**FOX Program**

The Fox program in splited in 2 programs – Interrupt and Main Program. Each of this Program, has multiple Functions. Interrupt : Reset, Mode ; Main: Init, Type, ConverToMorse, CW\_Keying, Send\_Word;

The Interrupt is in charge with the Timing and User Interface (RS232 data). What is the Interrupt? The interrupt is a Sub Program that is accessed when an event occurs. In this program we have configured 2 events:

1. When the Timer 1 overflows
2. USART Receive, when the data is received on the serial (RS232)

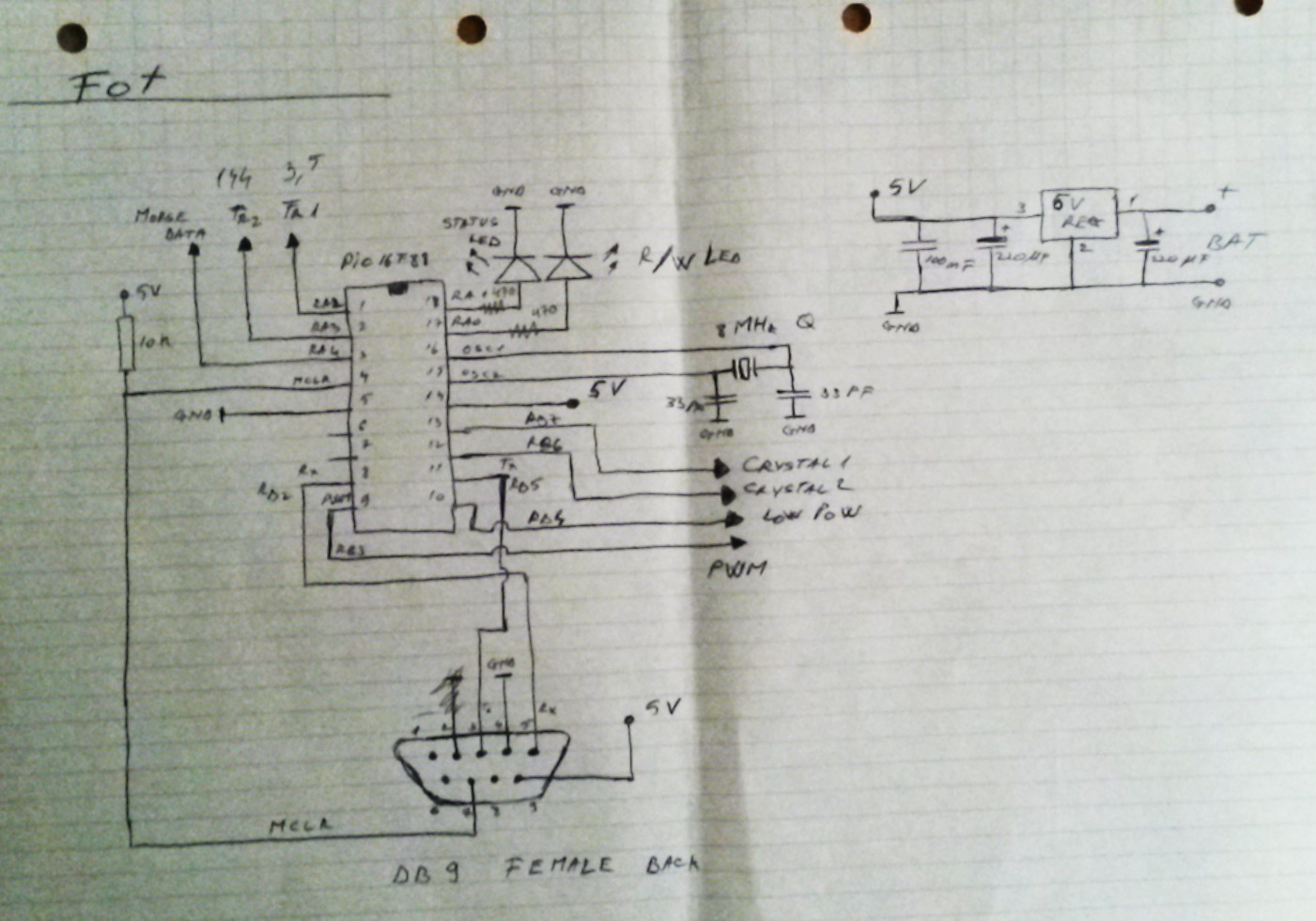
The Main program has the job to send the Morse data according with the Type Selected.

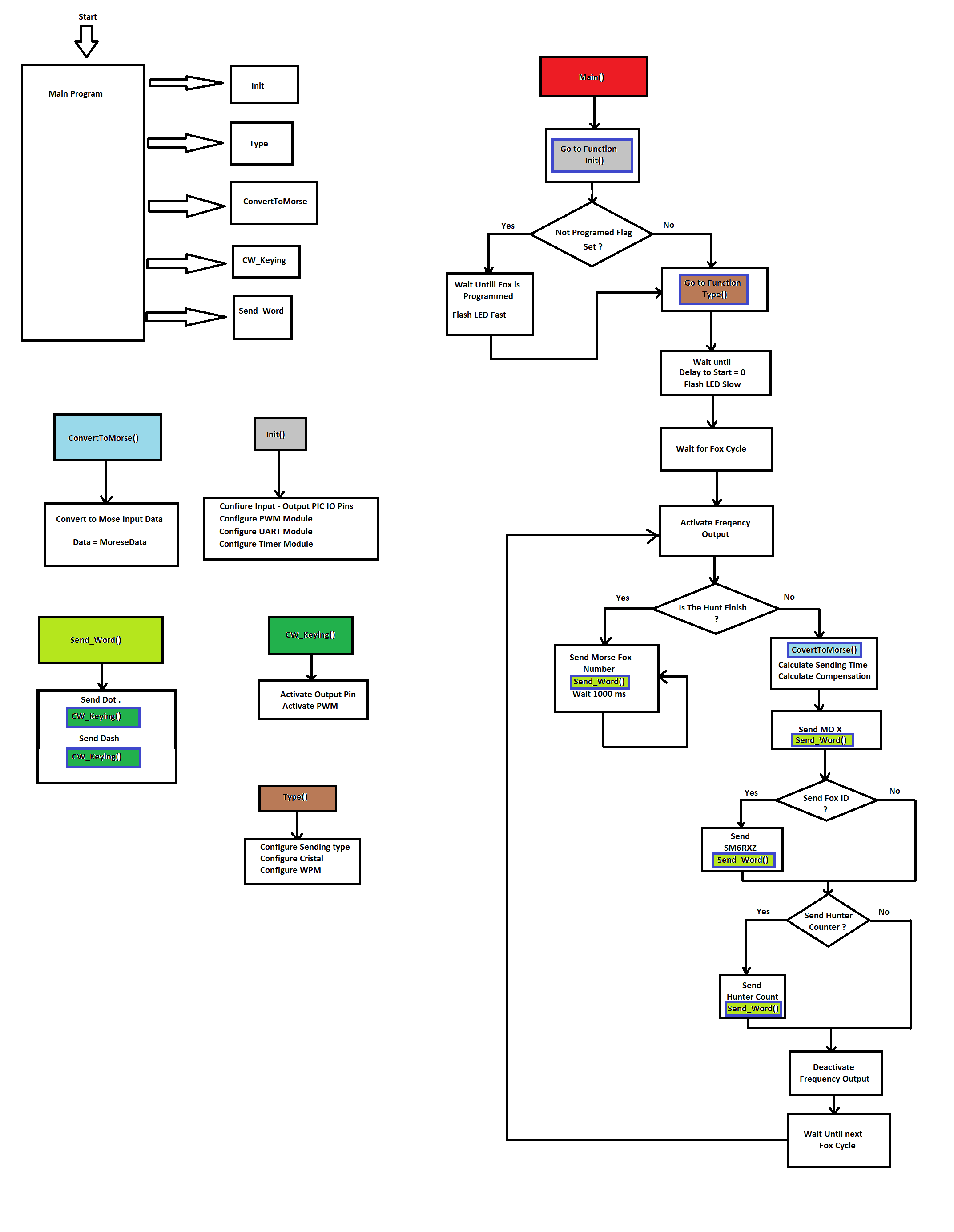
**Document information :**

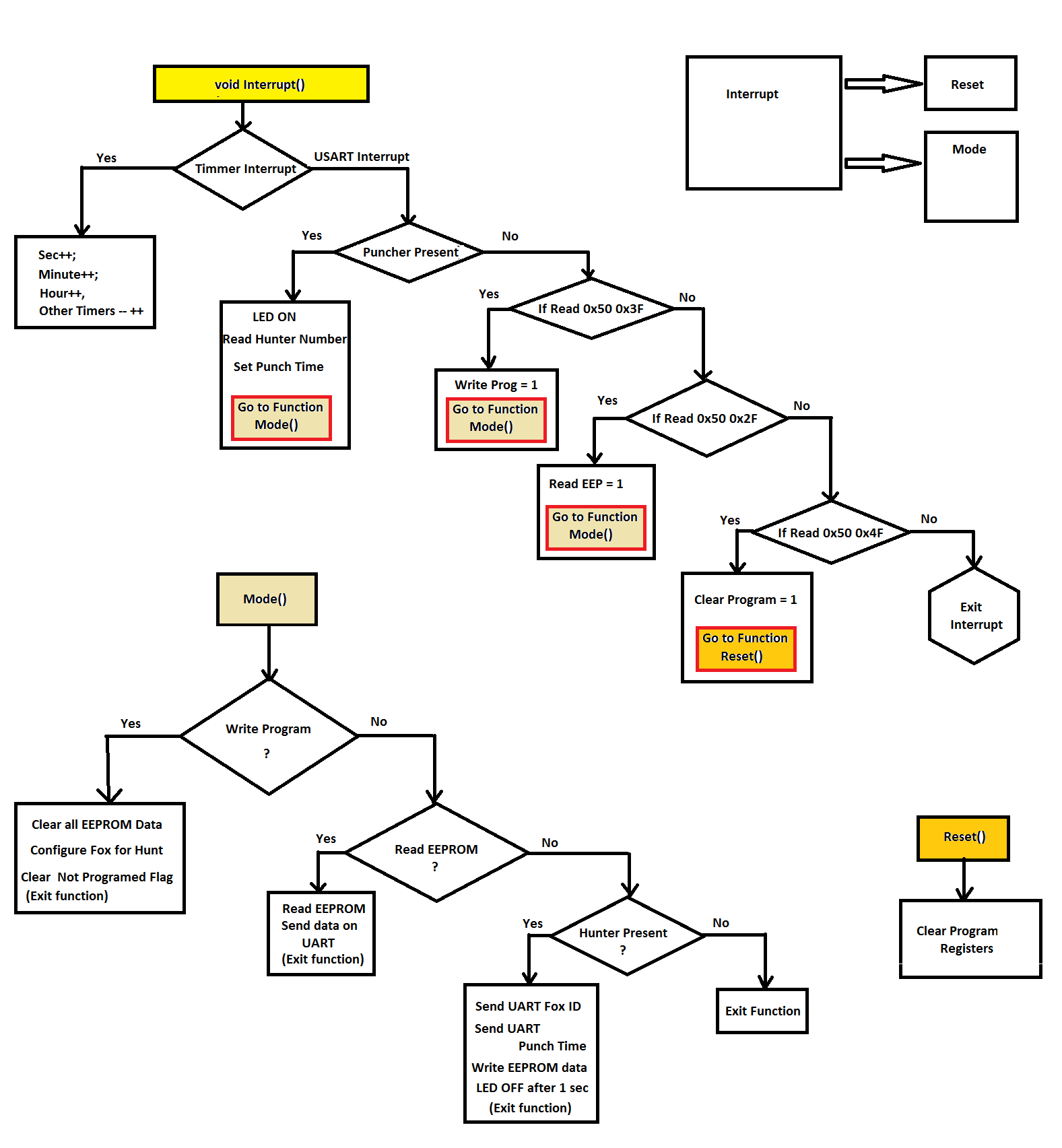
**Black : SW description**

**Red : SW CODE**

**Green : SW CODE Comments**

**Schematic and SW Block diagram:**





**Define the Global Variables**

The Program needs the variables to store the data during the run of the Program. Variables are reinitialized during each start. Some of them are just with a name, other have the initial value 0 and other have special value. For example, each Fox the “Hard coded” number, this number cannot be change, only by changing the C program.

/\* ---- Project Desription -----

Project Name = FoxHunt

Micro used = PIC16F88

Oscilator = 8 MHz

=================================== SNIP ========================================================

MORSE ENCODING...

One morse character per BYTE, bitwise, LSB to MSB. 0 = dit, 1 = dah. The byte is shifted to the right bit by bit, until the last 1 is left, this 1 is an END OF CHARACTER indicator. A maximum of 7 elements can be encoded, (error) is excluded.

KN -.--. 00101101 EOM 11111111 U ..- 00001100

SK ...-.- 01101000 SPACE 00000000 V ...- 00011000

AR .-.-. 00101010 D -.. 00001001 W .-- 00001110

BT -...- 00110001 E . 00000010 X -..- 00011001

AS .-... 00100010 F ..-. 00010100 Y -.-- 00011101

/ -..-. 00101001 G --. 00001011 Z --.. 00010011

0 ----- 00111111 H .... 00010000

1 .---- 00111110 I .. 00000100 Fox 1 . 00000010

2 ..--- 00111100 J .--- 00011110 Fox 2 .. 00000100

3 ...-- 00111000 K -.- 00001101 Fox 3 ... 00001000

4 ....- 00110000 L .-.. 00010010 Fox 4 .... 00010000

5 ..... 00100000 M -- 00000111 Fox 5 ..... 00100000

6 -.... 00100001 N -. 00000101 Fox 6 -. 00000101

7 --... 00100011 O --- 00001111 Fox 7 -.. 00001001

8 ---.. 00100111 P .--. 00010110 Fox 8 -... 00010001

9 ----. 00101111 Q --.- 00011011 Fox 9 -.... 00100001

A .- 00000110 R .-. 00001010 Fox 10 -..... 01000001

B -... 00010001 S ... 00001000

C -.-. 00010101 T - 00000011

============================ SNIP ===============================================================\*/

**Data types:**

In the Program the values can be :

Bin: 0b00001111

Dec: 0 - 9

Hex: 0x00 – 0xFF

**Define type length:**

If the variable will not exceed the value 255, then we will define the value as *Char* because will use just 1 byte from the RAM memory. If the value is bigger, than we use *Int*, this will have 2 byest and *Long Int* for 4 bytes.

| **Type** | **Size in bytes** | **Range** |
| --- | --- | --- |
| (unsigned) char | 1 | 0 .. 255 |
| signed char | 1 | - 128 .. 127 |
| (signed) short (int) | 1 | - 128 .. 127 |
| unsigned short (int) | 1 | 0 .. 255 |
| (signed) int | 2 | -32768 .. 32767 |
| unsigned (int) | 2 | 0 .. 65535 |
| (signed) long (int) | 4 | -2147483648 .. 2147483647 |
| unsigned long (int) | 4 | 0 .. 4294967295 |

char Message\_to\_Send\_Wrd\_1 = 0b00000111; // M

char Message\_to\_Send\_Wrd\_2 = 0b00001111; // 0

//...............................................................................................................................................................

char Message\_to\_Send\_Wrd\_3 = 0b00000010; // Fox Number 1

char data\_lgt\_fox = 4; // Length of the Morse Fox Count 4

char FoxHuntNumber = 1; // Fox Hunt Number - max foxes 254

//................................................................................................................................................................

char Message\_to\_Send\_Wrd\_4 = 0b00001000; // S Fox ID 8

char Message\_to\_Send\_Wrd\_5 = 0b00000111; // M Fox ID 10

char Message\_to\_Send\_Wrd\_6 = 0b00100001; // 6 Fox ID 14

char Message\_to\_Send\_Wrd\_7 = 0b00001010; // R Fox ID 10

char Message\_to\_Send\_Wrd\_8 = 0b00011001; // X Fox ID 14

char Message\_to\_Send\_Wrd\_9 = 0b00010011; // Z Fox ID 14

char Message\_to\_Send\_FoxHunters\_1 = 0x00;

char Message\_to\_Send\_FoxHunters\_2 = 0x00;

char data\_lgt = 0;

char Delay\_to\_Start = 0; // Min\*10

char Max\_Time = 10; // Min\*10

int Delay\_to\_Start\_Led = 0;

int Max\_Time\_Var = 100;

char Hunt\_Type = 1; // Hunt type (1 - 15)

char Frequency = 3; // Frequency (3,5 ; 144 ; both) 3 = 3,5 144 = 144 255 = both

//Speed of CW 5 WPM 1200/5 = 240 ms Based upon a 50 dot duration standard word such as PARIS

unsigned WPM\_data = 5; // WPM sending normal data MOx

unsigned WPM\_data\_ID = 20; // WPM sending ID

char FoxHunters\_Enable = 0; // Enable Sending FoxHunters number

char FoxSndID\_Enable = 1; // Enable Sending Fox ID

unsigned Foxes = 5; // Number of Foxes

unsigned Cycle = 5; // Min of a Cycle

unsigned ms\_FoxCycle = 0; // sec of a ms\_FoxCycle

unsigned s\_FoxCycle = 0; // sec of a s\_FoxCycle

char MinuteToEnd = 0;

char Time = 0;

char Sec = 0;

char Minute = 0;

char Hour = 0;

char SecToTx = 61;

char Program\_Flg = 0xFF; // Programming Flag

//char Sync\_Flg = 0;

char Programming\_Fox = 0;

char Programming\_Fox\_Data[7];

char Programming\_Fox\_Cnt = 0;

char ClearParam =0;

char HunterNumer = 0;

char HunterSec = 0;

char HunterMinute = 0;

char HunterHour = 0;

char FoxFinderNumer = 0;

int CW\_Speed\_1 = 0;

int CW\_Speed\_2 = 0;

char EEP\_data = 0;

char UART\_Read\_Data = 0;

char EEP\_Data\_Read\_Write = 0;

int i=0;

int j=0;

int TimeToEnd=0;

int TimeDev = 0;

int TimeToSend = 0;

char TimeHntNr\_1 = 0;

char TimeHntNr\_2 = 0;

int Wait\_Time = 0;

int s\_End\_Wait\_Time = 0;

int cycles\_MOx=0;

int cycle\_Corection=0;

int WPM\_Sliding=0;

int MOx\_Length=0;

int ms\_TimSnd\_MOx=0;

char HuntersIDs[60];

char HuntersIDs\_Flag = 0;

1. **Interrupt**

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////-- Interrupt--/////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Before each main part of the program, Interrupt or Main, we must build our Functions. In the interrupt part, as show in the Block diagram, we have the Reset Function and Mode Function.

**Reset Function**

Reset Function has the job to reset the microcontroller, the same as a power On. It is used if you, by mistake, write a wrong “Hunt Program”. It is necessary to reset the unit to have a fresh start.

// ---------------Reset Function -----------------------------------------------

void Reset() // Declaration of the function

{

Programming\_Fox=0; // Set Programming Flag to 0

PCLATH = 0x00; // Clear Microcontroller registers

PCL = 0x00;

}

**Mode Function**

Mode Function has 3 subfunctions:

* Write the “Hunt Program”
* Read the Fox EEPROM
* Send the Fox Number and the Time to the *Puncher* (Each participant has his own Device to store the Time when he catch a Fox)

// ------------- Serial Read Write Mode Function ---------------------------------------------------------------------------------

void Mode() Function Declaration

{

// ----- Program / Read / Erase Fox ---

First is check if the User is sending a Diagnostic command. The flag *Programming\_Fox* will have the value 1 if Yes or 0 if No.

if(Programming\_Fox == 1) // ------- Program Fox ---

{

If the Diagnostic command is send, then is checking if the SubFunction form Diagnostic command is for Write the “Hunt Program”. The variable *EEP\_Data\_Read\_Write* will be 0 if no Read or Write command was sent, 1 for Read and 2 for Write.

if (EEP\_Data\_Read\_Write == 2)

{

If the SubFunction from the Diagnostic command was for writing the “Hunt Program”, then we start to record in the array *Programming\_Fox\_Data* the rest of the 5 bytes send on the Serial port (USART). Data received should have the follow order: *Delay\_to\_Start, Max\_Time, Hunt\_Type, Frequency, Program\_Flg.*

Programming\_Fox\_Cnt++;

Programming\_Fox\_Data[Programming\_Fox\_Cnt] = UART\_Read\_Data ;

After receiving the first character on the serial, we must check if the Fox was already written or not, by verifying the variable *Program\_Flg*, if is different than 0xFF. If Yes, than we set the receiving data counter *Programming\_Fox\_Cnt* to 5 and put the LED OFF (“0” logic) from the Pin 0 from port A.

if(Program\_Flg != 0xFF) Programming\_Fox\_Cnt = 5, PORTA.F0 = 0;

If the receiving data counter is 5 or greater, we enter in the next If statement.

if(Programming\_Fox\_Cnt >= 5)

{

Here is check again if the Fox was written before, to not overwrite the existing Hunt Program or erasing the EEPROM.

if(Program\_Flg == 0xFF)

{

If the Fox was not written before and all the data was received, we start to Program the Fox .

Stop first all the interrupts by setting in the Register *INTCON* the bit *GIE* to 0 (see PIC Datasheet for more details).

INTCON.GIE = 0;

Erase the EEPROM Locations from 1 to 249. To have an empty EEPROM for the next hunt.

for(i=1;i<250;i++) // Erase EEPROM

{

EEPROM Write function, write on position i the value 0x00.

EEPROM\_Write(i,0x00);

Wait 10 ms between each EEPROM Write.

delay\_ms(10);

}

Write on Position 0 in the EEPROM the Fox Number.

EEPROM\_Write(0, FoxNumber);

delay\_ms(10);

Start the Interrupt functions after EEPROM access. This Procedure is necessary to not affect the EEPROM Write procedure.

INTCON.GIE = 1;

Assign to each variable the data receive from USART.

Delay\_to\_Start = Programming\_Fox\_Data[1]; // Scaling 10 Min

Max\_Time = Programming\_Fox\_Data[2]; // Scaling 10 Min

Hunt\_Type = Programming\_Fox\_Data[3]; // Hunt type (1 - 15)

Frequency = Programming\_Fox\_Data[4]; // Frequency (3,5 ; 144 ; both) 3 = 3,5 14 = 144 25 = both

Program\_Flg = Programming\_Fox\_Data[5]; // Fox Programming state (0xFF not programmed, Different than 0xFF Programmed)

For the Delay to start time and Hunt Time, the values must be multiply by 10 because this is the scaling for the time to have a bigger range. Maximum will be 2550 minutes => 42,5 hours

Max\_Time\_Var = Max\_Time\*10;

Delay\_to\_Start\_Led = Delay\_to\_Start\*10;

Clear the Puncher History. Who punch the Fox will be erase.

for(i=0;i<60;i++) HuntersIDs[i] = 0 ;

Set all the times 0, except the sec. The Hunt Programming takes 3 sec and to synchronize the starting time at the exact time we push the Button for programming.

Time = 0;

Sec = 3;

Minute = 0;

Hour = 0;

}

Set the 3 variables to 0 to exit the Hunt Programming action.

Programming\_Fox\_Cnt = 0;

EEP\_Data\_Read\_Write = 0;

Programming\_Fox = 0;

}

}

// ------- Read all EEP data ---

If the SubFunction form Diagnostic command is for Read the Fox EEPROM, we send all the data from EEPROM on the USART port.

else if (EEP\_Data\_Read\_Write == 1)

{

Disable the Interrupts.

INTCON.GIE = 0;

Send on USART first the Fox Number.

Usart\_Write(FoxNumber);

delay\_ms(10);

Than read EEPROM from position 1 to 240 and send it on the USART port.

for(i=1;i<241;i++)

{

EEP\_data = EEPROM\_Read(i);

Usart\_Write(EEP\_data);

delay\_ms(10);

}

Activate back the Interrupts.

INTCON.GIE = 1;

Set the variables to 0 to exit the Read action and also set the LED OFF.

EEP\_Data\_Read\_Write = 0;

Programming\_Fox = 0;

PORTA.F0 = 0;

} // End Read

}

// ----- Send to Puncher the Time ---

If there was no Diagnostic request and the Received data on the USART Port is < than 0x3D, this mean that is a Hunter that Punch the Fox.

else if(HunterNumer<0x3D) //(Hunter\_Present == 1)&&

{

Check if the Hunter punched before the Fox.

for(i=0;i<60;i++) // Check if the hunter punch the same fox

{

if (HunterNumer == HuntersIDs[i]) HuntersIDs\_Flag = 1 ;

}

If is the second time when the Hunter is punching the Fox,

if(HuntersIDs\_Flag == 1)

{

send dummy data.

Usart\_Write(0x60); // Write dummy data

Usart\_Write(0x00);

Usart\_Write(0x00);

Usart\_Write(0x00);

HuntersIDs\_Flag = 0;

}

else, if is the first time,

else

{

send to the Punch the Fox number and the Fox Time.

Usart\_Write(FoxNumber); // and send data via UART

Usart\_Write(HunterHour); // and send data via UART

Usart\_Write(HunterMinute); // and send data via UART

Usart\_Write(HunterSec); // and send data via UART

Write in the variable array the Punch number, to have a record if he punch again.

HuntersIDs[FoxFinderNumer] = HunterNumer;

Disable the Interrupts and Write in the EEPROM the Punch Number and the Time From The Fox.

INTCON.GIE = 0;

EEPROM\_Write(((4\*FoxFinderNumer)+1),HunterNumer), delay\_ms(1);

EEPROM\_Write(((4\*FoxFinderNumer)+2),HunterHour), delay\_ms(1);

EEPROM\_Write(((4\*FoxFinderNumer)+3),HunterMinute),delay\_ms(1);

EEPROM\_Write(((4\*FoxFinderNumer)+4),HunterSec), delay\_ms(1);

Enable the Interrupts.

INTCON.GIE = 1;

Increment the Fox Finder Number variable, this variable is than converted and send in a Morse Code Format.

FoxFinderNumer++;

}

Reset the Hunt Number received by the USART in the Punch action, for safety.

HunterNumer = 0;

}

}

//-----------------------------------------------------------------------------------------------------------------------------------

//---------------------------- Interrupt Program ---------------------------------------------------------------------------------------------------

//---------------------------------------------------------------------------------------------------------------------------------------

**Interrupt**

Definition:

In systems programming, an interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities. There are two types of interrupts: hardware interrupts and software interrupts.

Hardware Interrupt: USART - universal asynchronous receiver/transmitter

Software Interrupt: Timer 1 Overflow - the change from 0xFFFF to 0x0000.

void Interrupt()

{

//---------- Read UART Data -----------

When the Interrupt event occurs, first is check if was the USART event.

if(PIR1.RCIF == 1) // Interrupt generated at 250 ms

{

If Yes, Read the USART Register in the variable *UART\_Read\_Data* .

UART\_Read\_Data = Usart\_Read(); // read the received data,

Then, we check if a Hunter punch the Fox, for this we have more conditions. The Hunt number must be less than 0x3D (61) and bigger than 0. The second conditions must be that the Fox is not in Diagnostic Mode *Programming\_Fox==0 ,* must have Hunt Programmed *Program\_Flg<0xFF,* To not be in the Wait time before the Hunt starts *Delay\_to\_Start\_Led==0* and the Hunt time to be bigger than 0 *Max\_Time\_Var>0.*

if((UART\_Read\_Data >0)&(UART\_Read\_Data < 0x3D)&(Programming\_Fox==0) & (Max\_Time\_Var>0) & (Program\_Flg<0xFF) & (Delay\_to\_Start\_Led==0)) // Verify if is Puncher

{

If all the conditions are True, than we set the LED ON.

PORTA.F0 = 1;

Reset the LED ON timer (LED must stay 1 sec ON, even if the Punch duration is 50 ms)

SecToTx = 0;

Assign to the follow variables the Punch Number and the current Fox running Time.

HunterNumer = UART\_Read\_Data;

HunterSec = Sec;

HunterMinute = Minute;

HunterHour = Hour;

Jump to Mode Function ->

Mode();

}

If the Hunter did not punch the FOX, we check the Diagnostics commands.

else // Else - read diagnosis commands

{

If the Fox is in Hunt Programming Mode, than Jump to the Function Mode ->

if((EEP\_Data\_Read\_Write==2)&(Programming\_Fox==1)) Mode();

else, check if the data receive is 0x50, if Yes set the Programming\_Fox flag to 1 and set the LED ON.

else if(UART\_Read\_Data == 0x50) Programming\_Fox = 1, PORTA.F0 = 1; // Led ON

else, if the data receive is 0x3F and the Programming\_Fox flag is 1 and no Read/Write command was selected before, set the variable *EEP\_Data\_Read\_Write=2* , this means that the Fox is set for Hunt Programming. Next time when will enter in the Interrupt, will go on the first branch -> to Mode and record the serial data.

else if((UART\_Read\_Data == 0x3F)&(EEP\_Data\_Read\_Write==0)&(Programming\_Fox==1)) EEP\_Data\_Read\_Write=2; // Write

else, if the data receive is 0x2F and the Programming\_Fox flag is 1 and no Read/Write command was selected before *EEP\_Data\_Read\_Write=0*, set the variable *EEP\_Data\_Read\_Write=1 and go to the Function Mode* , this means that reading all the EEPROM data and send it to the serial bus USART.

else if((UART\_Read\_Data == 0x2F)&(EEP\_Data\_Read\_Write==0)&(Programming\_Fox==1)) EEP\_Data\_Read\_Write=1, Mode(); // Read

else, if the data received is 0x4F and the Programming\_Fox flag is 1 and no Read/Write command was selected before *EEP\_Data\_Read\_Write=0*, than go to the Function Reset; This is called in the PRAL – Clear Program.

else if((UART\_Read\_Data == 0x4F)&(EEP\_Data\_Read\_Write==0)&(Programming\_Fox==1)) Reset(); // Reset Fox

}

}

**USART Commands:**

The USART is using the RS-232 protocol but with 0-5 V levels. It has 1 start bit, 8 data bits and 1 stop bit.

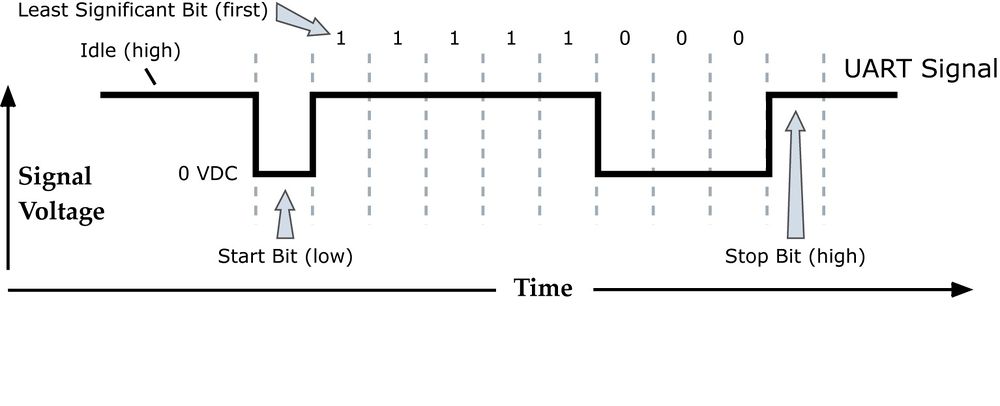


Fig. 1 Serial signal.

The communication in the Fox is split in 2 types, Puncher and PRAL.

**Puncher:**

When the Hunter punch the Fox, the Puncher is sending first the Punch ID, this ID should be less than 60 and bigger than 0. Than the Fox will send the Fox number and the Time from the start of the Hunt.

Puncher: 0x03 -> Fox

0x03 is the Puncher ID

Fox: 0x01 0x02 0x03 0x04 -> Puncher

0x01 Fox Number, 0x02 Hour, 0x03 Minutes, 0x04 Seconds

**PRAL:**

For Programming the Fox or Reading or Resetting, we have special commands.

PRAL : 0x50 0x3F 0x01 0x02 0x03 0x04 0x00 -> Fox

Programming the Fox, 0x50 0x3F – Command for Programming, 0x01 – Delay to start, 0x02 – Hunt Time,

0x03 – Hunt Type, 0x04 – Frequency, 0x00 – Programming Flag.

PRAL : 0x50 0x2F -> Fox

0x50 0x2F – Command for Read all the Memory

Fox (send the EEPROM data from address 1 to 240 –> 60 bytes) -> PRAL

PRAL : 0x50 0x4F -> Fox

0x50 0x4F – Reset command, the ECU will restart exactly as after Power ON.

//---------- Timer Function -----------

The second check when the Interrupt occurs is the Timer interrupt at Overflow.

if(PIR1.TMR1IF == 1)

{

The Fox Timer is a 2 bytes counter, that means 65535 iterations until it overflows. In the Init Function, the timer is configured with a Prescaler Rate 8, => Overflow time = ( 1 / (Cristal Frequency / 4 / 8) ) \* 65535 = 0.26214 s .

To have a fix time interrupt at 0.250 s we start the counter from 3036 (3036 \* 0.000004 = 0.012144 s) => 0.26214 - 0.012144 = 0.249996 s ~ 250 ms

Assign to the 16 bit counter value 3036 = 0x0B 0xDC.

TMR1H = 0x0B;

TMR1L = 0xDC;

Set the Interrupt Flag to 0. To exit the Interrupt program after finishing the tasks.

PIR1.TMR1IF =0;

Each interrupt event is at 250 ms, we need to count 4 interrupts for 1 second. The variable *Time* is incremented.

Time++;

If the variable *Time* is equal with 4,

if(Time >= 4 )

{

increment seconds,

Sec++; // Increase Seconds

if variable *s\_End\_Wait\_Time* is > 0 than decrease by one. This variable is for waiting after each Mores sending cycle.

if(s\_End\_Wait\_Time > 0) s\_End\_Wait\_Time--;

if variable *Wait\_Time* is > 0 than decrease by one. This variable is use to count the wait seconds before start to send the first word. For example if the Fox number is 2, we wait 2 min before start in case of Hunt Type 3.

if(Wait\_Time > 0) Wait\_Time--;

if variable *TimeToSend* is > 0 than decrease by one. This arable is the time in seconds for each Cycle.

if(TimeToSend > 0) TimeToSend--;

This condition is the blink the LED at 2 seconds if we are Programed and the Wait time before the Hunt starts is > 0 , *Delay\_to\_Start\_Led* > 0.

if((Delay\_to\_Start\_Led > 0)&(Program\_Flg != 0xFF)&(Programming\_Fox==0)) PORTA.F0 =!PORTA.F0;

If the Sec variable reach 60, this means that we have 1 min.

if(Sec >= 60 )

{

Increment Minute variable by 1.

Minute++;

if *Delay\_to\_Start\_Led* variable is > than 0, decrease by 1.

if(Delay\_to\_Start\_Led > 0) Delay\_to\_Start\_Led--;

if *Max\_Time\_Var* variable is > than 0 and the Delay before Hunt start is 0, decrease *Max\_Time\_Var* by 1. This variable is the Hunt Time.

if((Max\_Time\_Var > 0)&(Delay\_to\_Start\_Led == 0)) Max\_Time\_Var--;

if Minute is > than 60, increment the Hour by 1 and set Minute to 0.

if(Minute >= 60 ) Hour++, Minute=0;

Reset second to 0.

Sec=0;

}

Reset time to 0.

Time=0;

}

}

If the Hunter punch the Fox, we lite the LED 1 second. This function is counting 1 second after the Fox receive the Hunter ID. There are 4 condition to set the LED OFF. We must be in Non Programming mode, the Hunt to be started, the delay before the Hunt to be 0 and the Fox to be Programed.

if(SecToTx < 4) SecToTx++;

else if((Programming\_Fox==0) & (Program\_Flg != 0xFF) & (Max\_Time\_Var > 0) & (Delay\_to\_Start\_Led == 0)) PORTA.F0 = 0;

} // End Interrupt

End Interrupt Program.

1. **Main Program**

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////-----------------MAIN--------------------/////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Pin Input/Output Information

// PORTA.F1 = LED\_Mode;

// PORTA.F0 = LED\_R\_W;

// PORTA.F4 = CW\_Keying; // Sending Morse

// PORTA.F3 = En\_144;

// PORTA.F2 = En\_3,5 ;

// PORTB.F7 = Crystal 1;

// PORTB.F6 = Crystal 2;

// PORTB.F4 = Low Pow;

// PORTB.F3 = PWM ;

// PORTB.F2 = Rx;

// PORTB.F5 = Tx ;

**Initialization Function**

Initialization is the first part in the main program. In this part we configure the direction of each pin and the function as Serial communication protocol USART and PWM. The Interrupt timer configuration is also here. All the functions must be before the main program to be able to access them.

// ----------------------- Init ECU Function -----------------------------------------------------------------------------------------------------

void init()

{

Disable the Analog inputs. Some of the microcontrollers need to disable manually the ADC port because this is the default value.

ANSEL = 0x00; // Configure AN pins as digital ANSELH = 0;

ADCON0 = 0x00;

Disable the Comparator input and use the pins as Digital. See the PIC16F88 Datasheet for more details related to this register.

CMCON = 0x07;

Configure the pins for RA port and RB port. If the bit is 1 means Input if is 0 means Output. TRISA = 0xC0 = 1100 0000 from pin RA0 to RA5 we have output pins and pin RA6 and RA7 are inputs. TRISB = 0x04 = 0000 0100 here we have configured just RB2 as Input, rest of the pins are Output.

TRISA = 0xC0; // Configure PORT A as Input

TRISB = 0x04; // Configure PORT B as Input

Set all the outputs to 0 logic = GND.

PORTA = 0x00; // Set PORT A to 0x00;

PORTB = 0x00; // Set PORT B to 0x00;

Activate the Interrupts for USART and Overflow of the Timer1.

INTCON = 0xC0; // Activate Interrupt - PEIE: Peripheral Interrupt Enable bit - T0IE: Timer0 Overflow Interrupt Enable bit

PIE1 = 0x21; // RCIE: EUSART Receive Interrupt Enable and TMR 1

T1CON = 0xF1; // Activate 16 bit Timer 1 - Prescalar Rate 8 count at 4 us - 65536 - > max time 262,144 ms

Configure USART speed as 4800 bps. This is a MikroC function. This function is configure all the bits from the USART register from the PIC.

Usart\_Init(4800); // Initialize UART module at 4800 bps

Activate the PWM. This is also a MikroC function, is more easy to use this function instead set all the bits from the PIC registres. See the HELP from MikroC program for more details.

Set PWM frequency to 700 kHz.

Pwm\_Init(700);

Start PWM.

Pwm\_Start();

Set Duty cycle to 0%.

Pwm\_Change\_Duty(0);

Activate the LED for the startup process. Is just a confirmation that the PIC is powered and ready to use.

PORTA.F0 = 1;

Assigned the fox number to the variable FoxNumber.

FoxNumber = FoxHuntNumber;

}

**Type Function**

After the Fox was programmed, the next function accessed is the Type Function. Here we set all the parameters according with the following Hunt Types.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Type** | **Cycle** | **Foxes** | **Count+Tone** | **ID (sm6rxz)** | **power** | **wpm** | **comment** | **Use** |
| 1 | 1 | 1 | y | y | High | 5 |  | GRJ 1.1 |
| 2 | 2 | 2 | y | y | High | 5 |  | GRJ 1.2 |
| 3 | 3 | 3 | y | y | High | 5 |  | GRJ 1.3 |
| 4 | 4 | 4 | y | y | High | 5 |  | GRJ 1.4 |
| 5 | 5 | 5 | y | y | High | 5 |  | GRJ 1 |
| 6 | 7 | 7 | y | y | High | 5 |  | GRJ 2 |
| 7 | 10 | 10 | y | y | High | 5 |  | GRJ 3 |
| 8 | 5 | 5 | n | y | High | 5 |  | SM Night |
| 9 | 7 | 7 | n | y | High | 5 |  | SM Day |
| 10 | 10 | 10 | n | y | High | 5 |  |  |
| 11 | N/A | X | n | n | Low | 8 | Low power Rules for Foxoring specifies use of 5 to 10 foxes | Beginners training/show, also Foxoring |
| 12 | N/A | X | n | n | High | 5 | High power Rules for Foxoring specifies use of 5 to 10 foxes |  |
| 13 | 1 | 5 | n | n | Low | See comment | 10WPM - 1,2,3,4,5 - Crystal 1 14WPM - 1,2,3,4,5 - Crystal 2 the Foxes 6,7,8,9,10 will be changed (6-1, 7-2, 8-3, 9-4, 10-5)  (each fox transmits 12 second each, in a 1 minute cycle) | Sprint |
| 14 | 2 | 10 | n | n | High | 16 | faster (16wpm), no ID (sm6rxz)  (each fox transmits 12 seconds each, in a 2 minute cycle) | Renberg-hunt (GRJ Sprint) |
| 15 | 5 | 5 | n | n | High | 8 | 6 and 7 send as 4 and 5 - on Crystal2 | National |

// ----------------------- Type Function ---------------------------------------------------------------------------------------------------

void Type(int Hunt\_Type)

{

Set the default values for the Hunt.

Enable the Cristal 1. For this we set the PORTB Pin 6 to GND and Pin 7 to 5V.

PORTB.F6 = 0; // Enable crystal 1 by default

PORTB.F7 = 1;

Enable High Power.

PORTB.F4 = 1; // High Power

Set the WPM to 5.

WPM\_data = 5;

Enable Sending the ID SM6RXZ .

FoxSndID\_Enable = 1;

Enable the Hunt counters. To send the count how Meany Hunters punch the Fox.

FoxHunters\_Enable = 1;

Configure all the variables according with the Hunt Type programed.

if (Hunt\_Type == 0x01) Cycle = 1, Foxes = 1; // FoxHunters\_Enable = 1, FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // GRJ 1 1

else if(Hunt\_Type == 0x02) Cycle = 2, Foxes = 2; // FoxHunters\_Enable = 1, FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // GRJ 1 2

else if(Hunt\_Type == 0x03) Cycle = 3, Foxes = 3; // FoxHunters\_Enable = 1, FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // GRJ 1 3

else if(Hunt\_Type == 0x04) Cycle = 4, Foxes = 4; // FoxHunters\_Enable = 1, FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // GRJ 1 4

else if(Hunt\_Type == 0x05) Cycle = 5, Foxes = 5; // FoxHunters\_Enable = 1, FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // GRJ 1

else if(Hunt\_Type == 0x06) Cycle = 7, Foxes = 7; // FoxHunters\_Enable = 1, FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // GRJ 2

else if(Hunt\_Type == 0x07) Cycle = 10, Foxes = 10;// FoxHunters\_Enable = 1, FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // GRJ 3

else if(Hunt\_Type == 0x08) Cycle = 5, Foxes = 5, FoxHunters\_Enable = 0; // FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // SM Night

else if(Hunt\_Type == 0x09) Cycle = 7, Foxes = 7, FoxHunters\_Enable = 0; // FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5; // SM Day

else if(Hunt\_Type == 0x0A) Cycle = 10, Foxes = 10, FoxHunters\_Enable = 0; // FoxSndID\_Enable = 1, High\_Pow(1), WPM\_data=5;

else if(Hunt\_Type == 0x0B) Cycle = 0, Foxes = 99, FoxHunters\_Enable = 0, FoxSndID\_Enable = 0, PORTB.F4 = 0, WPM\_data=8; // Beginners training/show or Foxoring 1

else if(Hunt\_Type == 0x0C) Cycle = 0, Foxes = 99, FoxHunters\_Enable = 0, FoxSndID\_Enable = 0, PORTB.F4 = 1, WPM\_data=8;

else if(Hunt\_Type == 0x0D) Cycle = 1, Foxes = 5, FoxHunters\_Enable = 0, FoxSndID\_Enable = 0, PORTB.F4 = 0, WPM\_data=10; // Sprint

else if(Hunt\_Type == 0x0E) Cycle = 2, Foxes = 10, FoxHunters\_Enable = 0, FoxSndID\_Enable = 0, PORTB.F4 = 1, WPM\_data=16; // Renberg-hunt

else if(Hunt\_Type == 0x0F) Cycle = 5, Foxes = 5, FoxHunters\_Enable = 0, FoxSndID\_Enable = 0, PORTB.F4 = 1, WPM\_data=8; // National

If the Hunt type is select as Sprint and the Foxes are 6,7,8,9,10 than we must change the sending Fox number to 1,2,3,4,5 and change the sending speed to 14 WPM.

if (Hunt\_Type == 0x0D) // Sprint - 14WPM - 1,2,3,4,5 - Crystal 2 the Foxes 6,7,8,9,10 will be changed (6-1, 7-2, 8-3, 9-4, 10-5)

{

Verify if the Fox number is 6.

if (FoxHuntNumber == 6)

{

If yes, change the sending Morse code

Message\_to\_Send\_Wrd\_3 = 0b00000010; // Fox Number 1

data\_lgt\_fox = 4; // Length of the Morse Fox Count 4

Change the Fox number to 1

FoxHuntNumber = 0x01; // Fox Hunt Number - max foxes 254

Change the WPM to 14.

WPM\_data=14;

Enable Crystal 2.

PORTB.F6 = 1, PORTB.F7 = 0; // Enable cristal 2

}

Verify if the Fox number is 7.

else if (FoxHuntNumber == 7)

{

If yes, change the sending Morse code

Message\_to\_Send\_Wrd\_3 = 0b00000100; // Fox Number 2

data\_lgt\_fox = 6; // Length of the Morse Fox Count 6

Change the Fox number to 2

FoxHuntNumber = 0x02; // Fox Hunt Number - max foxes 254

Change the WPM to 14.

WPM\_data=14;

Enable Crystal 2.

PORTB.F6 = 1, PORTB.F7 = 0; // Enable cristal 2

}

Verify if the Fox number is 8.

else if (FoxHuntNumber == 8)

{

If yes, change the sending Morse code

Message\_to\_Send\_Wrd\_3 = 0b00001000; // Fox Number 3

data\_lgt\_fox = 8; // Length of the Morse Fox Count 8

Change the Fox number to 3

FoxHuntNumber = 0x03; // Fox Hunt Number - max foxes 254

Change the WPM to 14.

WPM\_data=14;

Enable Crystal 2.

PORTB.F6 = 1, PORTB.F7 = 0; // Enable cristal 2

}

Verify if the Fox number is 9.

else if (FoxHuntNumber == 9)

{

If yes, change the sending Morse code

Message\_to\_Send\_Wrd\_3 = 0b00010000; // Fox Number 4

data\_lgt\_fox = 10; // Length of the Morse Fox Count 10

Change the Fox number to 4

FoxHuntNumber = 0x04; // Fox Hunt Number - max foxes 254

Change the WPM to 14.

WPM\_data=14;

Enable Crystal 2.

PORTB.F6 = 1, PORTB.F7 = 0; // Enable cristal 2

}

Verify if the Fox number is 10.

else if (FoxHuntNumber == 10)

{

If yes, change the sending Morse code

Message\_to\_Send\_Wrd\_3 = 0b00100000; // Fox Number 5

data\_lgt\_fox = 12; // Length of the Morse Fox Count 12

Change the Fox number to 5

FoxHuntNumber = 0x05; // Fox Hunt Number - max foxes 254

Change the WPM to 14.

WPM\_data=14;

Enable Crystal 2.

PORTB.F6 = 1, PORTB.F7 = 0; // Enable cristal 2

}

}

If the Hunt type is National and the foxes are 6 and 7 than the Crystal 2 must be selected and the Fox numbers to be changed to 4 and 5.

Verify if the Hunt type is 15.

else if (Hunt\_Type == 0x0F) // National - 6 and 7 send as 4 and 5 - on Crystal 2

{

Verify if the Fox number is 6.

if (FoxHuntNumber == 6)

{

If yes, change the sending Morse code

Message\_to\_Send\_Wrd\_3 = 0b00010000; //6 -> 4

data\_lgt\_fox=10;

Change the Fox number to 4

FoxHuntNumber = 0x04;

Enable Crystal 2.

PORTB.F6 = 1, PORTB.F7 = 0; // Enable cristal 2

}

Check if the Fox is 7

else if (FoxHuntNumber == 7)

{

If Yes, Change the sending Morse code

Message\_to\_Send\_Wrd\_3 = 0b00100000; //7 -> 5

data\_lgt\_fox=12;

Change the Fox number to 5

FoxHuntNumber = 0x05;

Enable Crystal 2.

PORTB.F6 = 1, PORTB.F7 = 0; // Enable cristal 2

}

}

Convert the WPM speed in time , ms.

There are 2 speeds, CW\_Speed\_1 for sending the MOX and Hunter numbers that punch the Fox and CW\_Speed\_2 for sending the ID SM6RXZ.

CW\_Speed\_1 = 1200 / WPM\_data; // Calculate WPM in ms

CW\_Speed\_2 = 1200 / WPM\_data\_ID; // Calculate WPM in ms

Calculate the sending time in ms and s based on how many foxes and cycles are configured from the Type.

s\_FoxCycle = (Cycle \* 60) / Foxes; Time in seconds.

ms\_FoxCycle = s\_FoxCycle\*1000; Time in mili seconds.

}

**Convert to Morse Function**

This function is used to transform the integer data in Morse data. Is using a Case structure. Each number has assigned a Morse code.

// --------------------------- Convert To Morse ------------------------------------------------------------------------------------------------

char ConvertToMorse(char lettre\_cw)

{

char Conv\_Data = 0;

The function returns one byte codded in Mores, according with the Input *int* variable *letter\_cw.*

switch(lettre\_cw)

{

case 0: Conv\_Data = 0b00111111, data\_lgt= 22; break;

case 1: Conv\_Data = 0b00111110, data\_lgt= 20; break;

case 2: Conv\_Data = 0b00111100, data\_lgt= 18; break;

case 3: Conv\_Data = 0b00111000, data\_lgt= 16; break;

case 4: Conv\_Data = 0b00110000, data\_lgt= 14; break;

case 5: Conv\_Data = 0b00100000, data\_lgt= 12; break;

case 6: Conv\_Data = 0b00100001, data\_lgt= 14; break;

case 7: Conv\_Data = 0b00100011, data\_lgt= 16; break;

case 8: Conv\_Data = 0b00100111, data\_lgt= 18; break;

case 9: Conv\_Data = 0b00101111, data\_lgt= 20; break;

}

return Conv\_Data;

}

**Activate outputs for sending Morse data Function**

The function is use to change the Out pins according with the *data* input variable.

// --------------------------- Send MORSE Data Out Function ---------------------------------------------------------------------------------------------

void CW\_Keying(char data)

{

If *data* variable is 1.

if(data==1)

{

Change the PORTA pin 4 to 1 logic, 5V. This is the Keying pin. This is having another condition, based on the PORTA pin 2 that is the Frequency.

if(PORTA.F2==1) PORTA.F4 = 1;

Send 50% duty cycle on the PWM pin if the PORTA pin 3 is activated.

if(PORTA.F3==1) Pwm\_Change\_Duty(127);

Activate PORTA pin 1 for testing the data output.

PORTA.F1 = 1;

}

If the *data* variable is different than 1.

else

{

Deactivate all the outputs for morse sending.

PORTA.F4 = 0;

Pwm\_Change\_Duty(0);

PORTA.F1 = 0;

}

}

**Send Morse Data Function**

Function use to send data according with the Morse code and speed. One morse character per BYTE, bitwise, LSB to MSB. 0 = dit, 1 = dah. The byte is shifted to the right bit by bit, until the last 1 is left, this 1 is an END OF CHARACTER indicator. A maximum of 7 elements can be encoded, (error) is excluded.

// --------------------------- Send Word Function ---------------------------------------------------------------------------------------------

void Send\_Word(int Speed, int Word)

{

Check of the Input data *Word* is 0, if yes then we wait 2 dots with wait time variable *Speed,* WPM speed in ms.

if (Word == 0) VDelay\_ms(Speed\*2); // Word Space time 5 dots

If the Input variable *Word* is 0xFF, we do nothing.

if (Word == 0xFF) {}

else we start to transmit the data.

else

{

While sending data is bigger than 1 e send the dots and dashes.

while(Word>1)

{

Check if the first bit from the morse character is dash.

if((Word & 0x01) == 0x01)

{

If yes, set the Keying to 1.

CW\_Keying(1); // Dash 3 dots

Wait dash time, this is based on the speed variable multiply by 3.

VDelay\_ms(Speed\*3);

Deactivate the Outputs.

CW\_Keying(0);

}

else, if is a dot

else

{

set the Keying to 1.

CW\_Keying(1); // Dot 1 dot

Wait one dot time.

VDelay\_ms(Speed);

Deactivate the outputs.

CW\_Keying(0);

}

Move to the next bit from the Morse character.

Word = Word >> 1;

Wait one dot time before sending another dot or dash.

VDelay\_ms(Speed); // Inter Dot Space 1 dot

}

Wait another 2 dots after all the dots and dashes are sent. In total will be 3 dots between the Morse words.

VDelay\_ms(Speed\*2); // Inter Char Space 3 dots

This is a condition to deactivate the sending if the user is trying to Program the FOX.

while(Programming\_Fox == 1)

{

CW\_Keying(0);

}

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Start Main Program \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Main Program**

After the PIC is supply, here is the start of the Program. Every time when is an interrupt or a function called, the return of the program will always be here, where the program was interrupt.

void main()

{

// ------- Initialize the Micro --------

Go to initialization function *init(),* to set all the outputs.

init();

Disable the Sending Morse data.

CW\_Keying(0);

// ------- Verify if not Prog --------

Check if the Fox is not proggramed.

while(Program\_Flg == 0xFF)

{

If is not programmed, chance the PORTA pin 0 to the negation of the current state. So if the pin is 0 the negation is 1. This process is done with a 120 ms delay.

Vdelay\_ms(120);

if(Programming\_Fox == 0) PORTA.F0 = !PORTA.F0;

}

// ------- Configure Hunt type - Delay to start - Reset time -------

After Programming the Fox, we configure the Hunt type according with the received value during the Programming process.

Type(Hunt\_Type);

// ------- Wait startup delay -----

As the delay before start is bigger than 0 we do nothing.

while(Delay\_to\_Start\_Led > 0){} // Wait

Reset all the times after end of Waiting time.

Time = 0; // Reset Time

Sec = 0;

Minute = 0;

Hour = 0;

PORTA.F0 = 0; // LED OFF

// ------- Wait time if the Fox is not number 1 -------

If the cycle is different than 0, we must wait for the Fox number time.

if(Cycle != 0x00)

{

Calculate the wait time, if the fox for example is 4 we wait 3 \* cycle time in seconds. If one cycle is 1 sec, we wait 3 sec and send between 3 and 4.

Wait\_Time =((FoxHuntNumber-1) \* s\_FoxCycle);

Wait until the Wait Time is 0.

while( Wait\_Time > 0 ) {}

}

// ----------------------- Start Sendin Morse Data Program ------------------------

Start the infinite loop. This loop is end when the power from the Fox is removed.

while(1)

{

// --- enable the frequency befor transmission

According with the Programed Frequency, activate the Output pins.

if(Frequency == 0x00) PORTA.F3 = 0, PORTA.F2 = 1; // Enable 3,5

else if(Frequency == 0x01) PORTA.F3 = 1, PORTA.F2 = 0; // Enable 144

else if(Frequency == 0x02) PORTA.F3 = 1, PORTA.F2 = 1; // Enable both

else PORTA.F3 = 0, PORTA.F2 = 0; // Disable both

// --- If the Hunt is finish - send Fox ID

Check if the Hunt Time is finish.

while(Max\_Time\_Var == 0)

{

If yes, set the LED to ON.

PORTA.F0 = 1;

Send the FOX number.

Send\_Word(CW\_Speed\_1, Message\_to\_Send\_Wrd\_3); // 2\* dot x

Wait 1 sec.

Vdelay\_ms(1000);

}

Check if the cycle is different than 0.

if(Cycle != 0x00) // If Cycle is not 0 - calculate WPM Sliding

{

// --- Convert Hunter number in Morse code ---

If Yes, calculate convert the numbers of the Hunter count to Morse Data.

Message\_to\_Send\_FoxHunters\_1 = ConvertToMorse(FoxFinderNumer/10);

TimeHntNr\_1 = data\_lgt;

Message\_to\_Send\_FoxHunters\_2 = ConvertToMorse(FoxFinderNumer - ((FoxFinderNumer/10)\*10));

If the First digit is 0, do not send it.

if(Message\_to\_Send\_FoxHunters\_1 == 63) TimeHntNr\_1 = 0, Message\_to\_Send\_FoxHunters\_1=0xFF;

TimeHntNr\_2 = data\_lgt;

// --- Add End Time for different configurations ---

Set the variable *TimeDev* to the end wait time, this is dependent if the Sending ID and/or Hunter count is activated. It is also used for define the sending data in the cycle.

if((FoxHunters\_Enable == 1) & (FoxSndID\_Enable == 1)) TimeDev = ((TimeHntNr\_1 + TimeHntNr\_2 + 8)\*CW\_Speed\_1) + 8830;

else if(FoxSndID\_Enable == 1) TimeDev = 6830;

else TimeDev = 2500 ;

Reinitialize the Sending time, in seconds. Is the Cycle time for each transmition.

TimeToSend = s\_FoxCycle; // Calculate Time to send in s

Calculate the MOX length, for example M= -- =0b00000111 the length is 10, we have (2 dash = 6) + (2 inter dash space with 1 dit length) + (2 dits inter char space) the transformation in dits will look like this: 0b0000010(000)0(000)00

The length is the number of the fox length + M (10 dots) + O (14 dots) + inter Word space (2 dots)

MOx\_Length = data\_lgt\_fox + 26; // Calculate MOx length 38

ms\_TimSnd\_MOx = (CW\_Speed\_1\*MOx\_Length); // Calculate Time to send MOx in ms

// calculate WPM Drifting-------------------

To have the exact sending time with no interruption of the message, there is implemented a Words Per Minute Drift algorithm. If we change just a little of the WPM in sending the MOx message, than we can have the exact time lext for sending the ID and Hunter Count. This algorithm in disable when the Fox is sending continuously.

1. First we calculate the integer value of how many MOx cycles can be sent in the time FoxCycle.

cycles\_MOx = (ms\_FoxCycle - TimeDev) / ms\_TimSnd\_MOx;

1. Then we calculate the error, the value that is remaining after we divide Window time to the MOx time.

cycle\_Corection = ((ms\_FoxCycle - TimeDev) % ms\_TimSnd\_MOx);

1. If the correction is less than half of the time for sending MOX, than increase the WPM to have the exact cycles for MOX.

if(cycle\_Corection < (ms\_TimSnd\_MOx/2)) WPM\_Sliding = CW\_Speed\_1 + (cycle\_Corection / MOx\_Length / cycles\_MOx);

else we decrease the WPM time and add a new MOX cycle.

else cycles\_MOx = cycles\_MOx+1, WPM\_Sliding = (CW\_Speed\_1 - ((ms\_TimSnd\_MOx - cycle\_Corection )/MOx\_Length / cycles\_MOx));

}

---------------------- Example:

One cycle is 60 sec => ms\_FoxCycle = 60000 ms

TimeDev = 0;

ms\_TimSnd\_MOx = 38\*240 = 9120 ms

cycles\_MOx = 60000 / 9120 = 6.578 we can send 6 MOX in one cycle, but we still have time to send “MO”. To not confuse the Hunter, we change the WPM Speed to have exact number of MOX in a sending cycle.

cycle\_Corection = ms\_FoxCycle % ms\_TimSnd\_MOx = 5280 ms , this is the time left , is less than 9120 ms and in not enough for MOX.

To not have a bigger deviation from the base WPM speed, we check which is better, to increase the WPM and have the same MOX cycles or to decrease the WPM and have one more MOX cycle. In this case the WPM will have the range between 4,5 – 5,5 WPM .

If cycle\_Corection (5280 ms)< (ms\_TimSnd\_MOx/2)( 4560 ms)

WPM\_Sliding = CW\_Speed\_1 (240 ms) + (cycle\_Corection (5280 ms)/ MOx\_Length (38)/ cycles\_MOx(6)) =263 ms (4,56 WPM)

Else

cycles\_MOx = cycles\_MOx+1 = 7

WPM\_Sliding = (CW\_Speed\_1(240 ms) - ((ms\_TimSnd\_MOx (9120ms) - cycle\_Corection(5280 ms) )/MOx\_Length (38)/ cycles\_MOx(6)) = 223 ms (5,38 WPM)

--------------------------------------------------------------------

Else if the Cycle from the Type is 0, this means that the Fox will send continuously and we don’t care to fit in the cycle. The speed will be the default one.

else // Else - set the WPM Sliding to Default WPM value

{

WPM\_Sliding = CW\_Speed\_1;

cycles\_MOx = 5; // MOX cycles befor sending the Hunter counter or/and FoxSender ID

}

While the MOX cycles are bigger than 0, send MOX.

while(cycles\_MOx > 0) // Send MOx

{

Send “M”

Send\_Word(WPM\_Sliding, Message\_to\_Send\_Wrd\_1); // 10 dot

Send “O”

Send\_Word(WPM\_Sliding, Message\_to\_Send\_Wrd\_2); // 14 dot

Send “X” (number of the FOX)

Send\_Word(WPM\_Sliding, Message\_to\_Send\_Wrd\_3); // 2\* dot x

Wait between the Words.

Send\_Word(WPM\_Sliding, 0); // 2 dot

Decrease by one the MOX cycles.

cycles\_MOx--;

}

Check if the Enable Sending ID is set, if yes , send SM6RXZ.

if(FoxSndID\_Enable == 1) // Send Fox ID SM6RXZ

{

Send\_Word(CW\_Speed\_2, Message\_to\_Send\_Wrd\_4); // 8 dot

Send\_Word(CW\_Speed\_2, Message\_to\_Send\_Wrd\_5); // 10 dot

Send\_Word(CW\_Speed\_2, Message\_to\_Send\_Wrd\_6); // 14 dot

Send\_Word(CW\_Speed\_2, Message\_to\_Send\_Wrd\_7); // 10 dot

Send\_Word(CW\_Speed\_2, Message\_to\_Send\_Wrd\_8); // 14 dot

Send\_Word(CW\_Speed\_2, Message\_to\_Send\_Wrd\_9); // 14 dot

Send\_Word(CW\_Speed\_2, 0); // 2 dot

} // \_\_\_\_\_\_\_\_\_\_\_\_\_\_

// 72 dots

Check if the function to Send the Hunter count is Enable,

if(FoxHunters\_Enable == 1) // If FoxHounters count enable, send the Hunters Number

{

If yes, than send the Hunters number, that Punch the Fox.

Send\_Word(CW\_Speed\_1,Message\_to\_Send\_FoxHunters\_1); //0

Send\_Word(CW\_Speed\_1,Message\_to\_Send\_FoxHunters\_2); // 22

Send\_Word(CW\_Speed\_1, 0); // 5 dot

Send long tone.

CW\_Keying(1); // Send Long Tone

Wait until the Fox sending cycle is finish.

while(TimeToSend > 0){} // Wait cycle to end time

CW\_Keying(0); // Stop Long Tone

}

if(Cycle != 0x00) // If Cycle is not 0 - Wait for the fox number cycle

{

Disable the Transmitters

PORTA.F3 = 0, PORTA.F2 = 0; // disable the transmission for both Frq

Wait until the Fox sending cycle is finish.

while(TimeToSend > 0){} // Wait cycle to end time

Calculate the waiting time, based on the number of the cycles and the cycle time.

s\_End\_Wait\_Time = ((Foxes-1) \* s\_FoxCycle); // Wait untill next Tx

if(s\_End\_Wait\_Time == 0) Send\_Word(CW\_Speed\_1, 0); // Wait inter Cycle space

Wait until the rest of the Foxes sent their own data.

while(s\_End\_Wait\_Time > 0){}

}

} // End While

} // End Main

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ End Main Program \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Microcontroller Program Statistics and Information**

