDR(){
        uint16_t adc_result;
        adc_result = adc_read(0);
        char buf[20];

        sprintf(buf, "LDR : %d",adc_result);
        enviarCadena(buf);
        if (adc_result < LTHRES){ //Noche
                PORTD |= (1<<PIND3);
                NOCHE=1;
                enviarCadena("\r\nLuces de patio encendidas\r\n");
        }
        else{                                   //DIA
                PORTD &= ~(1<<PIND3);
                NOCHE=0;
                enviarCadena("\r\nLuces de patio apagadas\r\n");
        }
}
/********SERVO********/
void servo()
{
        if (Puerta==1)
        {
                enviarCadena("\r\nPuerta abierta\r\n");
                OCR1B = 420;
                _delay_ms(2000);
                OCR1B = 200 ;
                enviarCadena("\r\nPuerta cerrada\r\n");
                Puerta=0;
        }
        if (Porton==1)
        {
                enviarCadena("\r\nPorton abierto\r\n");
```

```
                OCR1A = 420;
                _delay_ms(2000);
                OCR1A = 200 ;
                enviarCadena("\r\nPorton cerrado\r\n");
                Porton=0;
        }
}
/********SIR********/
ISR (INT0_vect){

        if (NOCHE==1)
        {
                PORTD &=~ (1<<PIND0);
                enviarCadena("\r\nLuz de garage prendida\r\n");
        }
        Porton=1;
        servo();
}

/********MAIN********/
int main(void) {
        char xhorario[16];
        uint8_t aux[16];
        char c,x1,x;
        uint8_t n1,cant=0;
        uint8_t diaAlarma[7]= {0};

        //entrada
        DDRD &=~ (1<<DDD2);//SIR
        PORTD |= (1<<PIND2);
        DDRB &=~ (1<<DDD6);//pulsador
        PORTB |= (1<<PIND6);
        DDRC &=~ (1<<DDC1); //pulsador
        PORTC |=  (1<<PINC1);
        DDRC &=~ (1<<DDC2); //pulsador
        PORTC |=  (1<<PINC2);
        DDRC &=~ (1<<DDC3); //pulsador
        PORTC |=  (1<<PINC3);
        // salida
        DDRD |= (1<<DDD0);//RIEGO
        PORTD &=~ (1<<PIND0);
        DDRD |= (1<<DDD1); //LUZ GARAGE
        PORTD &=~ (1<<PIND1);
        DDRD |= (1<<DDD3); //PATIO
        PORTD &=~ (1<<PIND3);
        DDRD |= (1<<DDD4); //BANIO
        PORTD &=~ (1<<PIND4);
        DDRD |= (1<<DDD5); //PIEZA
        PORTD &=~ (1<<PIND5);

        //init uart
        mi_UART_Init(BAUD);
        //ADC
```

```
        adc_init();
        //INTERRUPCIONES
        EICRA |= _BV(ISC11);
        EIMSK |= _BV(INT0) | _BV(INT1);
        sei();

        //SERVO
        DDRB |= (1<<PINB1) | (1<<PINB2);
        TCCR1A |= (1 << WGM11) | (1 << COM1A1) | (1<<COM1B1);
        TCCR1B |= (1 << WGM12) | (1 << WGM13) |(1<<CS10)|(1<<CS11);
        ICR1 = 4999;
        OCR1A = 200 ;
        OCR1B = 200 ;

        //init ds1307
        ds1307_init();
        sei();
        char buf[50];
        char* days[7]= {"Domingo","Lunes","Martes","Miercoles","Jueves","Viernes",
"Sabado"};

        uint8_t anio = 0;
        uint8_t mes = 0;
        uint8_t dia = 0;
        uint8_t hora = 0, horaAlarma1=0, horaAlarma2=0;
        uint8_t minuto = 0, minutoAlarma1=0, minutoAlarma2=0;
        uint8_t segundo = 0;
        uint8_t sem=0;

        _delay_ms(10);
        enviarCadena("Desea configurar la hora?\r\n");
        _delay_ms(30);
        enviarCadena(" 1. SI - 2. NO \r\n");

        x= mi_getc();

        switch(x){
               case '1':
               enviarCadena("\nIngrese fecha con el siguiente formato\r\n");
               enviarCadena("dd.mm.aa\r\n");
               _delay_ms(30);

               for (int i=0;i<8;i++)
               {
                      xhorario[i]=mi_getc();
               }
               for (int i=0;i<8;i++)
               {
                      putc(xhorario[i]);
               }
               for (int i=0;i<8;i++)
               {
```

```
                aux[i]=xhorario[i]-'0';
        }
        dia=aux[0]*10+aux[1];
        mes=aux[3]*10+aux[4];
        anio=aux[6]*10+aux[7];

        enviarCadena("\nIngrese hora con el siguiente formato\r\n");
        enviarCadena("hh.mm.ss\r\n");
        _delay_ms(30);

        for (int i=8;i<16;i++)
        {
                xhorario[i]=mi_getc();
        }
        for (int i=8;i<16;i++)
        {
                putc(xhorario[i]);
        }
        for (int i=8;i<16;i++)
        {
                aux[i]=xhorario[i]-'0';
        }
        hora=aux[8]*10+aux[9];
        minuto=aux[11]*10+aux[12];
        segundo=aux[14]*10+aux[15];

        ds1307_setdate(anio, mes, dia, hora, minuto, segundo);

        break;
        case '2':
        break;
        default:
        enviarCadena("valor invalido\r\n");
        break;
}

_delay_ms(10);
enviarCadena("\nDesea configurar horario del sistema de riego?\r\n");
_delay_ms(30);
enviarCadena(" 1. SI - 2. NO \r\n");
x= mi_getc();

switch(x){
        case '1':
        enviarCadena("\nIngrese cantidad de veces a la semana\r\n");
        enviarCadena("Si desea todos los dias ingrese 7\r\n");

        c=mi_getc();
        cant= c - '0';

        enviarCadena("Ingrese\r\n");
```

```
		enviarCadena("0. Domingo\r\n");
		enviarCadena("1. Lunes\r\n");
		enviarCadena("2. Martes\r\n");
		enviarCadena("3. Miercoles\r\n");
		enviarCadena("4. Jueves\r\n");
		enviarCadena("5. Viernes\r\n");
		enviarCadena("6. Sabado\r\n");
		if (cant==7)
		{
			for (int i=0; i<cant; i++)
			{
				diaAlarma[i]=i;
			}
		}
		else
		{
			for (int i=0; i<cant; i++)
			{
				x1=getc();
				putc(x1);
				n1= x1 - '0';
				diaAlarma[i]=n1;
			}
		}

		enviarCadena("\nIngrese hora de inicio con el siguiente formato\r\n
");
		enviarCadena("hh.mm\r\n");
		_delay_ms(30);

		for (int i=8;i<13;i++)
		{
			xhorario[i]=mi_getc();
		}
		for (int i=8;i<13;i++)
		{
			putc(xhorario[i]);
		}
		for (int i=8;i<13;i++)
		{
			aux[i]=xhorario[i]-'0';
		}
		horaAlarma1=aux[8]*10+aux[9];
		minutoAlarma1=aux[11]*10+aux[12];
		segundo=0;

		enviarCadena("\nIngrese hora de fin con el siguiente formato\r\n");
		enviarCadena("hh.mm\r\n");
		_delay_ms(30);
```

```
                for (int i=8;i<13;i++)
                {
                        xhorario[i]=mi_getc();
                }
                for (int i=8;i<13;i++)
                {
                        putc(xhorario[i]);
                }
                for (int i=8;i<13;i++)
                {
                        aux[i]=xhorario[i]-'0';
                }
                horaAlarma2=aux[8]*10+aux[9];
                minutoAlarma2=aux[11]*10+aux[12];
                segundo=0;

                break;
                case '2':
                break;
                default:
                enviarCadena("valor invalido\r\n");
                break;
        }

        while(1) {


                ds1307_getdate(&sem, &anio, &mes, &dia, &hora, &minuto, &segundo);

                sprintf(buf, "%3s  %d/%d/20%d %d:%d:%d",days[sem],dia, mes,  anio,
hora, minuto, segundo);
                enviarCadena(buf);
                enviarCadena("\r\n");
                for (int i=0; i<cant; i++)
                {
                        if (diaAlarma[i]==sem)
                        {
                                if ((hora>= horaAlarma1 && minuto>=minutoAlarma1) && (
hora <=horaAlarma2 && minuto <= minutoAlarma2))
                                {
                                        enviarCadena("\r\nRiego encendido\r\n");
                                        PORTD |= (1<<PIND0);

                                }else{
                                        PORTD &=~(1<<PIND0);
                                }
                        }
                }

                _delay_ms(1000);
```

```
                LDR();
                if (bit_is_clear(PIND,PIND6)) // se apreto el boton?
                {
                        _delay_ms(20);

                        if (bit_is_clear(PIND,PIND6)){

                                Puerta=1;
                                servo();
                                if (NOCHE==1)
                                {
                                        enviarCadena("Luz de garage apagada \n");
                                        PORTD = (1<<PIND3);
                                        _delay_ms(1000);
                                }
                                while(bit_is_clear(PIND,PIND6)); // espera a que el
ususario deje de pulsar
                        }
                }

                enviarCadena("----------------------------------------");

        }
}
```

RTC.H

```
#ifndef INCFILE1_H_
#define INCFILE1_H_

#define DS1307_ADDR (0x68<<1) //direccion del sensor
#define DS1307_I2CFLEURYPATH "../i2c/i2cmaster.h" //define the path to i2c fleury
lib
#define DS1307_I2CINIT 1 //init i2c

extern void ds1307_init();
extern uint8_t ds1307_setdate(uint8_t anio, uint8_t mes, uint8_t dia, uint8_t
hora, uint8_t minuto, uint8_t segundo);
extern void ds1307_getdate(uint8_t *dsem, uint8_t *anio, uint8_t *mes, uint8_t *
dia, uint8_t *hora, uint8_t *minuto, uint8_t *segundo);

#endif /* INCFILE1_H_ */
```

RTC.C

```
#define F_CPU 16000000
#include "avr/io.h"
#include "avr/pgmspace.h"
#include "util/delay.h"


#include "rtc.h"
#include DS1307_I2CFLEURYPATH

//lo guardo en la memoria Flash con progmem
const uint8_t ds1307_daysinmonth [] PROGMEM = { 31,28,31,30,31,30,31,31,30,31,30,
31 };

void ds1307_init() {
	#if DS1307_I2CINIT == 1
	i2c_init();
	_delay_us(10);
	#endif
}

//cambiar valores

uint8_t ds1307_dec2bcd(uint8_t val) {
	return val + 6 * (val / 10);
}
static uint8_t ds1307_bcd2dec(uint8_t val) {
	return val - 6 * (val >> 4);
}

//numero de dias del anio
static uint16_t ds1307_date2days(uint8_t y, uint8_t m, uint8_t d) {
	uint16_t days = d;
	for (uint8_t i = 1; i < m; ++i)
	//lee un byte de la memoria flash
	//entre parentesis es la direccion donde indice donde reside la variable
```

```
        days += pgm_read_byte(ds1307_daysinmonth + i - 1);
        if (m > 2 && y % 4 == 0)
        ++days;

        return days + 365 * y + (y + 3) / 4 - 1;
}
// dia de la semana

uint8_t ds1307_getdayofweek(uint8_t y, uint8_t m, uint8_t d) {
        uint16_t day = ds1307_date2days(y, m, d);
        return (day + 6) % 7;
}
// configurar fecha
uint8_t ds1307_setdate(uint8_t year, uint8_t month, uint8_t day, uint8_t hour,
uint8_t minute, uint8_t second) {

        if (second < 0 || second > 59 ||
        minute < 0 || minute > 59 ||
        hour < 0 || hour > 23 ||
        day < 1 || day > 31 ||
        month < 1 || month > 12 ||
        year < 0 || year > 99)
        return 8;

        if (month== 2 && year % 4 == 0 && day==29 )
        {
                //dia de la semana
                uint8_t dayofweek = ds1307_getdayofweek(year, month, day);

                //escribir fecha
                i2c_start_wait(DS1307_ADDR | I2C_WRITE);
                i2c_write(0x00);
                i2c_write(ds1307_dec2bcd(second));
                i2c_write(ds1307_dec2bcd(minute));
                i2c_write(ds1307_dec2bcd(hour));
                i2c_write(ds1307_dec2bcd(dayofweek));
                i2c_write(ds1307_dec2bcd(29));
                i2c_write(ds1307_dec2bcd(month));
                i2c_write(ds1307_dec2bcd(year));
                i2c_write(0x00);
                i2c_stop();
        }
        else
        {
                //dia basado en el mes
                if(day > pgm_read_byte(ds1307_daysinmonth + month - 1))
                return 0;

                //obtener dia de la semana
                uint8_t dayofweek = ds1307_getdayofweek(year, month, day);

                //escribir fecha
                i2c_start_wait(DS1307_ADDR | I2C_WRITE);
```

```
                i2c_write(0x00);
                i2c_write(ds1307_dec2bcd(second));
                i2c_write(ds1307_dec2bcd(minute));
                i2c_write(ds1307_dec2bcd(hour));
                i2c_write(ds1307_dec2bcd(dayofweek));
                i2c_write(ds1307_dec2bcd(day));
                i2c_write(ds1307_dec2bcd(month));
                i2c_write(ds1307_dec2bcd(year));
                i2c_write(0x00);
                i2c_stop();
        }
        return 1;
}
//obtener fecha
void ds1307_getdate(uint8_t *dsem,uint8_t *year, uint8_t *month, uint8_t *day,
uint8_t *hour, uint8_t *minute, uint8_t *second) {
        i2c_start_wait(DS1307_ADDR | I2C_WRITE);
        i2c_write(0x00);//stop oscillator
        i2c_stop();

        i2c_rep_start(DS1307_ADDR | I2C_READ);
        *second = ds1307_bcd2dec(i2c_readAck() & 0x7F);
        *minute = ds1307_bcd2dec(i2c_readAck());
        *hour = ds1307_bcd2dec(i2c_readAck());
        *dsem = ds1307_bcd2dec(i2c_readAck());
        *day = ds1307_bcd2dec(i2c_readAck());
        *month = ds1307_bcd2dec(i2c_readAck());
        *year = ds1307_bcd2dec(i2c_readNak());
        i2c_stop();
}
```

I2CMASTER.H

```
#define I2C_READ    1
#define I2C_WRITE   0


extern void i2c_init(void);
extern void i2c_stop(void);
extern unsigned char i2c_start(unsigned char addr);
extern unsigned char i2c_rep_start(unsigned char addr);
extern void i2c_start_wait(unsigned char addr);
extern unsigned char i2c_write(unsigned char data);
extern unsigned char i2c_readAck(void);
extern unsigned char i2c_readNak(void);
extern unsigned char i2c_read(unsigned char ack);
#define i2c_read(ack)  (ack) ? i2c_readAck() : i2c_readNak();
```

```
#endif
```

I2CMASTER.C

```
#include <inttypes.h>
#include <compat/twi.h>
#include "i2cmaster.h"
#ifndef F_CPU
#define F_CPU 4000000UL
#endif
#define SCL_CLOCK  10000L

void i2c_init(void)
{
	TWSR = 0;
	TWBR = ((F_CPU/SCL_CLOCK)-16)/2;

}
unsigned char i2c_start(unsigned char address)
```

```
{
	uint8_t   twst;
	TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
	while(!(TWCR & (1<<TWINT)));
	twst = TW_STATUS & 0xF8;
	if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

	// enviando direccion
	TWDR = address;
	TWCR = (1<<TWINT) | (1<<TWEN);

	// esperar a que se complete a transmicion y devolver un nack o un ack
	while(!(TWCR & (1<<TWINT)));

	//chequea valor del estatos de twi
	twst = TW_STATUS & 0xF8;
	if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

	return 0;

}
void i2c_start_wait(unsigned char address)
{
	uint8_t   twst;


	while ( 1 )
	{
		TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

		while(!(TWCR & (1<<TWINT)));
		twst = TW_STATUS & 0xF8;
		if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

		TWDR = address;
		TWCR = (1<<TWINT) | (1<<TWEN);

		while(!(TWCR & (1<<TWINT)));

		twst = TW_STATUS & 0xF8;
		if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
		{
			TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

			while(TWCR & (1<<TWSTO));

			continue;
		}
		break;
	}

}
unsigned char i2c_rep_start(unsigned char address)
```

```
{
	return i2c_start( address );
}
void i2c_stop(void)
{
	/* send stop condition */
	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

	// wait until stop condition is executed and bus released
	while(TWCR & (1<<TWSTO));

}
unsigned char i2c_write( unsigned char data )
{
	uint8_t   twst;

	// envia datos al sensor
	TWDR = data;
	TWCR = (1<<TWINT) | (1<<TWEN);

	// espera
	while(!(TWCR & (1<<TWINT)));

	twst = TW_STATUS & 0xF8;
	if( twst != TW_MT_DATA_ACK) return 1;
	return 0;

	}
unsigned char i2c_readAck(void)
{
	TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
	while(!(TWCR & (1<<TWINT)));

	return TWDR;

	}
unsigned char i2c_readNak(void)
{
	TWCR = (1<<TWINT) | (1<<TWEN);
	while(!(TWCR & (1<<TWINT)));

    return TWDR;

}
```