ELECTRONIC SYSTEM PROJECT TO CONTROL A HOSE REELS BY DGPS.

1. Memory

Albert Mach Casellas

Electronic Engineer

Can siset nº4, Ravós del Terri 17845

GIRONA (Spain)

E-mail: magnum@girona.com

Mob: +34630912488 / +48787995286

INDEX

1 Introduction

Throughout the world, irrigation (water for agriculture, or growing crops) is probably the most important use of water (except for drinking and washing). Almost 60 percent of all world's freshwater withdrawals go towards irrigation uses. Large-scale farming could not provide food for the world's large populations without the irrigation of crop fields by water obtained from rivers, lakes, reservoirs, and wells.

The water is a main factor limited for agricultural production in arid lands. So, the management of water resources in irrigation is a fundamental aspect for their sustainability. Often, for correct management it is necessary the use of several tools; sensors, actuators, integrate electronic systems...

There are many types of irrigation systems. But, most farmers have limited choices for their particular farm or field. Some systems are inherently more water and energy efficient while others are designed to overcome limitations such as irregular field shapes, sloping land, or limited water supplies.

The project purpose has been to design an electronic system to control a hose reel (Fig 1) for irrigating large, rectangular or regular shaped fields...

Sprinklers (6) may also be mounted on moving platforms connected to the water source (2) by a hose. Automatically moving wheeled systems known as traveling sprinklers may irrigate areas such as farms, sports fields, parks, pastures, and cemeteries unattended.

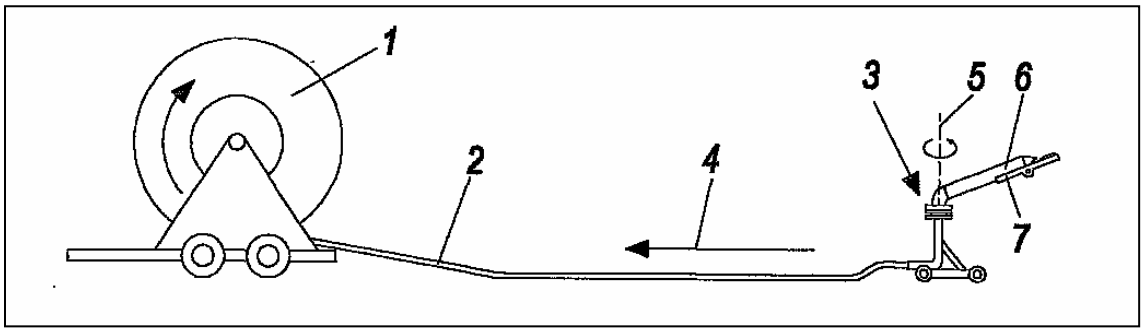


Fig 1. Main view Hosereel.

This traveler sprinkler on one hand lets to optimize the water resource because it only irrigates the interesting places and it doesn't irrigate the zone that you don't want; nearby landscape, neighbors house, roads... On the other hand, it is considerably reduced the fieldwork and it isn't necessary a qualified personal to control the sprinkler when this is working.

The operation of the traveler sprinkler is not too complicated. It's composed for two different parts; the hose is the fixed part in the extreme of the field (1) where the progress speed is controlled. The other part is a big volume gun (3). These guns have a circular movement (Fig 2), however when it has progressed a few meters, the irrigation area is a rectangle.

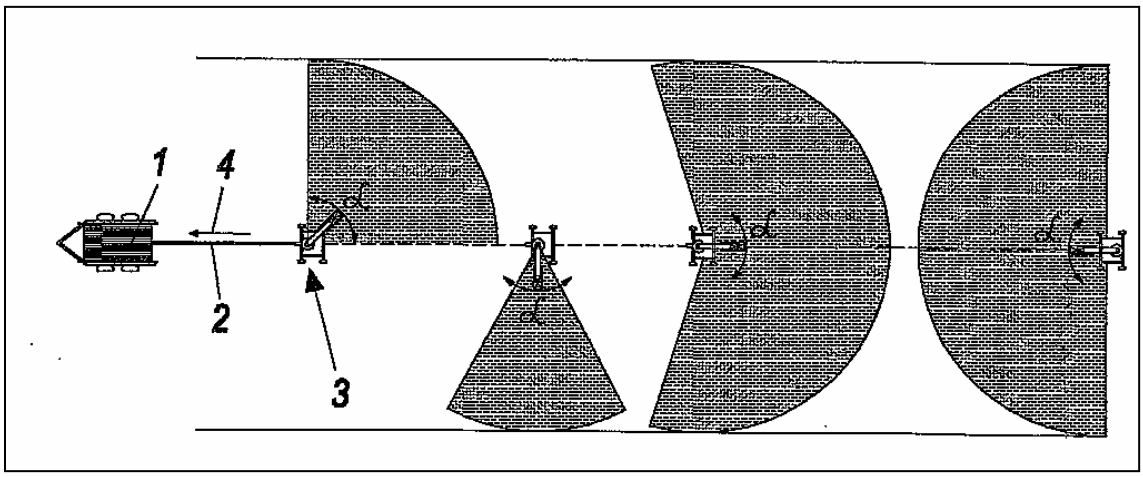


Fig 2. Bird's-eye view hosereel.

2 Electronic system in the sprinkler

2.1 Block diagram

The first block diagram is a electronic system used in the splinker (0). Inside the PCB there is a microcontroller. A computer-on-a-chip to do all arithmetic and logic elements of a general purpose and also integrates additional elements such as read-only and read-write memory, and input/output interfaces. Also, there is a RF transceiver to communicate with PCB_2 in the hose. The GPRS is a compact electronic system that makes possible the comunication between the sprinkler and the laptop, it also enables as to get the GPS coordenates. Finally, the inputs and outputs interface are used to adapt the diferents sensors and actuators with a microcontroller.

Also, it is used an analogical pressure sensor to determinate a width of the irrigated zone, an angle encoder to know which his exactly position and inductor sensor to know the initial point. The actuator that incorporates is an electrovalve to turn the direction of splinker and a servomotor to control the width of the irrigated zone. The power supply is generated by solar cells making the gun independent of external supply sources. A storage battery supports the operation of the gun at night and on cloudy days.

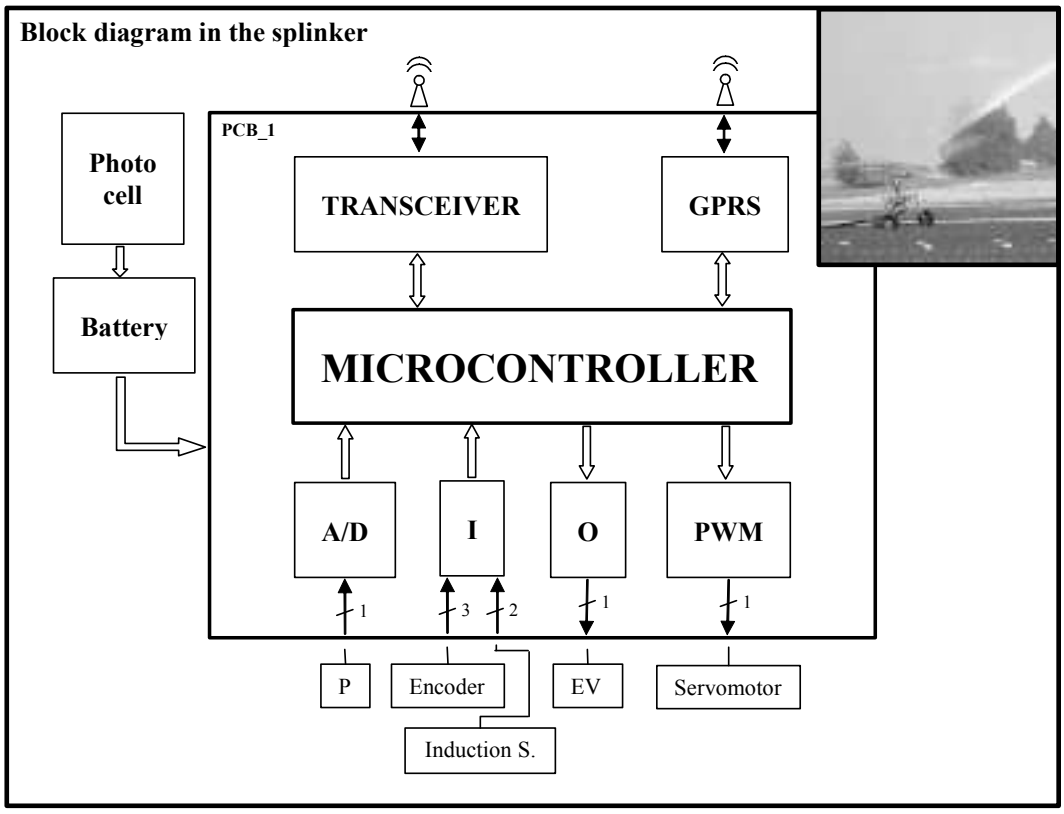


Fig 3. Block diagram in the traveller sprinkler

2.2 Description of the electronic components

2.2.1 Microcontroller PIC16F877

The necessity to use a microcontroller to develop this application has been essential to choose anyone that offer the system provisions require. The big demand that there are the last years with a good technique characteristics how; low cost, low power, quality, reliability, efficiency and a lot of information, the PIC microcontrollers (Peripheral Interface Controller) are an excellent device to use in many applications.

The architecture of the PIC 16FXXX family is a Harvard type with a 8 bits dates bus and a 14 bits instructions bus. It has a CPU the RISC type (Reduce instruction Set Computer) where only use 35 instructions. The chosen microcontroller is PIC 16F877, the mid-range in the Microchip's house. The main characteristics of this microcontroller are the next (Table 1).

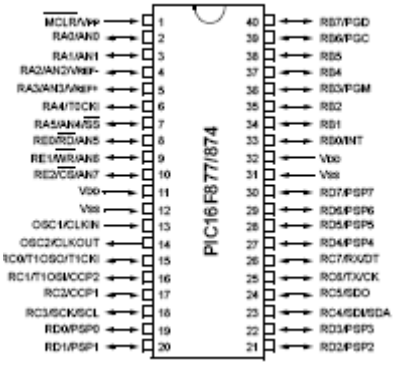
Features	
High performance RISC CPU	
Operating speed: DC-20MHZ clock input	
Up to 8K x 14 words of FLASH Program Memory Up to 368 x 8 bytes of Data Memory (RAM) Up to 256 x 8 bytes of EEPROM Data Memory	
Interrupt capability (up to 14 sources)	
Programmable code protection	
Power saving SLEEP mode	
In-Circuit Serial Programming (ICSP)	
Wide operating voltage range: 2 to 5.5V	
High source current: 25mA	
Low power consumption	
Three timers: 8 and 16 bit timer/counter with prescaler	
Two capture, compare, PWM modules	
Ten bit multi channel analog to digital converter	
Synchronous Serial Port (SSP) with SPI and I2C	
Universal Synchronous Asynchronous Receiver-Transmitter (USART/SCI) with 9 bit address detection	
Parallel Slave Port (PSP), RD, WR and CS control	

Table 1. PIC 16F877 features

2.2.2 Reference voltage REF02

The REF02 precision voltage reference provides a stable 5V output that can be adjusted over a ±6% range with minimal effect on temperature stability. Low cost, low noise and low power make the REF02 an excellent choice whenever a stable voltage reference is required (Table 2).

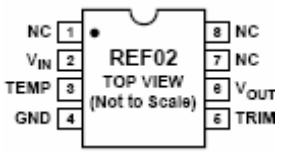
Features	
5 V output: ±0.3% maximum	
Temperature voltage output: 1.96mV/ºC	
Excellent temperature stability: 8.5 ppm/ºC maximum	
Low noise: 15 uV p-p maximum	
Low supply current: 1.4 mA maximum	
Wide input voltage range: 7V to 40V	
High load-driving capability: 10mA	
No external components	
High source current: 25mA	
Low power consumption	
Short-circuit proof	

Table 2. REF02 features

2.2.3 Driver ICL7667

The ICL7667 is a dual monolithic power MOSFET driver designed to translate TTL inputs to high voltage/current outputs. Its low delay and transition times make it ideal to drive power MOSFETs for switching power supplies, motor controllers, and DC-DC converters.

The ICL7667's high speed minimizes power losses in switching power supplies and DC-DC converters due to rapid charging/discharging of the gate capacitance of the power MOSFETs. The ICL7667 inputs are TTL compatible, enabling direct interface to common switched mode power supply controllers (Table 3).

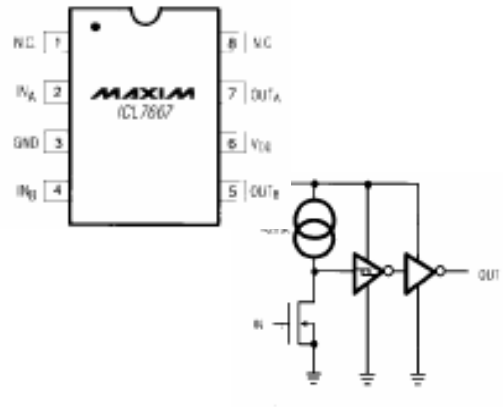
Features	
Fast rise and fall times: 20ns with 1000 pF load.	
Wide supply range: 4.5 to 17V	
Low power consumption: 6mW with inputs low and 120 mW with inputs High.	
TTL/CMOS input compatible	
Package dissipation plastic dip: 300 mW	
Storage temperature: -55 to 160ºC	
Low Rout = 4Ω	
Low Rout = 4Ω	

Table 3. ICL7667 features

2.2.4 Mosfet IRF540

These are N-Channel enhancement mode silicon gate power field effect transistors. Thet are advanced power MOSFETs designed, tested, and guaranteed to withstand a specified level of energy in the breakdown avalanche mode of operation.

All of theses power MOSFETs are designed for applications such as switching regulators, switching converters, motor drivers, relay drivers, an drivers for high poser bipolar switching transistors requiring high speed and low gate drive power. These types can be operated directly from integrated circuits (Table 4).

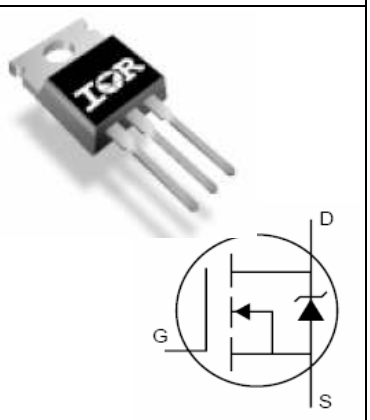
Features	
Continuous drain current: 25A	
Drain to gate voltage: 80V	
Single pulse avalanche energy rated	
Nanosecond switching speeds	
Linear transfer characteristics	
High input impedance	
Operating and storage temperature: -55 to 175ºC	
Maximum power dissipation: 150 W	
Drain to source on resistance: 80Mω	

Table 4. ICL7667 features

2.2.5 Diode BYW29-200

This state-of-the-art device is designed for use in switching power supplies, inverters and as free wheeling diodes (Table 5).

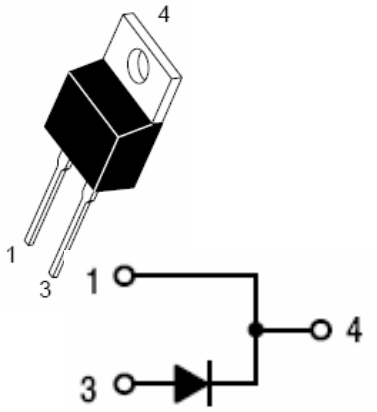
Features	
Working peak reverse voltage:200 V	
Average rectified current 8 A	
Peak repetitive forward current: 16 A	
Non repetitive peak surge current 100 A	
Low forward voltage	
Low leakage current	
High temperature glass passivated junction	
Operating and storage temperature: -55 to 175ºC	
Maximum power dissipation: 10 W	

Table 5. Diode BYW29-200

2.3 Input connectors.

2.3.1 Connector J1. Pressure sensor.

The connector J1 has four pins; one wire for power supply, two wire more for I2C interface. It has a Serial Clock Line (SCL) input and serial digital output data line and finally, one wire for GND connection. The output of the device is a corrected pressures value in a hexadecimal format with 12-bit resolution. It uses to know the absolute pressure in the sprinkler and to determinate the width irrigation (Table 6).

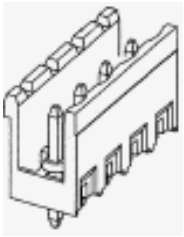
From connector J1	To Pic 16F877	Connector PCB
Pin_1. VCC	-	
Pin_2. SCL	Pin_34. RB1/SCL	
Pin_3. SCI	Pin_35. RB2/SDA	
Pin_4. GND	-	

Table 6. Pressure sensor connection

2.3.2 Connector J2. Encoder sensor.

The connector J2 has five pins; one wire for power supply, one wire more to know a referent point in the complete turn, two wires are for channels A and B and to know the position. Finally, also one wire for GND connection. It uses to know the exactly position of the sprinkler and to determinate how many degrees it is turning (Table 7).

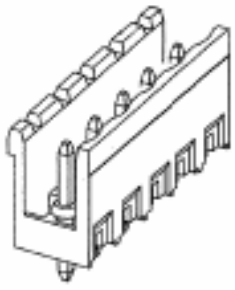
From connector J2	To Pic 16F877	Connector PCB
Pin_1. VCC	-	
Pin_2. Ref. point	Pin_36. RB3	
Pin_3. Channel A	Pin_37. RB4	
Pin_4. Channel B	Pin_38. RB5	
Pin_5. GND	-	

Table 7. Encoder sensor connection

2.3.3 Connector J3. Inductive sensor.

It has reserved two pins (pin_2 and pin_3) for NPN inductive sensors. These sensors are used to know an initial position of the sprinkler. Before it irrigates a determinate zone, the sprinkler will find between two sensors separated 30º and when the sprinkler cross any inductive sensor, it can know an initial position (Table 8).

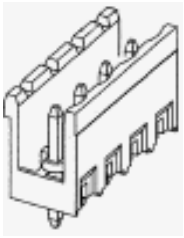
From connector J3	To Pic 16F877	Connector PCB
Pin_1. VCC	-	
Pin_2. Inductive S1	Pin_6. RA4	
Pin_3. Inductive S2	Pin_7. RA5	
Pin_4. GND	-	

Table 8. Inductive sensor connection

2.4 Output connectors.

2.4.1 Connector J4. Electrovalves.

The connector J4 uses to connect the electrovalves that let to change the splinker direction. There is commune pin connector (pin 3) and the pin one connector lets to control a turn right and pin two connector lets to control a left turn. Each turn is controlled for a different valve (Table 9).

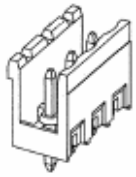
From Pic 16F877	To connector J4	Connector PCB
Pin_19. RD0	Pin_1. Electrovalve_R	
Pin_20. RD1	Pin_2. Electrovalve_L	
-	Pin_3. GND	

Table 9. Electrovalves connection

2.4.2 Connector J5. Servo motor.

The servomotor needs a constant power supply and a GND connection. The Pin 2 is used to PWM control. The main objective of the servo motor is controlled the irrigation distance. It is possible to get a difference irrigation distance changing the pulse width modulation (Table 10).

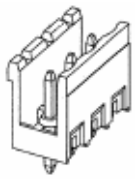
From Pic 16F877	To connector J4	Connector PCB
-	Pin_1. VCC	
Pin_17. RC2/PWM1	Pin_2. Servo motor	
-	Pin_3. GND	

Table 10. Servo motor connection

2.4.3 Connector J6. GPRS.

The communication between PIC 16F877 and GPRS modem by RS-232 protocol. It needs the one pin of power supply, other for GND connection and two pins more to do communication (Rx and Tx, respectability). The PIC 16F877 microcontroller has a hardware integrated USART and let to work with the asynchronous (full duplex) mode.

It uses to obtain a GPS coordinates and to know what is the actually position (Table 11).

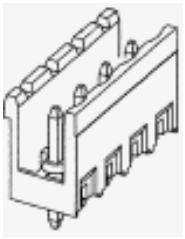
From Pic 16F877	To connector J4	Connector PCB
-	Pin_1. VDD	
Pin_25. RC6	Pin_2. GPRS_RX	
Pin_26. RC7	Pin_3. GPRS_TX	
-	Pin_4. GND	

Table 11. GPRS connection

2.4.4 Connector J7. Programmer connector.

One of the possibilities to program a microcontroller is by using a connector. There are only a 2x5 connector between microcontroller and the other parts of the board. Once it plugs in the programmer connector, it will be able to program PIC in system.

Once the development of a device is finished, the jumpers have to be restored for enabling the device to work without programmer. These jumpers establish connections from MCLR, RB6 and RB7 to peripherals on the board (Table 12).

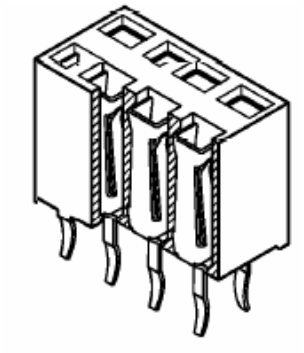
From connector J1	Connector PCB
Pin_1. VCC	
Pin_2. PIC_VCC	
Pin_3. TX/RB6	
Pin_4. No connect	
Pin_4. RX/RB/	
Pin_6. No connect	
Pin_7. MCLR/VPP	
Pin_8. VCC	
Pin_9. GND	
Pin_10. GND	

Table 12. Programmer connection

2.5 Sensors technical characteristics

2.5.1 Microstructure pressure sensor

The ASDX DO series sensors provide a very cost effective solution for pressure applications that require small size plus performance. Device is available to measure absolute pressure up to 100psi (SDX DO100). The general specifications are in the next table (Table 13).

GENERAL SPECIFICATIONS	
Supply Voltage (Vs)	4.75 Vdc to 5.25 Vdc
Maximum Supply Voltage*	6.50 Vdc max.
Current Consumption	6 mA typ.
Output current - sink	2 mA max.
Output current - source	2 mA max.
Lead Soldering Temperature	2 Sec to 4 Sec @ 250 °C [482 °F]

The ASDX DO device is in a standard DIP package and provides digital correction of sensor temperature coefficients and non-linearity. It use I2C compatible protocol, which allows easy interfacing to most commonly used microcontroller without additional components or electronic circuitry (Table 14).

Features	
Available in absolute	
Calibrate and temperature compensated output	
Pressure ranger 0 psi to 100 psi	
Response time 8 ms	
Standard DIP package	
ASIC enhanced Output	
I2C compatible protocol	

Table 14. Pressure sensor features

2.5.2 Inductive proximity sensor

The proximity sensor is specially adapted and o for use with drives. The sensor boxes feature sturdy construction for use in rough environments and can withstand corrosive and dusty environments.

Cylindrical type, 4 mm diameter, metal case and short DC supply (Table 15).

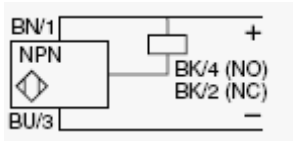
Features		
Nominal sensing distance	1 mm	 
3 wire	NPN	
Connection mode	By cable	
Temperature	-40...+85ºC	
Degree protect	IP67	
Current consumption, no-load	<=10Ma	
Rated supply voltage	5...24Vdc	
Maximum switching frequency	5000 Hz	
Case	Stainless still case	

Table 15. Inductive proximity sensor features

2.5.3 Modular Magnetic Encoder

The robust ERM modular magnetic encoder is especially suited for use in production machines. Their large possible inside diameters as well as the small dimensions and compact design of the scanning head (Fig 4).

Fig 4. Modular Magnetic Encoder

With the incremental measuring method, the graduation consists of a periodic grating structure. The position information is obtained by counting the individual increments (measuring steps) from some point of origin (Table 16).

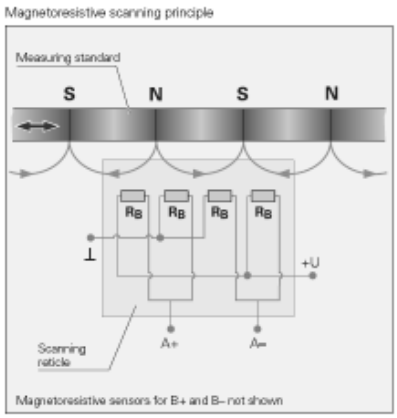
Features ERM 220		
Incremental signals	ERM 220 TTL	
Reference mark	One	
Power supply	5v	
Power consumption	≤ 150 mA	
Drum inside diameter	80 mm	
Line count	1024	
Shaft speed	≤13000 rpm	
Protection	IP 67	
Range operating temperature	-10...100ºC	

Table 16. Modular Magnetic Encoder features

The incremental signals are transmitted as the square-wave pulse trains Ua1 and Ua2, phase-shifted by 90º elec. The reference mark signal consists for one or more reference pulses Ua0, which are gated with the incremental signals (Table 17).

The distance between two successive edges of the incremental signals Ua1 and U a2 through 1 fold, 2 fold, or 4 fold evaluations is one measuring step.

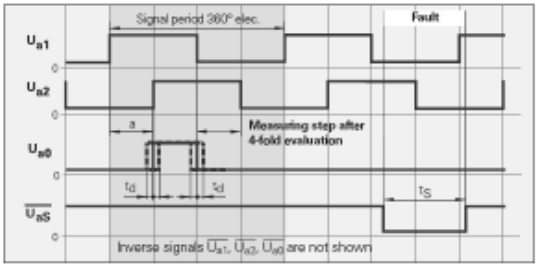
Interfaces incremental signals TTL		
Incremental signals	2TTL square-wave signals	
Fault detection signal	Ts ≥ 20 ms, HIGH	
Signal level	$U_H \geq 2.5V$ at $-I_H=20$ mA $U_L \leq 0.5V$ at $I_L=20$ mA	
Permissible load	Zo ≥ 100Ω $C_{load} \leq 1000$ pF	
Switching times	T ≤ 30 ns (10 ns)	

Table 17. Signals TTL

2.6 Actuators technical characteristics

2.6.1 Electrovalve 2 way

Alcon solenoid valves are chosen for this application. It has a compact valve design, normally closed and is specially adapted and o for use with drives (Table 18).

Features 21 series		
Standard body material	Brass	
Coil Voltage DC	12V	
Standard protection	IP67	
Range operating temperature	-10...50ºC	
Electrical connection	DIN 43650	
Pipe size	¼”	
Orifice	1.6 mm	
Max. Pressure	60 bar	
Power	14.5 W	

Table 18. Electrovalve specifications

2.6.2 Servomotor

A precision electro-mechanical servo for radio control applications with requirements exceeding those servos (Table 19).

Ruggedly designed and built for hostile environments and performances critical applications.

Features HS-5475HB STANDARD DIGITAL SERVO		
Control system	Pulse width control	
Voltage DC	4.8V to 6.0V	
Operating speed	0.23sec/60º	
Stall Torque	4.4 kg.cm	
Idle. Current	3mA	
Stall current	900 mA	
Dead band width	1 us	
Motor type	Cored metal brush	
Direction	Pulse traveling 1500 to 1900 us	

Table 19. Servo motor features

3 System description

For this application we suggest to use a microcontroller PIC18F2545 how central unity. The Pic microncontrellers are an escellent device to use in many applications; low cost, low poer, quality, reliability, efficiency… However, the microcontroller offers much recourse that we didn't use. Principally, we chose it for the possible amplifications in the future how can be the external EEPROM, CAN communication,

3.1 Main flowchart

Flowcharting is an alternative method for documenting the logic of a solution algorithm. Flowcharting uses graphics symbols rather than text to describe the logic of the solution. The main state is the first one and it is in there when the sprinkler does not work or does not irrigate (Fig 5). Immediately, when the sprinkler begins to move, by interrupt (Int_ind), keep the first position (Pos_ini). When the next capture is different that the last, it jumps to the next case and it keeps the end position (Pos_end). If the position doesn't change in few minutes, it returns to sleep mode.

Another possibility to leave the sleep mode is by interrupt of the encoder (Int_enc). When the encoder move, turn right or turn left, the microcontroller is woken up. Then try to find a point 0 of encoder or two inductive sensors (Int_ind1 or Int_ind2) that indicate where is it.

By GPS it knows what is the position of the sprinkler, with the encoder know what is the angle, with the pressure and the diameter of nozzle know what is the long of the water. So, it can know what is the further position where the water falls. Then the program decide if it's inside the field or not and change or not the way. Before to change the way, the servo motor put a piece of iron in front of the nozzle and the long of water became shorter. When it is shortest, it is change the way and began to do the contrary, take out the piece of iron. There are three parameters related with the distance of the water; pressure, diameter of nozzle and the distance the piece of iron is in the middle of the source.

At the beginning, when the sprinkler is at the end of the field it is impossible irrigate with full angle. So, it is necessary irrigate in two parts; first in one band and after "x" time, irrigate other band and change the band continually. After few minutes, the sprinkler move and the distance between sprinkler and end of the field grow. When this distance is bigger than the long of water, the sprinkler can irrigate full area.

Normally, the sprinkler irrigates the opposite band that advance and the most habitual is irrigation more than 180º. So, the user must choose which values of angles want that the sprinkler change the way (B1 and B2).

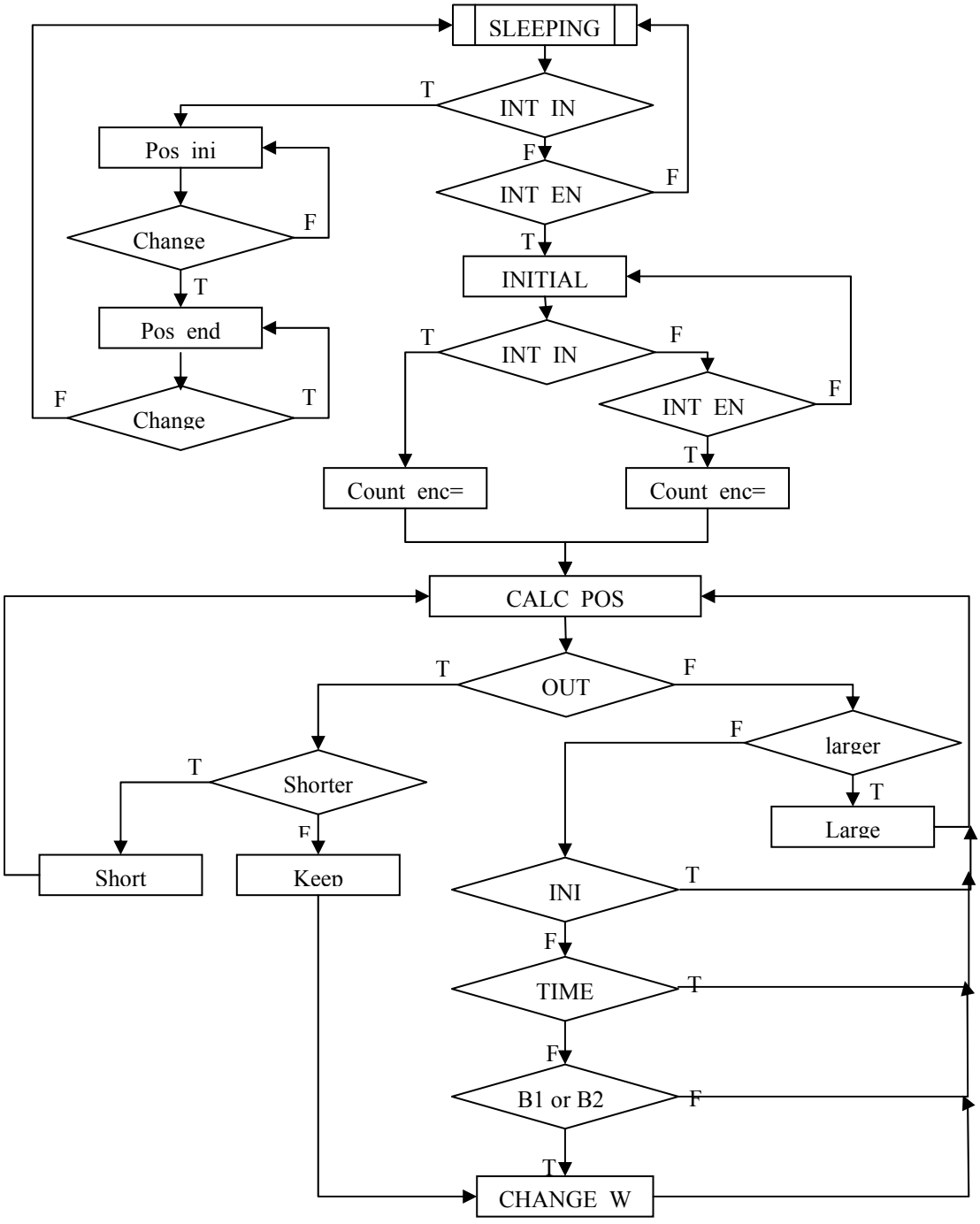


Fig 5. Main flow chart.

3.2 Flow chart to calculate the position

This flow chart decides what is the further point where the water falls (point P in the figure). First, it receives the point Pg from GPS (this point is referenced to {R} reference coordinates). It knows the irrigation distance with diameter nozzle, pressure irrigation and if there are piece of iron in the middle of the source or not.

The next step is calculated the position P respect {S} reference coordinates system. Finally, like this, it is calculated the position "P" respect {R} reference coordinates system (Fig 6).

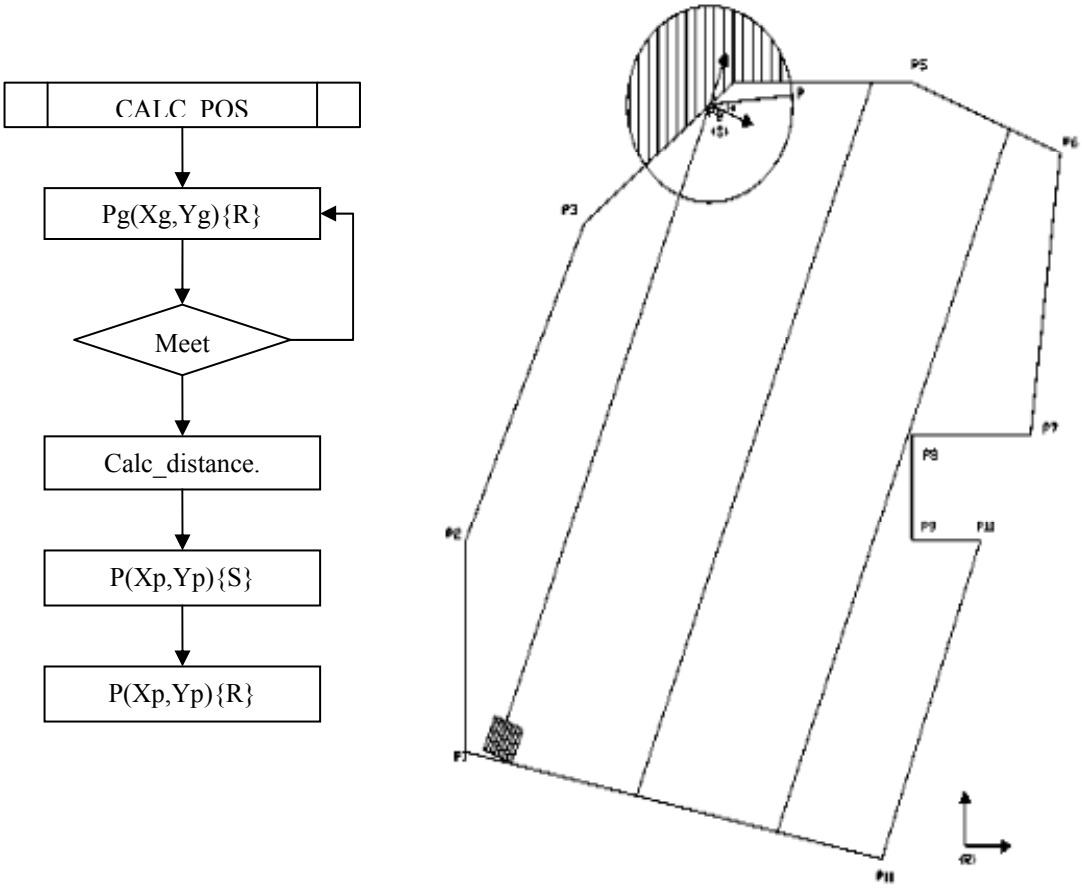


Fig 6. Flow chart to calculate the last position where the water falls.

3.3 PIC program

The most important part is the code in C. The compiler used is MPLAB IDE v8.00 that Microchip Company recommended. It is a best seller between PIC's compilers and there are a lot of information, examples, drivers, device…

The programmer used is a PICFLASH by USB with ICD (in circuit debbuger) the Mikroelectronika company. It lets to program many series of PICs in circuit, include the PIC18F4525.

3.3.1 PIC configuration

At the beginning it is necessary included a PIC device, the libraries and configure the features.

```
//SPRINKLER.C
#include <18F4550.h>
#fuses INTHS,NOWDt,NOPROTECT,NOLVP,NODEBUG,USBDIV,VREGEN,NOPBADEN
#use delay(clock=8000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

//LIBRARIES
#include <stdlib.h>
#include <math.h>
```

3.3.2 Declaration of variables

Next to the PIC configuration, it declares the constants used in the program.

```
//CONSTANTS
#define BUFFER_SIZE          64
#define DATA_IN              (ext_buffer_next_in != ext_buffer_next_out)
#define lengthh              22             //nº of character that we receive from PC
#define max_field            10             //maxium number of corners
#define max_f                10             //maxium number of intern values
#define nozzle               24             //diàmeter of nozzle
#define pressure             3              //work pressure
#define right                340            //change the direction, angle B1
#define left                 200            //change the direction, angle B2
#define clockk               16             //clockk= 32 és aprox un second
#define timee                10             //each 10 minuts change the band
```

The global variables are defined at the beginning, close to the constants.

```
//VARIABLES
short int sleep_mode,out,ini,inici,tmp;
short int bit_send=FALSE, a=0,zz=0, one=0,st=FALSE;
int pulse, ant_pulse,z,i;
char m=0,nozz, press,countini,countini1;
signed int16 tempa=0,pos,encoder,count,sec,waiting=0;
signed int16 time1=0,time2=0,min2,min1,a1=0,a2=0,aa=0;
signed int32 xd,encoderg,ttt;
```

3.3.3 Interrupts

The PIC18f4550 devices have multiple interrupt sources and an interrupt priority feature that allows each interrupt source to be assigned a high priority level or a low priority level. The high priority interrupt vector is at 000008h and the low priority interrupt vector is at 000018h. High priority interrupt events will interrupt any low priority interrupts that may be in progress.

3.3.3.1 Timer1 interrupt

It entrances inside this interrupt when there are overflow. Then count variable is zero (each second more or less). Each time that cont=0 the second counters are incremented.

```
//TIMER1 INTERRUPT
#INT_TIMER1
void clock_isr(){

    if(--count==0){
        sec++;
                waiting++;
                tempa++;
                a=!a;
                if(zz&&(m==5||m==11)){
                        encoderg--;
                }
                if(!zz&&(m==5||m==11)){
                        encoderg++;
                }

                if(sec>=60){
                        sec=0;
                        min1++;
                        min2++;
                }
                count=clockk;
    }
}
```

3.3.3.2 External interrupt 0

The next interrupt it is produced by INT_EXT0 and it uses to keep the first position. The sprinkler is in the beginning and began to go until end of the field. It leaves to the sleep mode.

```
// EXTERNAL INTERRUPT, KEEP FIRST POSITION
#INT_EXT
void handle_int_ext(){
        delay_ms(10);
        if(!input(PIN_B0)){
                sleep_mode=FALSE;
                m=1;
        }
}
```

3.3.3.3 External interrupt 1 and 2

The next interrupt it is produced by INT_EXT1/2, channel A/B of encoder. When there are interrupt, it looks which is the new state for then, depending the last state increment or decrement the counter of pulses (encoderg).

```
//EXTERNAL INTERRUPT BY ENCODER 1
#INT_EXT1
void handle_int_ext1(){
        delay_ms(1);
        if(input(PIN_B1)){
                if(input(PIN_B2)){
                        pulse=1;
                }
                else{
                        pulse=0;
                }
        }
        else{
                if(input(PIN_B2)){
                        pulse=2;
                }
                else{
                        pulse=3;
                }
        }
        if(m==0){
                sleep_mode=FALSE;
                m=4;
        }
}
```

3.3.3.4 Serial interrupt

The next interrupt it is use to the serial port and communicates the PC and board's sprinkler and GPS receiver. When the PC or GPS send some information, it keeps the bytes in the buffer and later, will read it.

```
//SERIAL INTERRUPT
#INT_RDA
void serial_isr()
{
   ext_buffer[ext_buffer_next_in] = getc();     // get a byte, put it in buffer

   if(++ext_buffer_next_in == BUFFER_SIZE)      // increment counter
      ext_buffer_next_in = 0;
}
```

3.3.4 Routines

3.3.4.1 Routine to take position

In this fragment of the code try to extract a "x" and "y" position. It remember that the x position is in 6 bytes and y position is in others 6 bytes. For example, the units is in the first byte, the decenas is in the second byte, the hundreds is in the next one and the last is the sign (positive or negative).

```
//ROUTINE TO CONVERT CHARS TO INTEGER
void take_pos(){
       int16 ii,jj;
       if(!DATA_IN)                                          // if no data in
       continue;                                             // loop back
       if(get_packet(packet_buffer))
       {
              x_f[3]=x_f[0];
              y_f[3]=y_f[0];
              x_f[0]=0;
              for(ii=8; ii<13; ii++){                        //Extract the x position
                     pos= packet_buffer[ii]-48;
                     for(jj=0; jj<(12-ii); jj++){
                            pos=pos*10;
                     }
                     x_f[0]=x_f[0]+pos;                     //integer with x position

              }
              if(packet_buffer[7]=='-'){x_f[0]=-x_f[0];}
              y_f[0]=0;
              for(ii=14; ii<(19); ii++){                    //extract the y position
              pos= packet_buffer[ii]-48;
                     for(jj=0; jj<(18-ii); jj++){
                            pos=pos*10;
```

```
                        }
                        y_f[0]=y_f[0]+pos;                  //integer with y position
                }
                if(packet_buffer[13]=='-'){y_f[0]=-y_f[0];}
        }
}
```

3.3.4.2 Routine of encoder counter

It knows when there is a new step of encoder by interrupt but, the routine try to increment or decrement a counter of number of steps. Following, there are a piece of code of the routine. When the actual pulse is 0 (pulse can be 0,1,2,3,0… when increment or 0,3,2,1,0… when decrement) , the last step only could be the step number 1 or number 3, else it is a error.

```
//ENCODER COUNTER
void count_encoder(){                               //increment or decrement counter
        switch(pulse){                              //actual step
                case 0:
                        if(ant_pulse==3){           //last step is 3, it increments
                                encoderg++;
                                ant_pulse=pulse;    //actual step is last step
                        }
                        else if (ant_pulse==1){     //last step is 1, it decrements
                                encoderg--;
                                ant_pulse=pulse;
                        }
                        else{
                                error;              //incorrect step
                        }
                        break;
                case 1:                             //repeat for case 1, 2 and 3.
                        if(ant_pulse==0){
                                encoderg++;
                                ant_pulse=pulse;
                        }
                        …
                case 2:
                        …
                case 3:
                        …
}
```

3.3.4.3 Routine to calculate a position.

The finality in this routine is gotten a x and y position in the global reference system {R}. First, it obtains a long of water in the table of nozzle and pressure. Then it tries to know where is the position (degrees) of the gun in the new coordinate system {S}. With the long and the angle of encoder it can get the x and y position in the system {S}.

Then it is necessary to know what is the director vector (actual position minus initial position). This vector will be “x” axis of the coordinate system {S}. So, then it can to change the base and pas the {S} system to {R} system. Finally, it has the further position that the water fall in the global reference system {R}.

```
//CALCULATE POSITION
void calc_encoder(){
        int32 bb;
        float llavor;
        int8 i;

        xd=(int32)length[nozz][press];                      //determinate long of water

        if(encoderg>512){encoderg=0;}
        else if(encoderg<=0){encoderg=512;}

        encoder=encoderg*7/10;                              //convert the 512 positions
                                                            // of encoder a 360°

        if((encoder>=0)&&(encoder<90)){                     //get a x and y position {S}
                y_f[7]=xd*(int32)table_cos[encoder]/100;
                x_f[7]=xd*(int32)table_cos[90-encoder]/100;
        }
        else if((encoder>=90)&&(encoder<180)){
                y_f[7]=-1*xd*(int32)table_cos[180-encoder]/100;
                x_f[7]=xd*(int32)table_cos[encoder-90]/100;
        }
        else if((encoder>=180)&&(encoder<270)){
                y_f[7]=-1*xd*(int32)table_cos[encoder-180]/100;
                x_f[7]=-1*xd*(int32)table_cos[270-encoder]/100;
        }
        else{
                y_f[7]=xd*(int32)table_cos[360-encoder]/100;
                x_f[7]=-1*xd*(int32)table_cos[encoder-270]/100;

        }

        x_f[5]=x_f[0]-x_f[4];                               //director vector
        y_f[5]=y_f[0]-y_f[4];                               //actual pos - initial pos

        x_f[6]=y_f[5];
        y_f[6]=-1*x_f[5];

        bb=abs(x_f[5]*x_f[5])+abs(y_f[5]*y_f[5]);           //vector unitari

        i=0;
        llavor=45.0;

        for(i=0; i<7; i++){
```

```
                llavor=(bb/(2*llavor))+(llavor/2);

        }
        bb=(int32)llavor;

        x_f[8]=(x_f[5]*x_f[7]+y_f[5]*y_f[7])/bb + x_f[0];   //change de base. Coordinate
        y_f[8]=(x_f[6]*x_f[7]+y_f[6]*y_f[7])/bb + y_f[0];   //system {S} to {R}

}
```

3.3.4.4 Routine to know if the point is inside or outside.

The most important routine is calculating a position in/out. The main idea is taking the one point (inside or outside the polygon), and find what are the vectors between the point and different vertex of the polygon. Then it finds what is the angle between first vector and the next one vector. It realizes the same with other vectors to get an others angles.

If the sum of all angles is 360º the point is inside, else, if the sum of all angles is 0º is outside. The system is very robust because there aren't collisions and complicates situations. It always work and when the point is outside it active the flag (out=true).

```
//CALCULATE A POSITION IN/OUT
void calc_position(){
        int8 i,j,cos_angle;
        int16 sum_ang=0;
        int32 aa, aaa,ddd, bb, cc, dd, angle;
        float  llavor;

        for(j=1; j<=x_field[0]; j++){                       //Calculate the angles
                x_f[1]=x_field[j]-x_f[8];                   //V1 vertex - actual pos.
                y_f[1]=y_field[j]-y_f[8];
                if(j==x_field[0]){
                        x_f[2]=x_field[1]-x_f[8];           //V2 vertex+1 - actual pos.
                        y_f[2]=y_field[1]-y_f[8];
                }
                else{
                        x_f[2]=x_field[j+1]-x_f[8];
                        y_f[2]=y_field[j+1]-y_f[8];
                }
                                                            //COS@=(|V1*V2|)/(|V1|*(V2))
                aa=(x_f[1]*x_f[2]) + (y_f[1]*y_f[2]);       //V1*V2
                aaa=abs((x_f[1]*x_f[2]) + (y_f[1]*y_f[2])); //|V1*V2|
                dd=(x_f[1]*y_f[2]) - (x_f[2]*y_f[1]);       //VECTORIAL PRODUCT
                ddd=abs((x_f[1]*y_f[2]) - (x_f[2]*y_f[1])); //MODULL OF VECTORIAL PRODUCT
                bb=abs(x_f[1]*x_f[1])+abs(y_f[1]*y_f[1]);   //|V1x*V1x+V1y*V1y|
                cc=abs(x_f[2]*x_f[2])+abs(y_f[2]*y_f[2]);   //|V2x*V2x+V2y*V2y|

                llavor=45.0;                                //square root V1
```

```
                for(i=0; i<7; i++){
                        llavor=(bb/(2*llavor))+(llavor/2);

                }
                bb=(int32)llavor;

                llavor=45.0;                                    //square root V2
                for(i=0; i<7; i++){
                        llavor=(cc/(2*llavor))+(llavor/2);
                }
                cc=(int32)llavor;

                angle=((100*aaa)/(bb*cc));                      // cos(angle)
                if(angle>100){angle=100;}
                cos_angle=table_acos[(char)angle];              //arc_cos(angle)

                if(aa!=aaa){
                        cos_angle=180-cos_angle;
                }

                if(dd==ddd){                                    //decide if sum or rest
                        sum_ang=sum_ang - (int16)cos_angle;
                }

                else{
                        sum_ang=sum_ang + (int16)cos_angle;

                }
        }

        if((sum_ang>180)){                                      //if angle>180 is outside
                out=FALSE;
        }
        else{                                                   //if not, it is inside
                out=TRUE;
        }
}
```

3.3.5 Main structure

3.3.5.1 Initialize

Inside the main program configure the oscillator, active the interrupts and configure the input and output ports.

```
void main() {
        setup_oscillator(OSC_8MHZ);

        setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
```

```
set_timer1(0);

enable_interrupts(GLOBAL);
enable_interrupts(int_rda);
enable_interrupts(int_timer1);
enable_interrupts(int_ext);
enable_interrupts(int_ext1);
enable_interrupts(int_ext2);

SET_TRIS_A( 0xFF );
SET_TRIS_B( 0xFF );
SET_TRIS_D( 0x00 );
SET_TRIS_E( 0x00 );
```

3.3.5.2 Main program

When it arrives a while(1) structure, continuously, it repeats the same code. Read the counter of encoder, look if there is sleep mode and choose the actual case.

```
while(1) {                      //loop
   count_encoder();             //look if the encoder change the step or not.

   if(sleep_mode){              //look if it is in the sleep mode
          sleep();
   }
   switch(m){                   //choose the case
```

In the first case, take the first position by GPS and keep this position in the PIC's internal EEprom memory. If the next position is the same that the last one, it continues in the same case. If not, it jumps in the case number two.

```
   case 1:                      //case 1. Take the first position

          …
                               //write in the eeprom memory the first position
          if((x_f[3]==x_f[0])&&(y_f[3]==y_f[0])){
                 m=1;
                 i=0;
                 for(i=0; i<4; i++){
                        write_eeprom((3-i), x_f[4]>>(i*8));
                        delay_ms(10);
                 }
                 for(i=0; i<4; i++){
                        write_eeprom((7-i), y_f[4]>>(i*8));
                        delay_ms(10);
                 }
          }                    //If the new position is the different that the
```

```
        else{                   //last one, it changes the case number two.
                m=2;
        }
        break;
```

In the case number two take the last position and it knows that is the last because after few minutes don't arrive a new position different the last one. Sure that this is the end position. Than it jumps to sleep mode and the microcontroller will be waiting for a external interrupt by encoder.

```
case 2:                         //take a second position
        output_d(0x20);
        take_pos();

        …

        if((x_f[3]==x_f[0])&&(y_f[3]==y_f[0])){
                sleep_mode=TRUE;
                m=0;
        }
        else{
                m=2;
        }

        x_f[0]=x_f[0]<<4;
        y_f[0]=y_f[0]<<4;
        break;
```

When there are a external interrupt 1 from channel A o external interrupt 2 from channel B of encoder it goes in the case number 4. Ii this case read the EEprom memory where there is the initial position.

```
case 4:
        x_f[4]=0;
        y_f[4]=0;

        for(i=0; i<4; i++){                     //read the x initial position
                ttt=(int32)(read_eeprom(3-i));
                ttt=ttt<<(8*i);
                x_f[4]=x_f[4]+ttt;
        }
        for(i=0; i<4; i++){                     //read the y initial position
                ttt=(int32)(read_eeprom(7-i));
                ttt=ttt<<(8*i);
                y_f[4]=y_f[4]+ttt;
        }
```

```
m=5;
min2=0;
break;
```

After case number 4, m=5 and it entrances in the case number 5. In this case it develops and organizes the actions. First take an actual position (take_pos()) from GPS. If the encoder didn't detect a inductive sensors or a reference point, jump and it is waiting to find them.

Then it calculates the position (calc_encoder()) and decide if it is inside or outside the field (calc_position()). Send a position and other parameters to PC via RS232. After this, it decides if it is necessary to do something or not.

```
case 5:
        output_d(0x50);
        take_pos();                     //take a actual position from GPS

        if(!st){continue;}              //waiting a reference point or ind. sensors
        calc_encoder();                 //routine that it calculates the position
        calc_position();                //routine that if it is outside or inside
        comunicat();                    //send information to PC
```

One of the first questions is to know which are the limits of the field and put the variable ini in low level.

```
if((OUT)&&(ini)){
        m=7;
        out=FALSE;
        countini++;
        countini1=0;
        calc_area();
        if(countini==2){
                ini=FALSE;
                countini=0;
        }
}
```

Seeming a before question, when the position is outside it is necessary to change the way (m=7).

```
else if((OUT)&&(!ini)){
        m=7;
}
```

However, if the gun is between angles B1 (right) and B2 (left) and it is at the beginning, normally, not change the way. There only are one specific situation that the sprinkler will

change the way. When the sprinkler is in the middle of the field and there aren't obstacles close to the sprinkler, it can irrigate in the normal position.

```
else if(((encoder<right)&&(encoder>left))&&ini){
        tempa=0;
        countini1++;
        if(one){
                if(countini1>=2){
                        m=7;
                        ini=FALSE;
                }
                else{
                        m=5;
                }
        }
        one=FALSE;
}
```

The last question is when there are B1 or B2 and ini is false. Normally, it changes the way again but, sometimes, depending the tmp (control the time it needs irrigate one band and another band) won't change the way and the sprinkler will pas to irrigate other band.

```
else if(((encoder<right)&&(encoder>left))&&!ini){
        tempa=0;
        if(one){
                m=7;
                if(tmp){
                        if((encoder>=90)&&(encoder<270)){
                                if(min1>time1){
                                        m=5;
                                        min1=0;
                                }
                        }
                        else{
                                if(min1>time2){
                                        m=5;
                                        min1=0;
                                        countini1++;
                                }
                        }
                }
        }
        one=FALSE;
}
```

If the program are not entrance inside the last questions, the sprinkler can irrigate normally and return to the case number 5 and it begins the sequence again.

```
else{
        m=5;
        one=TRUE;
        if((encoder>85)&&(encoder<95)){
                tmp=FALSE;
        }
}

break;
```

3.4 Simulations

This section talks about the simulations in Labview of the project. We can see the different graphics in dissimilar sprinkler's position. In the left side, there are an information about the position of sprinkler (GPS +xxxxx-yyyyy).

Also, it appears which is the end position that the water falls (ddd Imm+xxxxx-yyyyy m). The first information, "ddd" it is the angle of encoder in degrees. Then, there is a capital letter "I" or "O" that it shows when the end position it is outside or inside. Next, there are information about the time that it irrigates in one band or another. Following, there is a "x" and "y" position and the last information, it is the actual time.

3.4.1 Track 1,

3.4.1.1 Beginning.

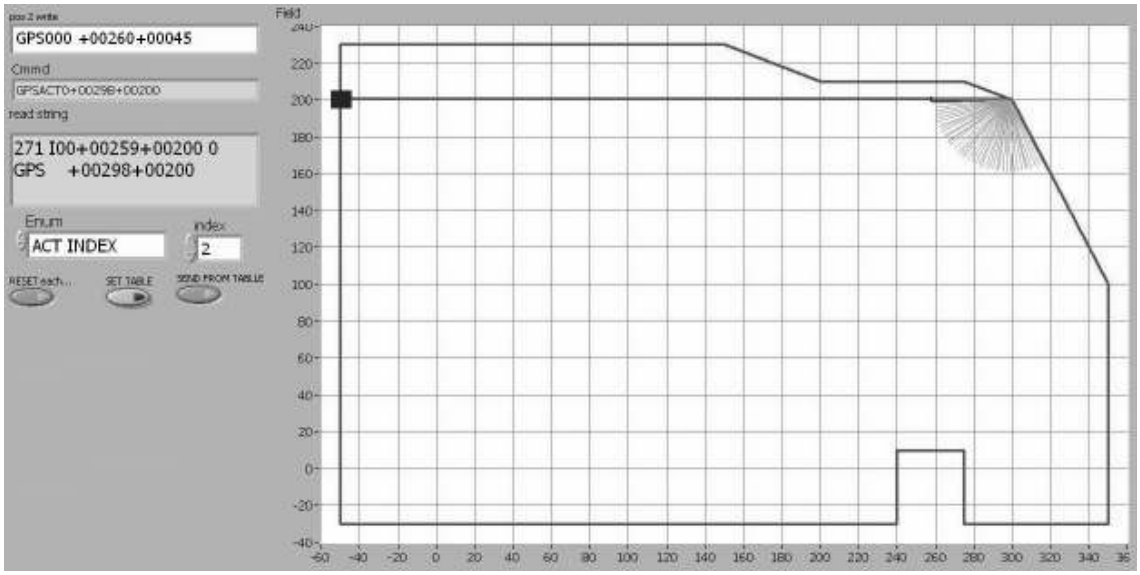


Fig 7. Track1, beginning.

In the initial moment, the sprinkler searches the two points where the water falls outside the field.

3.4.1.2 After 60 m.

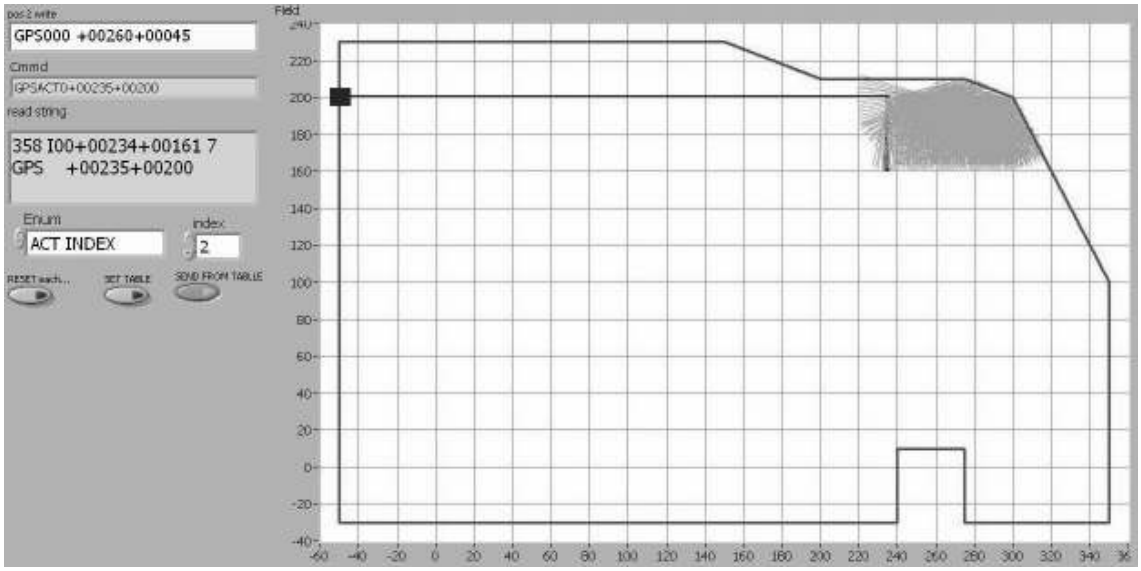


Fig 8. Track1, after 60 m.

The sprinkler irrigate only in one side.

3.4.1.3 After 140 m.

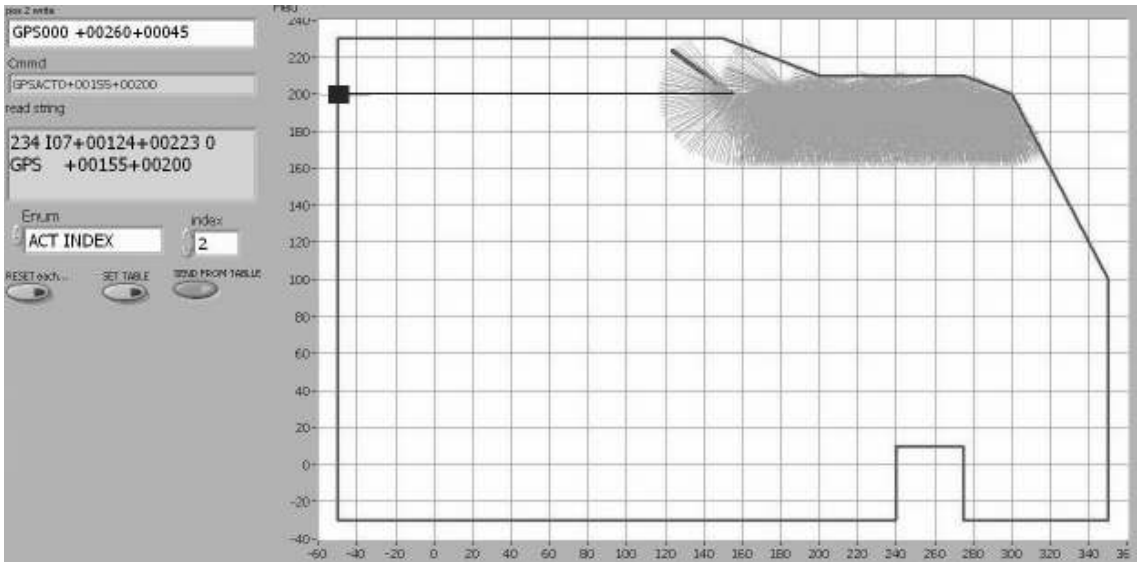


Fig 9. Track1, after 140 m.

The sprinkler, each “x” time checks if it can irrigate in the other band or not.

3.4.1.4 After 240 m.

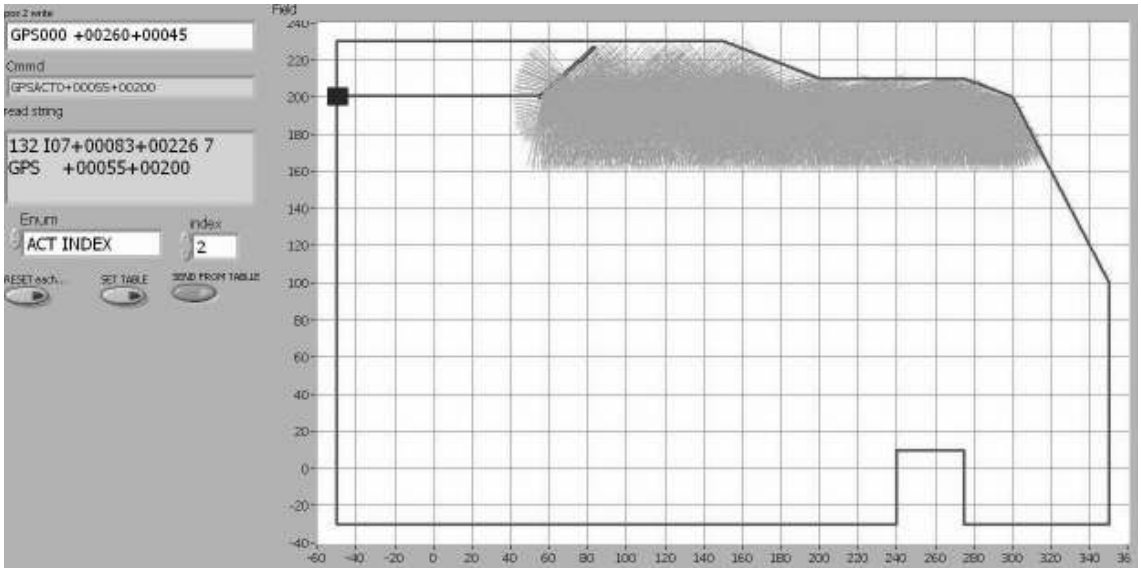


Fig 10. Track1, after 240 m.

In this case, the sprinkler irrigates in one band but, it uses practically full angle.

3.4.1.5 Ending.

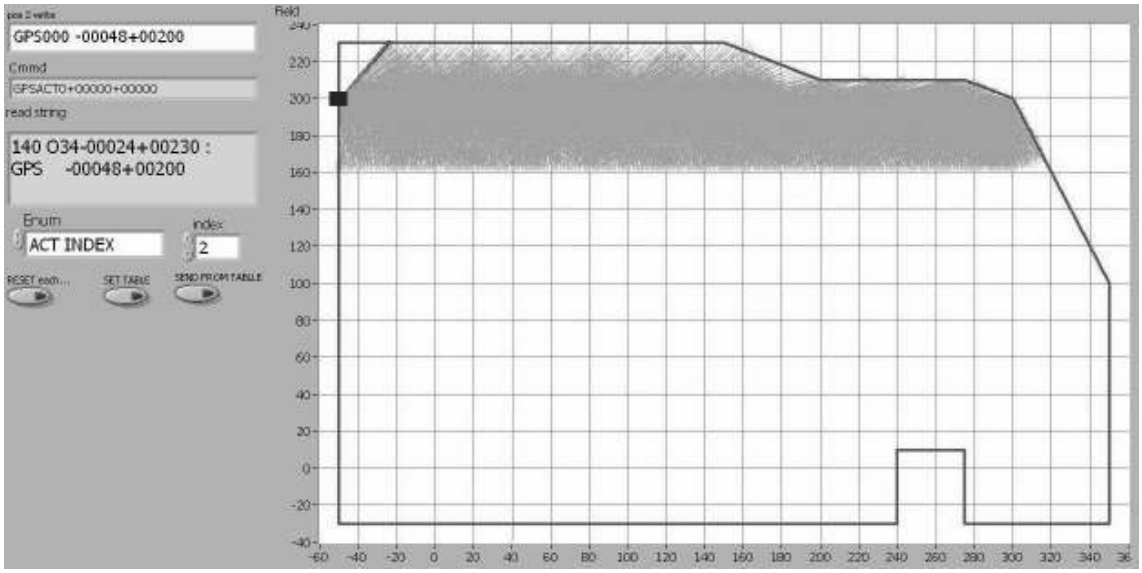


Fig 11. Track1, ending

The sprinkler has arrived at the end and the pump would become to turn off.

3.4.2 Track 2.

3.4.2.1 Beginning

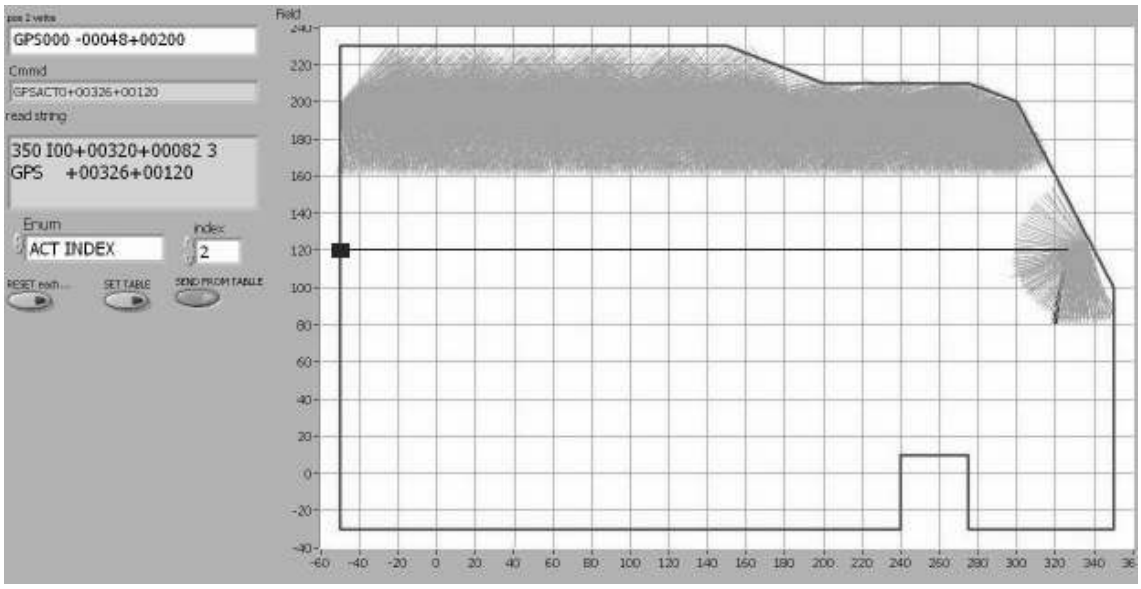


Fig 12. Track2, beginning.

This second track it can see that there aren't so difficulties. At the beginning, the splinker only irrigates in one band because in the other band, there aren't sufficient angle's width.

3.4.2.2 After 40 m.

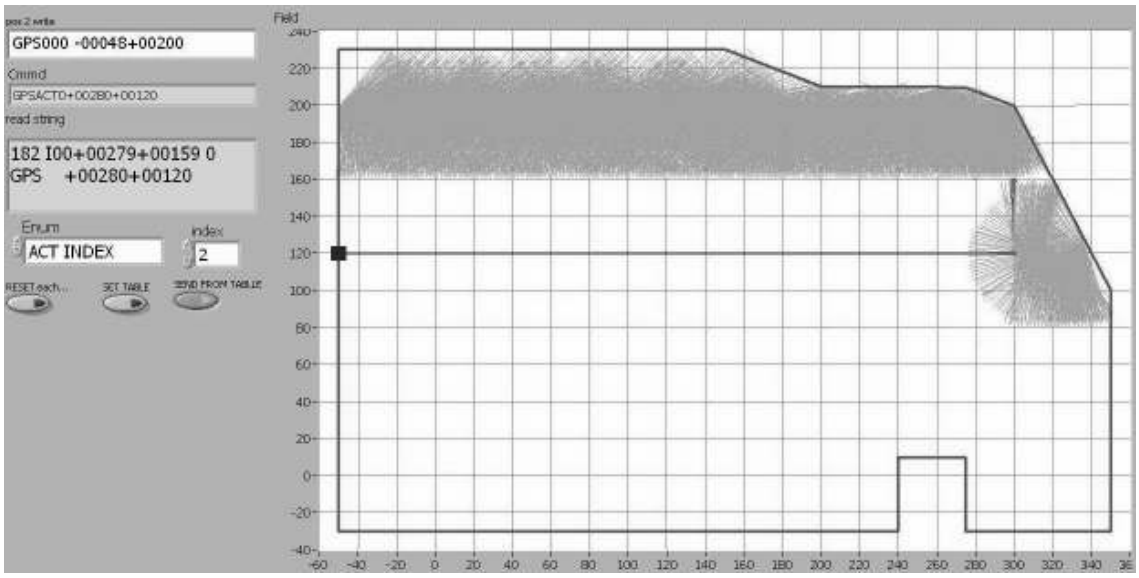


Fig 13. Track2, after 40 m.

Now, after the sprinkler has run few meters irrigates other side.

3.4.2.3 After 190 m.

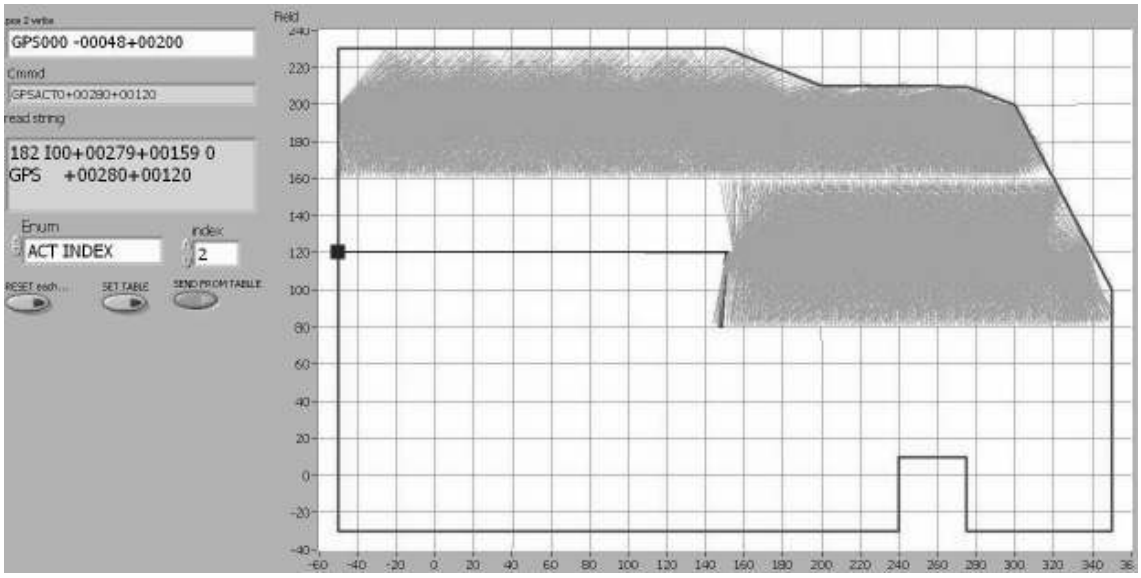


Fig 14. Track2, after 190 m.

The sprinkler irrigates with a complete range without obstacles.

3.4.2.4 Ending.

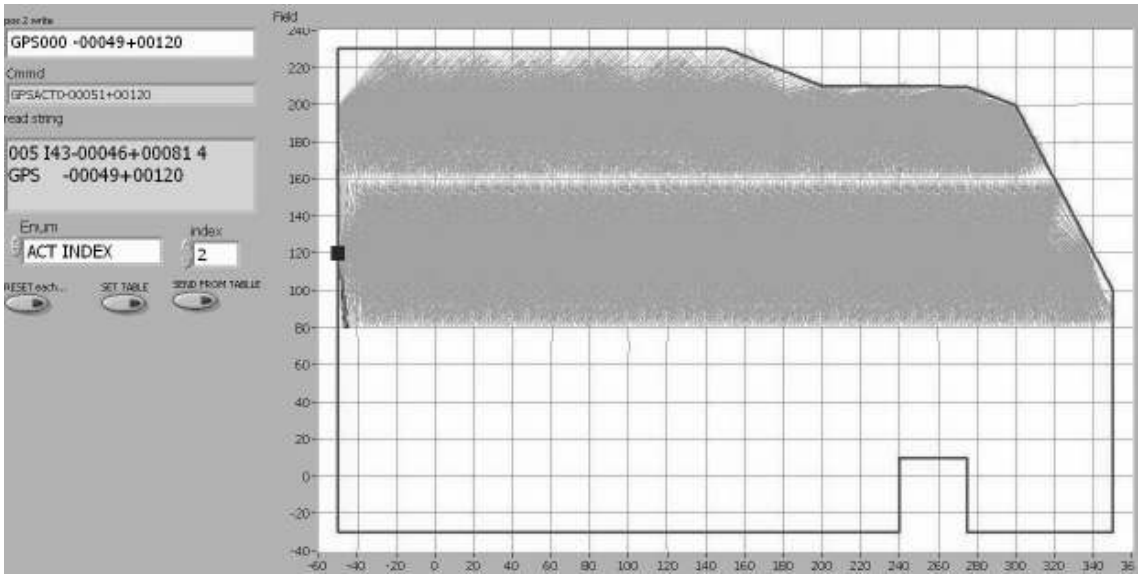


Fig 15. Track2, ending.

The splinker finishes to irrigate the second track.

3.4.3 Track 3.

3.4.3.1 Beginning

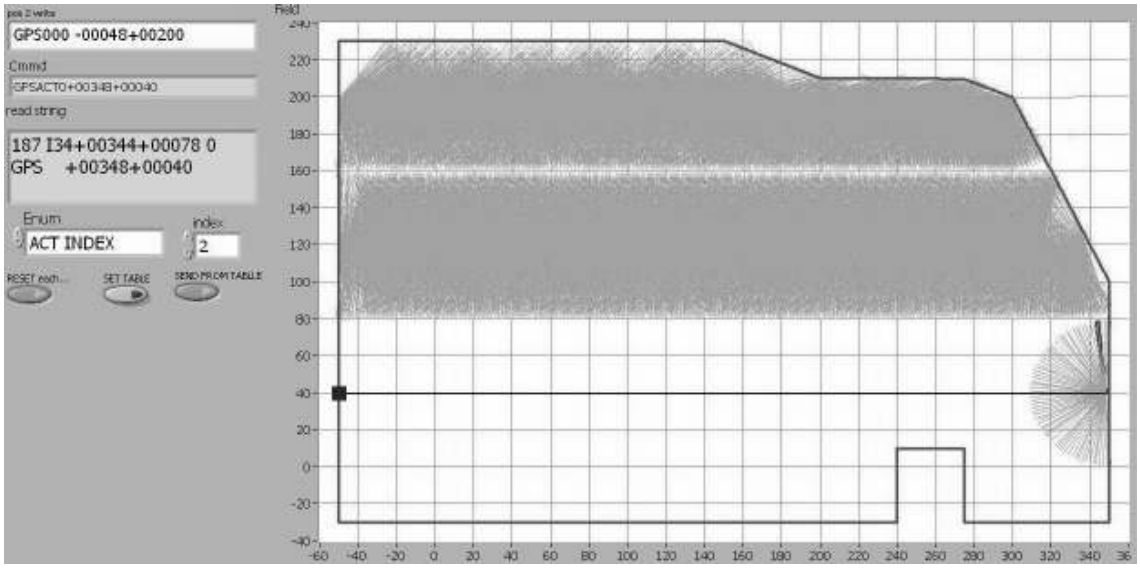


Fig 16. Track3, beginning

The splinker at the beginning searches a two end positions. Each 7 minutes, the sprinkler changes the band to irrigate. So, we can see the different times that splinker will spend in each side (three and four, respectability)

3.4.3.2 After10 m.

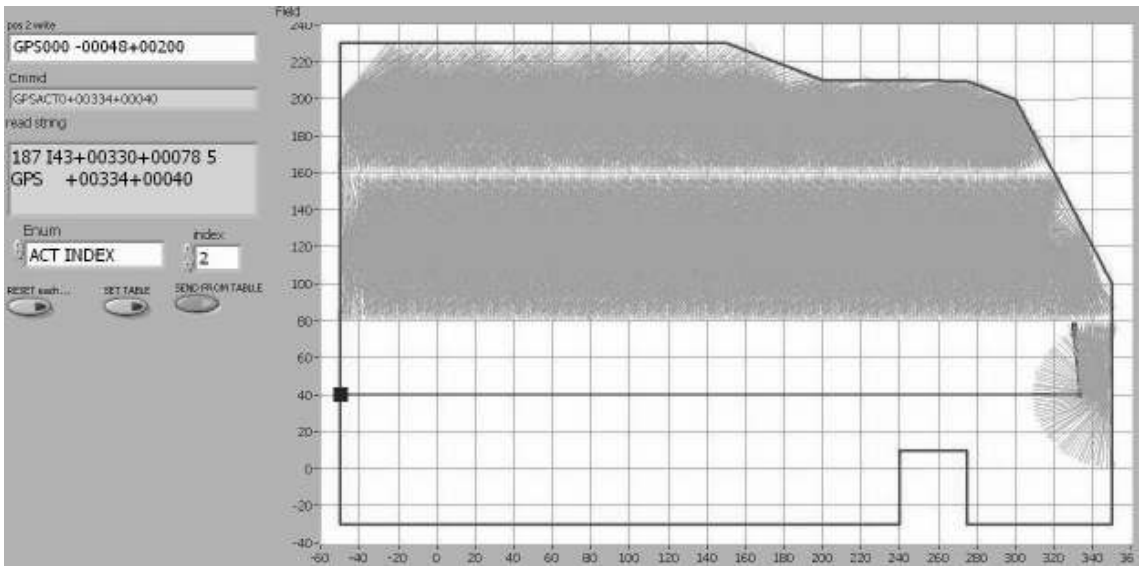


Fig 17. Track 3, after 10 m.

The splinker irrigates in the left side during 4 minutes and now, the actual time is five. So, the sprinkler will change the band and will irrigate the other band during 3 minutes more.

3.4.3.3 After 70 m.

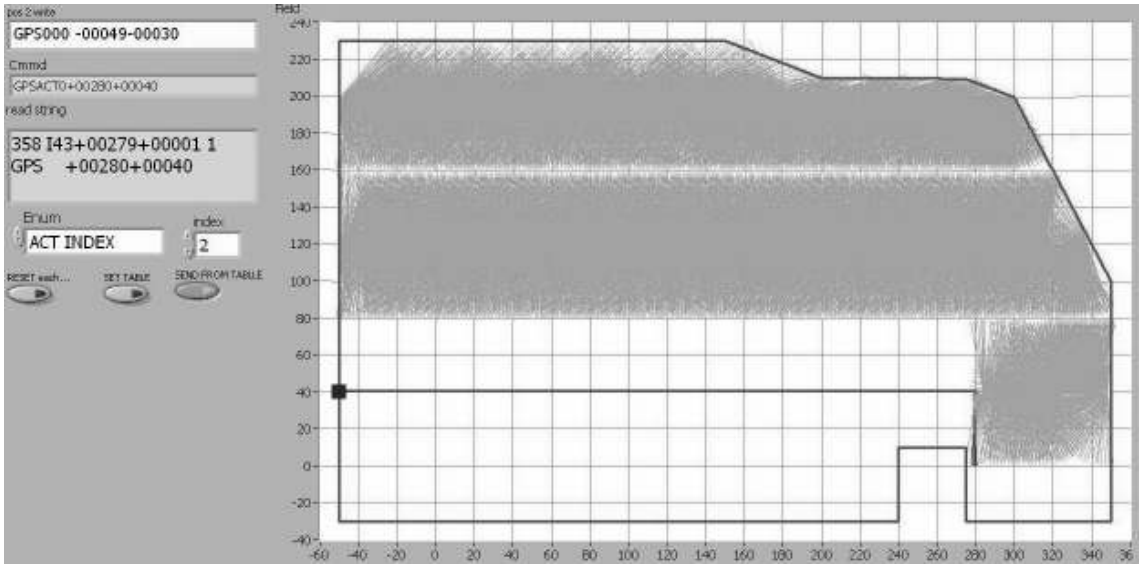


Fig 18. Track 3, after 70 m.

The splinker finds an obstacle and will have to change the way each time that find it.

3.4.3.4 After 90 m.

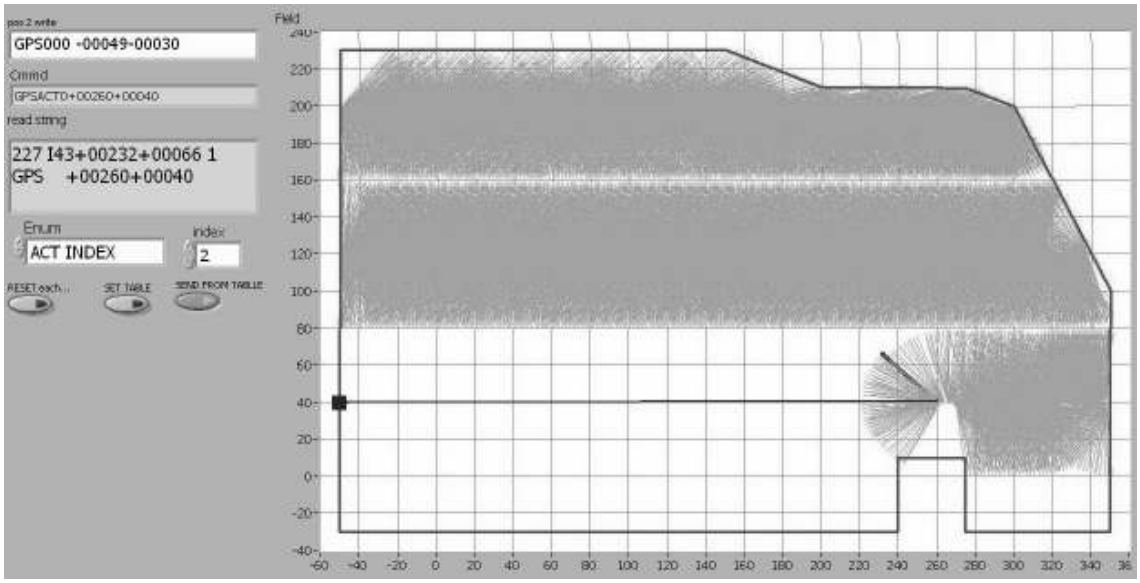


Fig 19. Track 3, after 90 m.

Now, the splinker is just in the middle of the obstacle and it tries to know which shape the field adopts.

3.4.3.5 After 110 m.

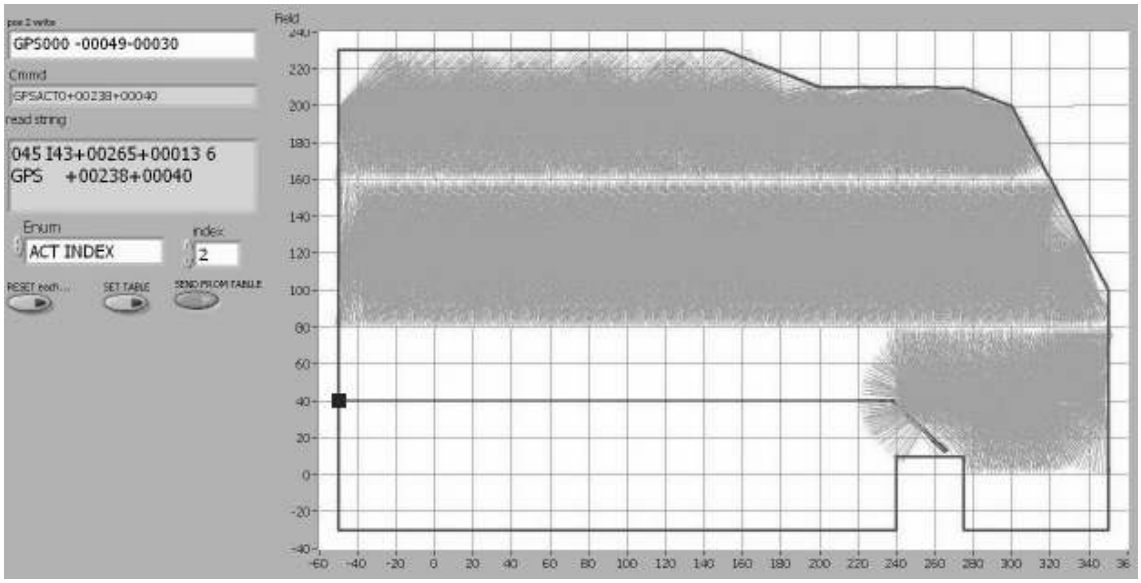


Fig 20. Track 3, after 110 m.

The splinker has reduced the irrigation area without throw water out the field.

3.4.3.6 After 180 m.

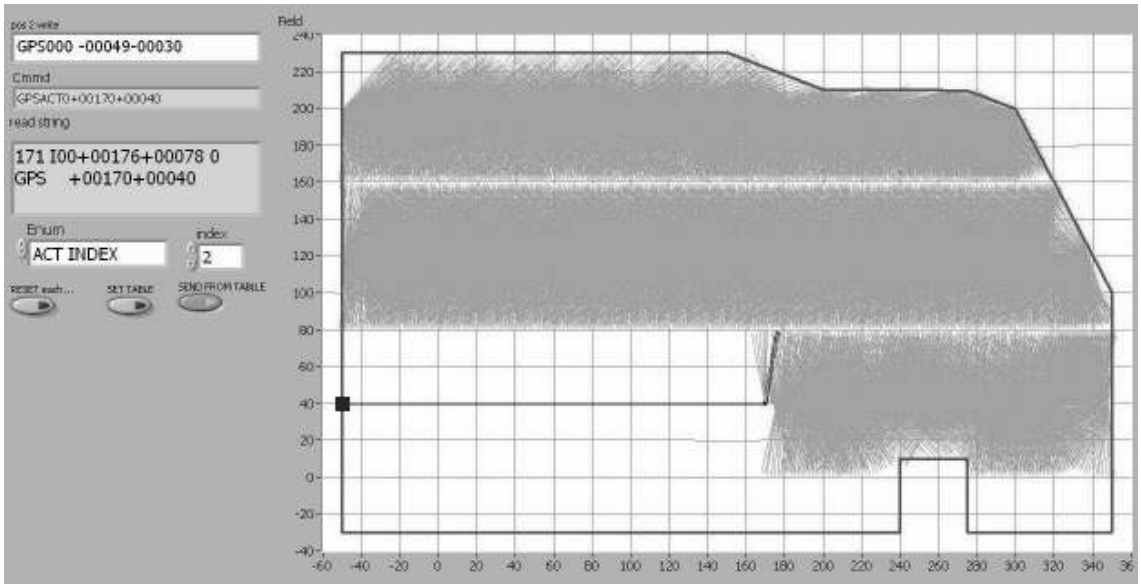


Fig 21. Track 3, after 180 m.

The sprinkler has overcome the obstacles and irrigates normally.

3.4.3.7 Ending.

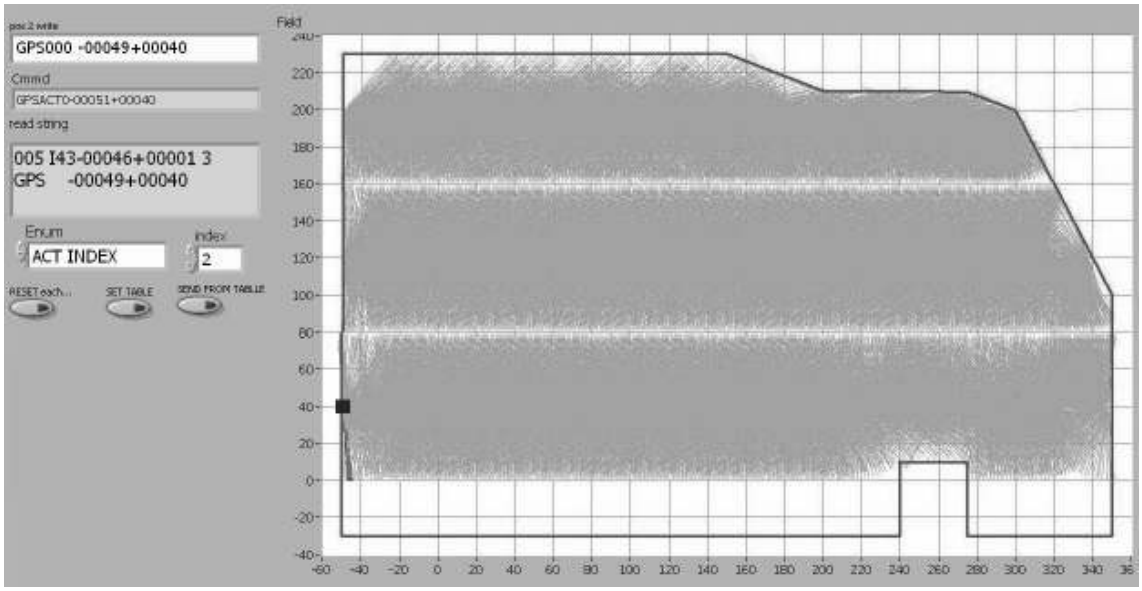


Fig 22. Track 3, ending.

Finally, the last graphic shows when the sprinkler is in the end of track 3.

In the real field, if the maximum range of water is 50 m, the distance between different tracks is approximately 80 m.

4 Electronic system in the hose

4.1 Block diagram

The second block diagram is an electronic system used in the hose (Fig 23). In this case, the microcontroller only controls a turbine, which is regulated by an electric driven by DC motor. This turbine is directly connected to a 4 speed gearbox for increased output and efficiency at optimal speeds. The electrovalve is suitable to switch off the irrigation when the travel sprinkler has arrived at the end of its route.

The inductors sensors are used to know the electronic valve position and also to know which is the actual speed gearbox. The analog pressure sensor determinates the entrance pressure on the machine. Finally, the RF transceiver comunicates with PCB_1 in the traveller sprinkler.

The power supply is generated by another solar cells and it has another storage battery supports the operation of the gun at night and on cloudy days.

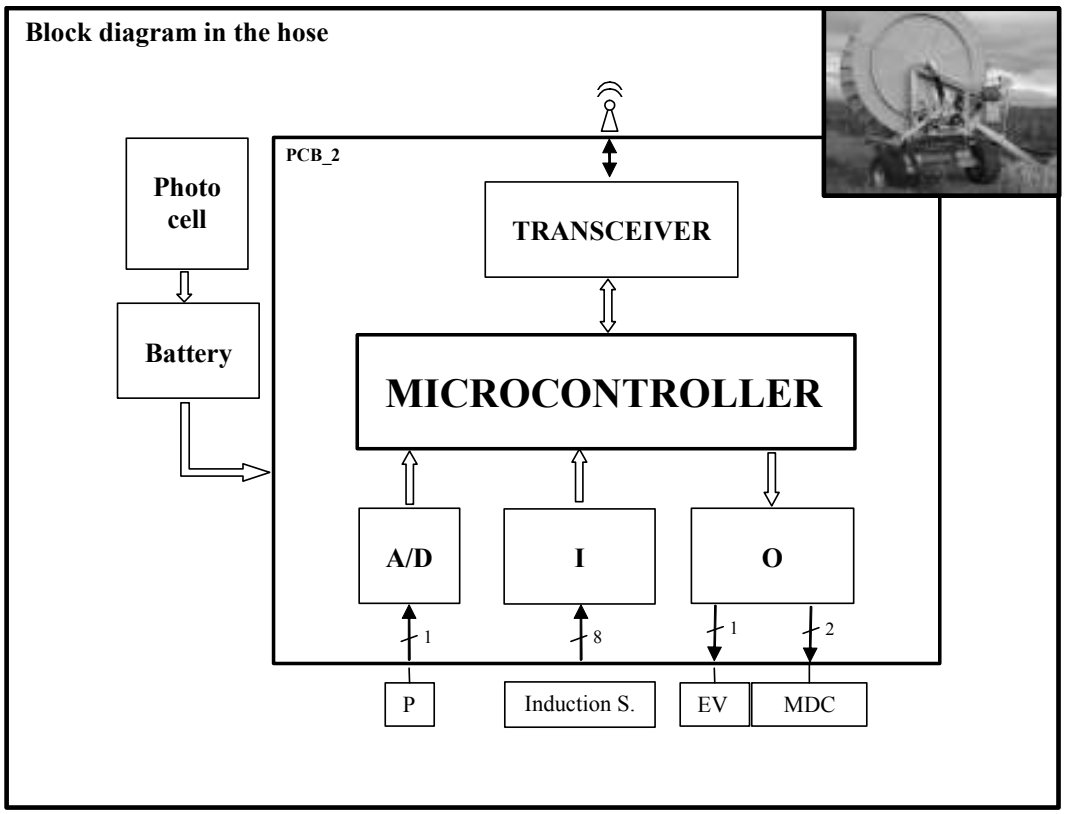


Fig 23. Block diagram in the hose

4.2 Description of the electronic components

4.2.1 Microcontroller PIC16F877

The necessity to use a microcontroller to develop this application has been essential to choose anyone that offer the system provisions require. The big demand that there are the last years with a good technique characteristics how; low cost, low power, quality, reliability, efficiency and a lot of information, the PIC microcontrollers (Peripheral Interface Controller) are an excellent device to use in many applications.

The architecture of the PIC 16FXXX family is a Harvard type with a 8 bits dates bus and a 14 bits instructions bus. It has a CPU the RISC type (Reduce instruction Set Computer) where only use 35 instructions. The chosen microcontroller is PIC 16F877, the mid-range in the Microchip's house. The main characteristics of this microcontroller are the next (Table 20).

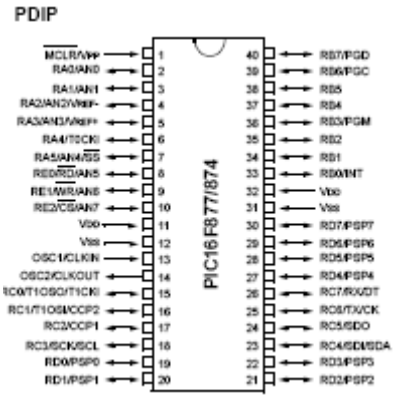
Features	
High performance RISC CPU	
Operating speed: DC-20MHZ clock input	
Up to 8K x 14 words of FLASH Program Memory Up to 368 x 8 bytes of Data Memory (RAM) Up to 256 x 8 bytes of EEPROM Data Memory	
Interrupt capability (up to 14 sources)	
Programmable code protection	
Power saving SLEEP mode	
In-Circuit Serial Programming (ICSP)	
Wide operating voltage range: 2 to 5.5V	
High source current: 25mA	
Low power consumption	
Three timers: 8 and 16 bit timer/counter with prescaler	
Two capture, compare, PWM modules	
Ten bit multi channel analog to digital converter	
Synchronous Serial Port (SSP) with SPI and I2C	
Universal Synchronous Asynchronous Receiver-Transmitter (USART/SCI) with 9 bit address detection	
Parallel Slave Port (PSP), RD, WR and CS control	

Table 20. PIC 16F877 features

4.2.2 Reference voltage REF02

The REF02 precision voltage reference provides a stable 5V output that can be adjusted over a ±6% range with minimal effect on temperature stability. Low cost, low noise, an low power make the REF02 an excellent choice whenever a stable voltage reference is required (Table 21).

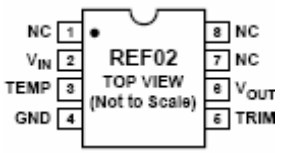
Features	
5 V output: ±0.3% maximum	
Temperature voltage output: 1.96mV/ºC	
Excellent temperature stability: 8.5 ppm/ºC maximum	
Low noise: 15 uV p-p maximum	
Low supply current: 1.4 mA maximum	
Wide input voltage range: 7V to 40V	
High load-driving capability: 10mA	
No external components	
High source current: 25mA	
Low power consumption	
Short-circuit proof	

Table 21. REF02 features

4.2.3 Driver ICL76673

The ICL7667 is a dual monolithic power MOSFET driver designed to translate TTL inputs to high voltage/current outputs. Its low delay and transition times make it ideal to drive power MOSFETs for switching power supplies, motor controllers, and DC-DC converters.

The ICL7667's high speed minimizes power losses in switching power supplies and DC-DC converters due to rapid charging/discharging of the gate capacitance of the power MOSFETs. The ICL7667 inputs are TTL compatible, enabling direct interface to common switched mode power supply controllers (Table 22).

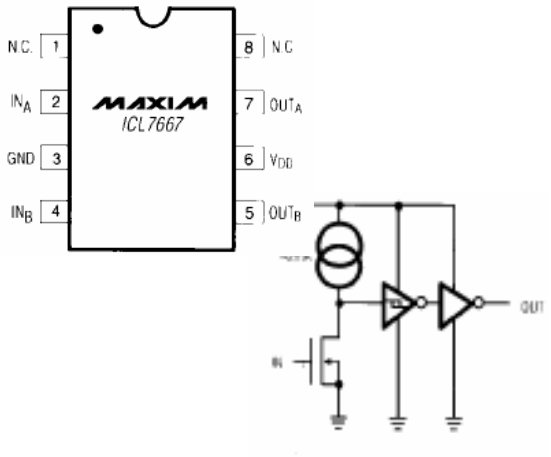
Features	
Fast rise and fall times: 20ns with 1000 pF load.	
Wide supply range: 4.5 to 17V	
Low power consumption: 6mW with inputs low and 120 mW with inputs High.	
TTL/CMOS input compatible	
Package dissipation plastic dip: 300 mW	
Storage temperature: -55 to 160ºC	
Low Rout = 4Ω	
Low Rout = 4Ω	

Table 22. ICL7667 features

4.2.4 Mosfet IRF540

These are N-Channel enhancement mode silicon gate power field effect transistors. Thet are advanced power MOSFETs designed, tested, and guaranteed to withstand a specified level of energy in the breakdown avalanche mode of operation.

All of theses power MOSFETs are designed for applications such as switching regulators, switching converters, motor drivers, relay drivers, an drivers for high poser bipolar switching transistors requiring high speed and low gate drive power. These types can be operated directly from integrated circuits (Table 23).

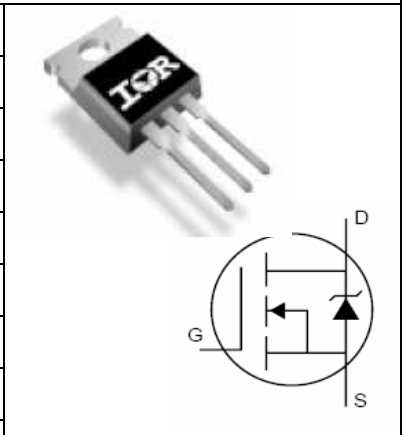
Features	
Continuous drain current: 25A	
Drain to gate voltage: 80V	
Single pulse avalanche energy rated	
Nanosecond switching speeds	
Linear transfer characteristics	
High input impedance	
Operating and storage temperature: -55 to 175ºC	
Maximum power dissipation: 150 W	
Drain to source on resistance: 80Mω	

Table 23. ICL7667 features

4.2.5 Diode BYW29-200

This state-of-the-art device is designed for use in switching power supplies, inverters and as free wheeling diodes (Table 24).

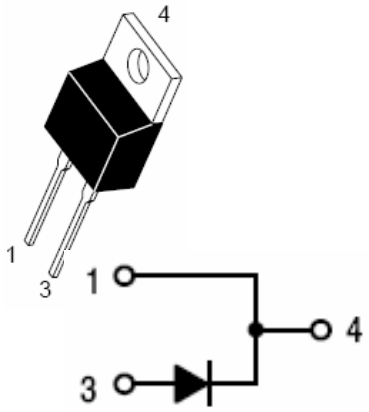
Features	
Working peak reverse voltage:200 V	
Average rectified current 8 A	
Peak repetitive forward current: 16 A	
Non repetitive peak surge current 100 A	
Low forward voltage	
Low leakage current	
High temperature glass passivated junction	
Operating and storage temperature: -55 to 175ºC	
Maximum power dissipation: 10 W	

Table 24. Diode BYW29-200

4.3 Input connectors.

4.3.1 Connector J1. Pressure sensor.

The connector J1 has four pins; one wire for power supply, two wire more for I2C interface. It has a Serial Clock Line (SCL) input and serial digital output data line and finally, one wire for GND connection. The output of the device is a corrected pressures value in a hexadecimal format with 12-bit resolution. It uses to know the absolute pressure in the sprinkler and to determinate the width irrigation (Table 25).

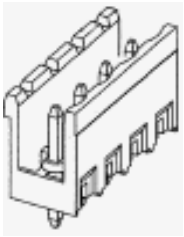
From connector J1	To Pic 16F877	Connector PCB
Pin_1. VCC	-	
Pin_2. SCL	Pin_34. RB1/SCL	
Pin_3. SCI	Pin_35. RB2/SDA	
Pin_4. GND	-	

Table 25. Pressure sensor connection

4.3.2 Connector J2. Inductive sensors.

It has reserved four pins (from pin_2 to pin_5) for NPN inductive sensors. These sensors are used to know when the butterfly valve is open, close and two intermediate positions (Table 26).

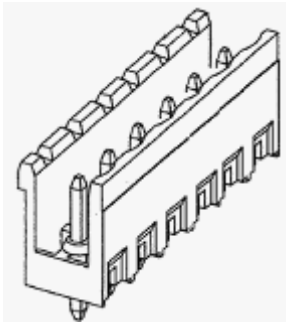
From connector J2	To Pic 16F877	Connector PCB
Pin_1. VCC	-	
Pin_2. Inductive S1	Pin_3. RA1	
Pin_3. Inductive S2	Pin_4. RA2	
Pin_4. Inductive S3	Pin_6. RA4	
Pin_5. Inductive S4	Pin_7. RA5	
Pin_6. GND	-	

Table 26. Inductive sensors connection

4.4 Output connectors.

4.4.1 Connector J3. Electronic butterfly valve.

The connector J3 uses to connect the electronic butterfly valve that let to regulate the speed of the traveler sprinkler. There is commune pin (pin 3) connector and the pin 1 connector lets to open the butterfly valve (motor right) and pin two connector lets to close the butterfly valve (motor left). The open/close valve has a proportional control (Table 27).

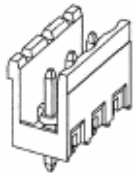
From Pic 16F877	To connector J3	Connector PCB
Pin_19. RD0	Pin_1. Electrovalve_R	
Pin_20. RD1	Pin_2. Electrovalve_L	
-	Pin_3. GND	

Table 27. Electronic butterfly valve connection

4.4.2 Connector J4. Electrovalve.

The connector J4 uses to connect the electrovalve that let to turn of the irrigation. There is GND (pin 2) connector and the pin 1 connector lets to open the valve. It is automatically close when there isn't current supply (Table 28).

From Pic 16F877	To connector J4	Connector PCB
Pin_21. RD2	Pin_1. Electrovalve	
-	Pin_2. GND	

Table 28. Electrovalve connection

4.4.3 Connector J5. Transceiver.

The communication between PIC 16F877 and transceiver is by RS-232 protocol. It needs the one pin of power supply, other for GND connection and two pins more to do communication (Rx and Tx, respectability). The PIC 16F877 microcontroller has a hardware integrated USART and let to work with the asynchronous (full duplex) mode.

It uses to communicate with PCB_1 in the traveller sprinkler (Table 29).

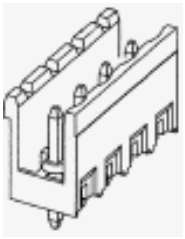
From Pic 16F877	To connector J4	Connector PCB
-	Pin_1. VDD	
Pin_25. RC6	Pin_2. GPRS_RX	
Pin_26. RC7	Pin_3. GPRS_TX	
-	Pin_4. GND	

Table 29. GPRS connection

4.4.4 Connector J7. Programmer connector.

One of the possibilities to program a microcontroller is by using a connector. There are only a 2x5 connector between microcontroller and the other parts of the board. Once it plugs in the programmer connector, it will be able to program PIC in system.

Once the development of a device is finished, the jumpers have to be restored for enabling the device to work without programmer. These jumpers establish connections from MCLR, RB6 and RB7 to peripherals on the board (Table 30).

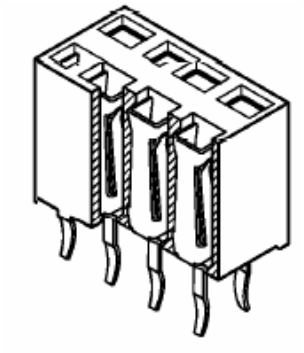
From connector J1	Connector PCB
Pin_1. VCC	
Pin_2. PIC_VCC	
Pin_3. TX/RB6	
Pin_4. No connect	
Pin_4. RX/RB/	
Pin_6. No connect	
Pin_7. MCLR/VPP	
Pin_8. VCC	
Pin_9. GND	
Pin_10. GND	

Table 30. Programmer connection

4.5 Sensors technical characteristics

4.5.1 Microstructure pressure sensor

The ASDX DO series sensors provide a very cost effective solution for pressure applications that require small size plus performance (Table 31). Device is available to measure absolute pressure up to 100psi (SDX DO100).

GENERAL SPECIFICATIONS	
Supply Voltage (Vs)	4.75 Vdc to 5.25 Vdc
Maximum Supply Voltage*	6.50 Vdc max.
Current Consumption	6 mA typ.
Output current - sink	2 mA max.
Output current - source	2 mA max.
Lead Soldering Temperature	2 Sec to 4 Sec @ 250 °C [482 °F]

Table 31. Pressure sensor general specifications

The ASDX DO device is in a standard DIP package and provides digital correction of sensor temperature coefficients and non-linearity. It use I2C compatible protocol, which allows easy interfacing to most commonly used microcontroller without additional components or electronic circuitry (Table 32).

Features	
Available in absolute	
Calibrate and temperature compensated output	
Pressure ranger 0 psi to 100 psi	
Response teim 8 ms	
Standard DIP package	
ASIC enhanced Ouptut	
I2C compatible protocol	

Table 32. Pressure sensor features

4.5.2 Inductive proximity sensor

The proximity sensor is specially adapted and o for use with drives. The sensor boxes feature sturdy construction for use in rough environments and can withstand corrosive and dusty environments.

Cylindrical type, 4 mm diameter, metal case and short DC supply (Table 33).

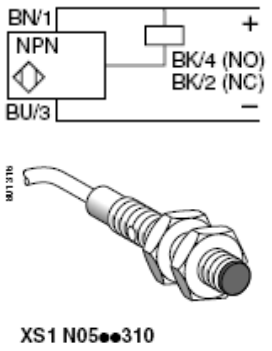
Features		
Nominal sensing distance	1 mm	
3 wire	NPN	
Connection mode	By cable	
Temperature	-40...+85ºC	
Degree protect	IP67	
Current consumption, no-load	<=10mA	
Rated supply voltage	5...24Vdc	
Maximum switching frequency	5000 Hz	
Case	Stainless stell case	

Table 33. Inductive proximity sensor features

4.6 Actuators technical characteristics

4.6.1 Electrovalve 2 way

Alcon solenoid valves are chosen for this application. It has a compact valve design, normally closed and is specially adapted and o for use with drives (Table 34).

Features 21 series		
Standard body material	Brass	
Coil Voltage DC	12V	
Standard protection	IP67	
Range operating temperature	-10...50ºC	
Electrical connection	DIN 43650	
Pipe size	¼”	
Orifice	1.6 mm	
Max. Pressure	60 bar	
Power	14.5 W	

Table 34. Electrovalve specifications

4.6.2 Butterfly valve and electrical actuator

Electric butterfly valve, available in wafer, designed to be used in the most difficult and troublesome industrial fields (Table 35).

This valve is submitted to hydrostatic, pneumatic and working test.

Features 21 series		
Actuator	Motor DC 12V	
Working angle	90º	
Supply voltage	12VDC	
Working temperature	-10...50ºC	
Maximum torque	1500Nm	
Protection	IP65	
Open/intermediate/close switch	4	
Speed control	From 30 to 60 s	

Table 35. Electric butterfly valve features.

5 Computer and hosereel communication

This part is focused in the communication between the computer and the sprinkler by GPRS (Fig 24). The program used is Labview.

On one hand the operator will be able to change the input parameters; nozzle diameter, advance speed and irrigation angle and on the other hand, the operator will be able to see in the real time the next parameters; the pressure difference between the hose and the sprinkler that determinates the lost of pressure in the pipe, flow rate and precipitations that are calculated by algorithm, the real advance speed, distance out and width irrigation. Finally, it can see different times; start time, the real time spent in irrigating the whole field and the foreseen finish time.

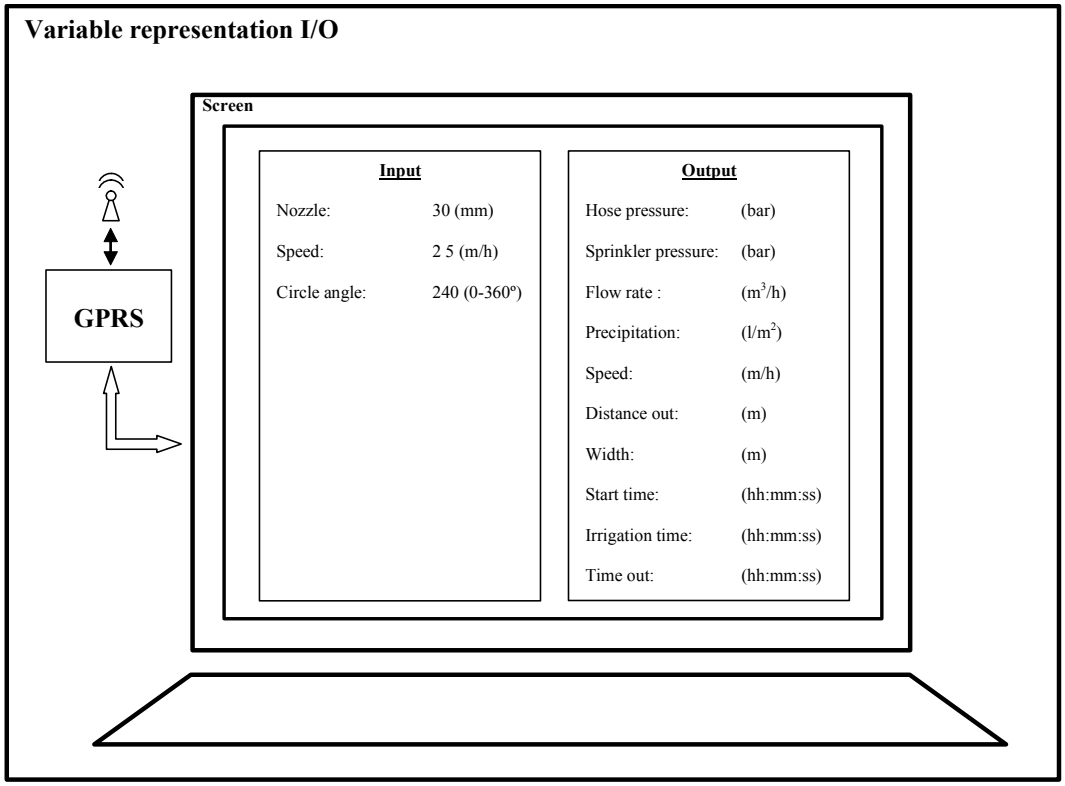


Fig 24. Inputs and outputs variables representation in the screen

6 Budget summary

The budget to make this project that include design format, make and mounted all things with all components used and ready to work. After the successful verification and after the period of practice the total import is one thousand and two hundred twenty-one centimes of euro without VAT for the electronic system controlled by DGPS.

7 Conclusions

The system designed, carry out with the initial objective of the project; create a microelectronic system controlled by DGPS that it guarantees reliability and irrigate the field without the user taking part in the process and trying to not throw away the water.

It is necessary to say that the program is robust and can go in outside without problems. The automatic system that implements in each machine will be programmable, flexible and extend if it is the circumstance needs it. This means that in the future, if the market necessities will want new applications, our system could implement it.

The main advantage that our system offers is a standard fabrication. All machines, little or big, are depending on the different values kept in the microcontroller that it means to do modifications and adequate all parameters in personal case.

Finally, the constructor is compromised to effect some modifications that it will be necessary in the mechanic part.

Albert Mach Casellas
Electronic engineer

Kraków, March 5th 2008

8 Bibliography

Cuenca, E. Martín. J.Mª Angulo Usategui, I. Angulo Martínez. Microcontroladores PIC. Madrid. Editorial Parafino. 1998.

Gijzen, Bonny. IC-Prog Prototype Programmer. (http://www.ic-prog.com, December 18th of 2007)

Gualda, J.A. S.Martínez, P.M.Martinez. Electronica industrial:Tecnicas de potencia. Madrid. Editorial marcombo. 1992.

Karpoff, Jame. Karpoff Spanish Tutor (http://welcome.to/karpoff/, November 29th of 2007)

Linone, Farnel. Farnell InOne. (http://welcome.to/karpoff/, November 15 of 2007)

Maxim Integrated Products, Inc. Dallas Semiconductor. (http://maxim-ic.com, December 10th of 2007)

MICROCHIP TECHNOLOGY INC. PIC 16F877, Datasheet 28/40/44-Pin Enhanced Flash Microcontrollers (www.microchip.com, October 23th of 2007).

NATIONAL SEMICONDUCTOR CORPORATION, LM7805.
(www.nationalsemiconductor.com, November 17th of 2007)

ANNEX A. Program Code

```
//SPRINKLER.C
#if defined(__PCH__)
#include <18F4550.h>
#fuses INTHS,NOWDt,NOPROTECT,NOLVP,NODEBUG,USBDIV,VREGEN,NOPBADEN
#use delay(clock=8000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#endif

#IF defined (__PCM__)
#rom 0x2100={1,2,3,4}
#elif defined(__PCH__)
#rom int 0xf00000={1,2,3,4}
#endif

//LIBRARIES
#include <stdlib.h>
#include <math.h>

//CONSTANTS
#define BUFFER_SIZE     64
#define DATA_IN         (ext_buffer_next_in != ext_buffer_next_out)
#define lengthh              22             //number of character that we receive from
PC
#define max_field            15             //maxium number of corners
#define max_f                10             //maxium number of intern values
#define nozzle               24             //diàmeter of nozzle
#define pressure             3              //work pressure
#define right                340            //change the direction when the sprinkler is
on the right side
#define left                 200            //change the direction when the sprinkler is
on the left side
#define clockk               16             //clockk= 32 és aprox un second
#define timee                10             //clockk= 32 és aprox un second

//VARIABLES
int sleep_mode,out,ini,inici,tmp;
signed                                                                          int16
tempa=0,pos,encoder,count,sec,waiting=0,time1=0,time2=0,min2,min1,a1=0,a2=0,aa=0;
int pulse, ant_pulse,z,i,countt=0;
char m=0,nozz, press,countini,countini1;
signed int32 xd,encoderg,ttt;
short int bit_send=FALSE, a=0,zz=0, one=0,st=FALSE,turn,fff=TRUE;


//TABLE WITH POINTS OF FIELD
signed int32 CONST x_field[max_field]={9,10,10,150,150,200,200,290,330,170,0,0,0,0,0};
signed    int32    CONST    y_field[max_field]    =    {9,-160,10,10,-30,-30,10,10,-200,-
160,0,0,0,0,0};
```

```
signed int8 length[6][6];
signed int8 f[2][2][2];

signed int32 x_f[max_f];
signed int32 y_f[max_f];

//x_f[0]; actual position
//x_f[1]; actual position - point[i] vector
//x_f[2]; actual position - point[i+1] vector
//x_f[3]; last position
//x_f[4]; initial position
//x_f[5]; main vector
//x_f[6];
//x_f[7]; long water [x]

// GLOBAL VARIABLES
int packet_buffer[BUFFER_SIZE];
int ext_buffer[BUFFER_SIZE];
int ext_buffer_next_in;
int ext_buffer_next_out;


signed int8 const table_acos[101]={
       90,     89,     89,     88,     88,     87,     87,     86,     85,     85,     84,
       84,
       83,     83,     82,     81,     81,     80,     80,     79,     79,     78,     77,
       77,     76,     76,     75,     74,     74,     73,     73,     72,     71,     71,
       70,     70,     69,     68,     68,     67,     66,     66,     65,     65,     64,
       63,     63,     62,     61,     61,     60,     59,     59,     58,     57,     57,
       56,     55,     55,     54,     53,     52,     52,     51,     50,     49,     49,
       48,     47,     46,     46,     45,     44,     43,     42,     41,     41,     40,
       39,     38,     37,     36,     35,     34,     33,     32,     31,     30,     28,
       27,     26,     25,     23,     22,     20,     18,     16,     14,     11,     8,
       0};

signed int8 const table_cos[91]={
100, 100, 100, 100, 100, 100, 99, 99, 99, 99, 98, 98,
       98,     97,     97,     97,     96,     96,     95,     95,     94,     93,     93,
       92,     91,     91,     90,     89,     88,     87,     87,     86,     85,     84,
       83,     82,     81,     80,     79,     78,     77,     75,     74,     73,     72,
       71,     69,     68,     67,     66,     64,     63,     62,     60,     59,     57,
       56,     55,     53,     52,     50,     49,     47,     45,     44,     42,     41,
       39,     38,     36,     34,     33,     31,     29,     28,     26,     24,     23,
       21,     19,     17,     16,     14,     12,     11,     9,      7,      5,      4,
       2,      0};
```

```
//TIMER1 INTERRUPT
#INT_TIMER1
void clock_isr(){

    if(--count==0){
        sec++;
                waiting++;
                tempa++;
                a=!a;
                if(zz&&(m==5||m==11)){
                        encoderg=encoderg-5;
                        turn=TRUE;
                }
                if(!zz&&(m==5||m==11)){
                        encoderg=encoderg+5;
                        turn=FALSE;
                }

                if(sec>=60){
                        sec=0;
                        min1++;
                        min2++;
                }
                count=clockk;
    }
}



// EXTERNAL INTERRUPT, KEEP FIRST POSITION
#INT_EXT
void handle_int_ext(){
        delay_ms(10);
        if(!input(PIN_B0)){
                sleep_mode=FALSE;
                m=1;
        }
}
```

```
//EXTERNAL INTERRUPT BY ENCODER      1
#INT_EXT1
void handle_int_ext1(){
        delay_ms(1);
        if(input(PIN_B1)){
                if(input(PIN_B2)){
                        pulse=1;
                }
                else{
                        pulse=0;
                }
        }
        else{
                if(input(PIN_B2)){
                        pulse=2;
                }
                else{
                        pulse=3;
                }
        }
        if(m==0){
                sleep_mode=FALSE;
                m=4;
        }
}

//EXTERNAL INTERRUPT BY ENCODER      2
#INT_EXT2
void handle_int_ext2(){
        delay_ms(1);
        if(input(PIN_B2)){
                if(input(PIN_B1)){
                        pulse=1;
                }
                else{
                        pulse=2;
                }
        }
        else{
                if(input(PIN_B1)){
                        pulse=0;
                }
                else{
                        pulse=3;

                }
        }
        if(m==0){
                sleep_mode=FALSE;
                m=4;
        }
}
```

```
//SERIAL INTERRUPT
#INT_RDA
void serial_isr()
{
   ext_buffer[ext_buffer_next_in] = getc();     // get a byte, put it in buffer

   if(++ext_buffer_next_in == BUFFER_SIZE)      // increment counter
      ext_buffer_next_in = 0;
}

// GET_BUFF_INT
// function to extract bytes from the buffer
int get_buff_int()
{
   int retval;

   while(!DATA_IN);                             // wait until data available

   retval = ext_buffer[ext_buffer_next_out];    // get the byte
   if(++ext_buffer_next_out == BUFFER_SIZE)     // increment counter
      ext_buffer_next_out = 0;

   return retval;
}

// SEND_PACKET
// function to send a packet of data to another PIC
void send_packet(int* packet_ptr, int16 packet_length)
{
       char i;

   for(i=0; i<packet_length; i++)               // send packet
      putc(packet_ptr[i]);
}

// GET_PACKET
// function to get a packet from the buffer and read the data
short get_packet(int* packet_ptr)
{
   short retval;
   int16 i;

   retval = TRUE;

   for(i=0; i<22; i++)                  // get the data
      packet_ptr[i] = get_buff_int();


   return retval;
}
```

```
void take_pos(){
       int16 ii,jj;


       if(!DATA_IN)                                // if no data in
       continue;
                                                     // loop back
       if(get_packet(packet_buffer))               // if valid packet, return retval
    {

              if(packet_buffer[3]=='1'){encoderg=404;st=TRUE;}
              else if(packet_buffer[5]=='1'){encoderg=464;st=TRUE;}
              else if(packet_buffer[4]=='1'){encoderg=0;st=TRUE;}


              x_f[3]=x_f[0];
              y_f[3]=y_f[0];
              x_f[0]=0;
              y_f[0]=0;
              ii=0;
              jj=0;
              for(ii=8; ii<13; ii++){                  // get the data
                     pos= packet_buffer[ii]-48;
                     for(jj=0; jj<(12-ii); jj++){
                            pos=pos*10;
                     }
                     x_f[0]=x_f[0]+pos;
              }
              if(packet_buffer[7]=='-'){x_f[0]=-x_f[0];}
              ii=0;
              jj=0;
              for(ii=14; ii<(19); ii++){                   // get the data
              pos= packet_buffer[ii]-48;
                     for(jj=0; jj<(18-ii); jj++){
                            pos=pos*10;
                     }
                     y_f[0]=y_f[0]+pos;
              }
              if(packet_buffer[13]=='-'){y_f[0]=-y_f[0];}

       }
       bit_send=TRUE;
       delay_ms(10);

       if(bit_send)                              // if button pushed
    {
        packet_buffer[0] = 'G';
        packet_buffer[1] = 'P';
        packet_buffer[2] = 'S';
        packet_buffer[3] = ' ';
        packet_buffer[4] = ' ';
        packet_buffer[5] = ' ';
```

```
         packet_buffer[6] = ' ';
            send_packet(packet_buffer,lengthh);        // send message

               bit_send=FALSE;
        }
}


//CALCULATE POSITION OF ENCODER
void calc_encoder(){
        int32 bb;
        float llavor;
        int8 i;

        xd=(int32)length[nozz][press];

        if(encoderg>512){encoderg=0;}
        else if(encoderg<=0){encoderg=512;}

        encoder=encoderg*7/10;

        if((encoder>=0)&&(encoder<90)){
               y_f[7]=xd*(int32)table_cos[encoder]/100;
               x_f[7]=xd*(int32)table_cos[90-encoder]/100;
        }
        else if((encoder>=90)&&(encoder<180)){
               y_f[7]=-1*xd*(int32)table_cos[180-encoder]/100;
               x_f[7]=xd*(int32)table_cos[encoder-90]/100;
        }
        else if((encoder>=180)&&(encoder<270)){
               y_f[7]=-1*xd*(int32)table_cos[encoder-180]/100;
               x_f[7]=-1*xd*(int32)table_cos[270-encoder]/100;
        }
        else{
               y_f[7]=xd*(int32)table_cos[360-encoder]/100;
               x_f[7]=-1*xd*(int32)table_cos[encoder-270]/100;

        }
        //director vector

        x_f[5]=x_f[0]-x_f[4];
        y_f[5]=y_f[0]-y_f[4];

        x_f[6]=y_f[5];
        y_f[6]=-1*x_f[5];

        bb=abs(x_f[5]*x_f[5])+abs(y_f[5]*y_f[5]);

        i=0;
        llavor=45.0;

        for(i=0; i<7; i++){
```

```
                llavor=(bb/(2*llavor))+(llavor/2);

        }
        bb=(int32)llavor;

        x_f[8]=(x_f[5]*x_f[7]+y_f[5]*y_f[7])/bb + x_f[0];
        y_f[8]=(x_f[6]*x_f[7]+y_f[6]*y_f[7])/bb + y_f[0];

}

//CALCULATE POSITION
void calc_position(){
        float  llavor;
        int32 aa, aaa,ddd, bb, cc, dd, angle;
        int16 sum_ang=0;
        int8 i,j,cos_angle;


        j=0;
        for(j=1; j<=x_field[0]; j++){
                x_f[1]=x_field[j]-x_f[8];
                y_f[1]=y_field[j]-y_f[8];
                if(j==x_field[0]){
                        x_f[2]=x_field[1]-x_f[8];
                        y_f[2]=y_field[1]-y_f[8];
                }
                else{
                        x_f[2]=x_field[j+1]-x_f[8];
                        y_f[2]=y_field[j+1]-y_f[8];
                }

                aa=(x_f[1]*x_f[2]) + (y_f[1]*y_f[2]);
                aaa=abs((x_f[1]*x_f[2]) + (y_f[1]*y_f[2]));
                dd=(x_f[1]*y_f[2]) - (x_f[2]*y_f[1]);
                ddd=abs((x_f[1]*y_f[2]) - (x_f[2]*y_f[1]));
                bb=abs(x_f[1]*x_f[1])+abs(y_f[1]*y_f[1]);
                cc=abs(x_f[2]*x_f[2])+abs(y_f[2]*y_f[2]);

                i=0;
                llavor=45.0;

                for(i=0; i<7; i++){
                        llavor=(bb/(2*llavor))+(llavor/2);

                }
                bb=(int32)llavor;

                i=0;
                llavor=45.0;
                for(i=0; i<7; i++){
                        llavor=(cc/(2*llavor))+(llavor/2);
                }
```

```
                cc=(int32)llavor;

                angle=((100*aaa)/(bb*cc));
                if(angle>100){angle=100;}
                cos_angle=table_acos[(char)angle];

                if(aa!=aaa){
                        cos_angle=180-cos_angle;
                }

                if(dd==ddd){
                        sum_ang=sum_ang - (int16)cos_angle;
                }

                else{
                        sum_ang=sum_ang + (int16)cos_angle;

                }
        }

        sum_ang=sum_ang+10;
        if((sum_ang>190)){
                out=FALSE;
        }
        else{
                out=TRUE;

        }
}

//CALCULATE THE TIME THAT IT HAS IRRIGATE
//IN ONE BAND AND OTHER BAND
void calc_area(){
        if(turn){
                a1=tempa;
        }
        else{
                a2=tempa;
        }
        aa=a1+a2;

        if((a1!=0)&&(a2!=0)){
                tmp=TRUE;
                time1=timee*a1*10;
                time1=time1/10/aa;
                time2=timee-time1;
        }
        else{
                tmp=FALSE;
        }
}
```

```
//TABLE OF LONG OF WATER, PRESSURE AND NOZZLE.
void full_table(){
int8 i;

length[0][0]=33;length[0][1]=39;length[0][2]=44;length[0][3]=48;length[0][4]=48;length[0
][5]=48;
length[1][0]=37;length[1][1]=41;length[1][2]=46;length[1][3]=51;length[1][4]=51;length[1
][5]=51;
length[2][0]=44;length[2][1]=44;length[2][2]=48;length[2][3]=52;length[2][4]=55;length[2
][5]=55;
length[3][0]=46;length[3][1]=46;length[3][2]=50;length[3][3]=54;length[3][4]=57;length[3
][5]=57;
length[4][0]=52;length[4][1]=52;length[4][2]=52;length[4][3]=56;length[4][4]=59;length[4
][5]=62;
length[5][0]=53;length[5][1]=53;length[5][2]=53;length[5][3]=57;length[5][4]=61;length[5
][5]=64;

}


// COUNTER OF ENCODER
void count_encoder(){
        switch(pulse){
                case 0:
                        if(ant_pulse==3){
                                encoderg++;
                                ant_pulse=pulse;
                        }
                        else if (ant_pulse==1){
                                encoderg--;
                                ant_pulse=pulse;
                        }
                        break;
                case 1:
                        if(ant_pulse==0){
                                encoderg++;
                                ant_pulse=pulse;
                        }
                        else if (ant_pulse==2){
                                encoderg--;
                                ant_pulse=pulse;
                        }
                        break;
                case 2:
                        if(ant_pulse==1){
                                encoderg++;
                                ant_pulse=pulse;
                        }
                        else if (ant_pulse==3){
                                encoderg--;
                                ant_pulse=pulse;
                        }
```

```
                        break;
                case 3:
                        if(ant_pulse==2){
                                encoderg++;
                                ant_pulse=pulse;
                        }
                        else if (ant_pulse==0){
                                encoderg--;
                                ant_pulse=pulse;
                        }
                        break;
        }
}


void sprinkler(){
        switch(nozzle){
                case 24:
                        nozz=0;
                case 26:
                        nozz=1;
                case 28:
                        nozz=2;
                case 30:
                        nozz=3;
                case 32:
                        nozz=4;
                case 34:
                        nozz=5;
        }

        switch(pressure){
                case 2:
                        press=0;
                case 3:
                        press=1;
                case 4:
                        press=2;
                case 5:
                        press=3;
                case 6:
                        press=4;
                case 7:
                        press=5;
        }
}

//SEND SOME INFORMATION BY RS232
 void comunicat(){
        float llavor;
        int32 bb;
        delay_ms(10);
```

```
   bit_send=TRUE;
   if(bit_send)                          // if button pushed
{
          llavor=fmod(encoder,10);
          packet_buffer[2] = 48+(int8)llavor;

          bb=encoder/10;
          llavor=fmod(bb,10);
          packet_buffer[1] = 48+(int8)llavor;

          bb=bb/10;
          llavor=fmod(bb,10);
          packet_buffer[0] = 48+(int8)llavor;

 //    packet_buffer[0] = 'P';
   //  packet_buffer[1] = 'O';
     //packet_buffer[2] = 'S';
     packet_buffer[3] = ' ';


          if(out){

          packet_buffer[4] = 'O';
                 packet_buffer[5] = 'U';
          packet_buffer[6] = 'T';

          }
          else{
          packet_buffer[4] = 'I';
                 packet_buffer[5] = 'N';
          packet_buffer[6] = ' ';
          }


          if(x_f[8]<0){
                 packet_buffer[7] = '-';
                 x_f[9]=-1*x_f[8];
          }
          else{
                 packet_buffer[7] = '+';
                 x_f[9]=x_f[8];
          }
          if(y_f[8]<0){
                 packet_buffer[13] = '-';
                 y_f[9]=-1*y_f[8];
          }
          else{
                 packet_buffer[13] = '+';
                 y_f[9]=y_f[8];
          }
```

```
            //packet_buffer[7] = ' ';

            llavor=fmod(x_f[9],10);
            packet_buffer[12] = 48+(int8)llavor;

            bb=x_f[9]/10;
            llavor=fmod(bb,10);
            packet_buffer[11] = 48+(int8)llavor;

            bb=bb/10;
            llavor=fmod(bb,10);
            packet_buffer[10] = 48+(int8)llavor;

            bb=bb/10;
            llavor=fmod(bb,10);
            packet_buffer[9] = 48+(int8)llavor;

            bb=bb/10;
            packet_buffer[8] = 48+(int8)bb;



            llavor=fmod(y_f[9],10);
            packet_buffer[18] = 48+(int8)llavor;

            bb=y_f[9]/10;
            llavor=fmod(bb,10);
            packet_buffer[17] = 48+(int8)llavor;

            bb=bb/10;
            llavor=fmod(bb,10);
            packet_buffer[16] = 48+(int8)llavor;

            bb=bb/10;
            llavor=fmod(bb,10);
            packet_buffer[15] = 48+(int8)llavor;

            bb=bb/10;
            packet_buffer[14] = 48+(int8)bb;
            packet_buffer[5] = 48+(int8)time1;
            packet_buffer[6] = 48+(int8)time2;
            packet_buffer[20] = 48+(int8)min1;

            send_packet(packet_buffer,lengthh);        // send message
          bit_send=FALSE;
        }

}
```

```
void main() {
        setup_oscillator(OSC_8MHZ);
        ext_buffer_next_in = 0;                         // init variables
        ext_buffer_next_out = 0;

        sleep_mode=TRUE;         // init sleep flag

        setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
        set_timer1(0);

        enable_interrupts(GLOBAL);
        enable_interrupts(int_rda);
        enable_interrupts(INT_TIMER1);
        enable_interrupts(int_ext);
        enable_interrupts(int_ext1);
        enable_interrupts(int_ext2);

        SET_TRIS_A( 0xFF );
        SET_TRIS_B( 0xFF );
        SET_TRIS_D( 0x00 );
        SET_TRIS_E( 0x00 );

        count=clockk;
        pulse=0;
        x_f[3]=0;
        y_f[3]=0;
        ant_pulse=0;
        countini=0;
        countini1=0;
        sec=0;
        min1=0;
        min2=0;
        z=0;
        ini=TRUE;
        encoderg=384;
        output_d(0);
        inici=TRUE;
        out=FALSE;
        tmp=TRUE;

        sprinkler();
        full_table();

    while(1) {
        count_encoder();

        if(sleep_mode){
                //sleep();
        }
        switch(m){

        case 1:                      //take the first position
```

```
        output_d(0x10);
        take_pos();


        x_f[4]=x_f[0];
        y_f[4]=y_f[0];

        x_f[3]=x_f[3]>>4;
        x_f[0]=x_f[0]>>4;
        y_f[3]=y_f[3]>>4;
        y_f[0]=y_f[0]>>4;


        if((x_f[3]==x_f[0])&&(y_f[3]==y_f[0])){
                m=1;
                i=0;
                for(i=0; i<4; i++){
                        write_eeprom((3-i), x_f[4]>>(i*8));
                        delay_ms(10);
                }
                for(i=0; i<4; i++){
                        write_eeprom((7-i), y_f[4]>>(i*8));
                        delay_ms(10);
                }
        }
        else{
                m=2;
        }
        x_f[0]=x_f[0]<<4;
        y_f[0]=y_f[0]<<4;

        break;


case 2:                         //take a second position
        output_d(0x20);
        take_pos();

        x_f[5]=x_f[3];
        y_f[5]=y_f[3];

        x_f[3]=x_f[3]>>4;
        x_f[0]=x_f[0]>>4;
        y_f[3]=y_f[3]>>4;
        y_f[0]=y_f[0]>>4;

        if((x_f[3]==x_f[0])&&(y_f[3]==y_f[0])){
                sleep_mode=TRUE;
                m=0;
        }
        else{
```

```
                m=2;
        }

        x_f[0]=x_f[0]<<4;
        y_f[0]=y_f[0]<<4;
        break;

case 4:
        x_f[4]=0;
        y_f[4]=0;
        i=0;

        for(i=0; i<4; i++){
                ttt=(int32)(read_eeprom(3-i));
                ttt=ttt<<(8*i);
                x_f[4]=x_f[4]+ttt;
        }
        for(i=0; i<4; i++){
                ttt=(int32)(read_eeprom(7-i));
                ttt=ttt<<(8*i);
                y_f[4]=y_f[4]+ttt;
        }
        m=5;
        min2=0;
        break;

case 5:
        output_d(0x50);
        take_pos();

        //if(!st){break;}

        if(x_f[0]==150&&y_f[0]==-40&&fff){
                        ini=TRUE;
                        fff=FALSE;
        }
        calc_encoder();
        calc_position();
        comunicat();

        if((OUT)&&(ini)){
                m=7;
                out=FALSE;
                countini++;
                countini1=0;
                calc_area();
                countt=0;
                if(countini==2){
                        ini=FALSE;
                        countini=0;
                }
        }
```

```
else if((OUT)&&(!ini)&&(tmp)){
        m=7;
        countt=0;
}
else if((OUT)&&(!ini)&&(!tmp)){
        m=7;
        countt=0;
}
else if(((encoder<right)&&(encoder>left))&&ini){
        tempa=0;
        min1=0;
        countini1++;
        if(one){
                if(countini1>=2){
                        m=7;
                        ini=FALSE;
                }
                else{
                        m=5;
                }
        }
        one=FALSE;
}
else if(((encoder<right)&&(encoder>left))&&!ini){
        tempa=0;
        countt++;
        if(one){
                m=7;
                if(countt>=3){
                        count=10;
                        tmp=FALSE;
                        time1=0;
                        time2=0;
                        min1=0;
                }
                if(tmp){
                        if((encoder>=90)&&(encoder<270)){
                                if(min1>time1){
                                        m=5;
                                        min1=0;
                                }
                        }
                        else{
                                if(min1>time2){
                                        m=5;
                                        min1=0;
                                        countini1++;
                                }
                        }
                }
        }
        one=FALSE;
```

```
                }
                else{
                        m=5;
                        one=TRUE;
                }
                break;

        case 6: //shorter
                m=5;
                break;

        case 7: //keep position
                m=10;
                break;

        case 8: //large
                m=5;
                break;

        case 9://wait
                m=10;
                break;

        case 10: //change way
                m=11;
                zz=!zz;
                waiting=0;
                break;

        case 11:
                output_d(0xb0);
                take_pos();
                calc_encoder();
                comunicat();
                if(waiting<7){m=11;}
                else{m=5;}
                out=FALSE;
                break;

        default:
                m=0;
                take_pos();
                output_d(0x00);
                break;

        }
        delay_ms(50);
 }
}
```