ANEXOS

SOFTWARE

Función Principal

```
int main(void)
{
   //INICIALIZAR EQUIPO
   SYSTEM_Initialize();

   //INICIALIZAR VARIABLES
   bool Reboot = 1;                //Variable de control de reinicio
   uint8_t Time[3];                //[Segundos Minutos Hora]
   uint8_t Date[4];                //[Dia Fecha Mes Anyo]
   uint8_t Bits_Equipos = 0;       //Cada bit representa uno de lso equipos ocnectados)
                                   //por bit: 1=Activar, 0=Desactivar
   uint8_t Bits_Equipos_Horario = 0;
   uint8_t Save = 0;               //Variable auxiliar
   uint16_t Count = 0;
   uint8_t led = 0;

   //LED parpadea 3 veces para indicar inicio
   LED = 1;
   while(led < 5){
      delay_ms(100);
      LED = !LED;
      led++;
   }

   //Leer hora antes de iniciar cualuqer operacion
   //Podria agregarse un error de reloj aqui en caso de que no se peuda leer
   RTC_LEER(0b1101000,0x00,3,Time);    //Lee la hora

   Time[0] = RTC_SECONDS(Time[0]);     //Segundos en decimal
   Time[1] = RTC_MINUTES(Time[1]);     //Minutos en decimal
   Time[2] = RTC_HOUR(Time[2]);        //Hora en decimal

   GET_MODBUS_ID();                    //Asigna a la remora el ID almacenado

   //CODIGO PRINCIPAL - BLUCLE INFINITO
```

```
  while (1)
  {

  if( ((Time[0] == 0)&&(Time[1] == 0)&&(Time[2] == 0)) || (Reboot == 1) ){  //Cada 24 horas o al reiniciar
    Reboot = 0;

    //Leer el dia, la fecha, el mess y el anyo del RTC
    RTC_LEER(0b1101000,0x03,4,Date);
    Date[0] = RTC_DAY(Date[0]);        //Dia en decimal
    Date[1] = RTC_DATE(Date[1]);       //Fecha en decimal
    Date[2] = RTC_MONTH(Date[2]);      //Mes en decimal
    Date[3] = RTC_YEAR(Date[3]);       //Anyo en decimal

    //Rellenar la estructura de datos
    CONTROL_BYTES();
    STRUCTURE_FILL(Date[0],Date[1],Date[2]);

    //Reset a los reles forzados
    RESET_RELES();
  }

  //Leer la hora del RTC
  if(Count == 50){                  //Cada medio segundo
    Count = 0;
    RTC_LEER(0b1101000,0x00,3,Time);
    Time[0] = RTC_SECONDS(Time[0]);    //Segundos en decimal
    Time[1] = RTC_MINUTES(Time[1]);    //Minutos en decimal
    Time[2] = RTC_HOUR(Time[2]);       //Hora en decimal
  }

  //Que EQUIPOS se activan segun la hora
  Bits_Equipos_Horario = ACTIVATION_HOURS (Time[1], Time[2]); //Equipo: #7 #6 #5 #4 #3 #2 #1 #0
                                                              //Bits:  MSB.....LSB
  //Que EQUIPOS son forzados a ON u OFF
  Bits_Equipos = RELES_FORZAR(Bits_Equipos_Horario);

  if(Bits_Equipos != Save){       //Solo si hay cambio en el estado de los pines
    Save = Bits_Equipos;
    RELES_ACTIVAR(Bits_Equipos); //Funcion que activa/desactiva los pines del microcontrolador
```

```
    }

    //Delay para hacer tiempo
    delay_ms(10);
    Count++;
    }

    return 0;
}
```

Comunicación Modbus y registros Modbus

```
//Modbus error codes
#define ERROR_FUNCTION      0x01
#define ERROR_ADDRESS       0x02
#define ERROR_VALUE         0x03

//Holdin register define
#define COIL_FLAGS_LEN            10
#define HOLDING_REGISTERS_LEN     105

#define HR_DEVICE_ID              0
#define HR_DEVICE_STATUS          1
#define HR_RTC_SEG                2
#define HR_RTC_MIN                3
#define HR_RTC_HOUR               4
#define HR_RTC_DAY                5
#define HR_RTC_DATE               6
#define HR_RTC_MONTH              7
#define HR_RTC_YEAR               8

#define HR_nEQUIPO                17
#define HR_nDIA                   18
#define HR_nESPECIAL              19
#define HR_ESPECIAL_COUNT         61

//Coils define
#define RELES_RESET_COIL          8
#define RTC_SET_COIL              9

//Declaracion de variables globales
uint8_t EQUIPO_SELECT = 0;     //Variable que selecciona que equipo se desea accesar
de la tabla
uint8_t DIA_SELECT = 1;         //Variable que selecciona el dia se desea accesar de la
tabla
uint8_t ESPECIAL_SELECT = 0;    //Variable que selecciona que dia especial se desea
accesar
uint8_t RTC_DATA[7] = {0,0,0,1,1,1,0};      //Buffer al qeu se fija la hora del reloj


UINT16 evaluateModbusData(UINT8 *dataIn, UINT16 len, UINT8 *dataOut){
   if(modbusCRC(dataIn,len) == 0){
```

```
      switch(dataIn[1]){
        case 1:
          return modbusError(dataIn, dataOut, ERROR_FUNCTION);
        case 2:
          return modbusError(dataIn, dataOut, ERROR_FUNCTION);
        case 3:
          return modbusF03(dataIn, dataOut);
        case 4:
          return modbusError(dataIn, dataOut, ERROR_FUNCTION);
        case 5:
          return modbusF05(dataIn, dataOut);
        case 6:
          return modbusF06(dataIn, dataOut);
        default:
          return modbusError(dataIn, dataOut, ERROR_FUNCTION);
      }
    }
    return 0;
}

UINT16 modbusF03(UINT8 *dataIn, UINT8 *dataOut){
    UINT16_VAL address, length, crc;
    UINT8 i;
    uint8_t tempReg = 0;

    address.v[1] = dataIn[2];
    address.v[0] = dataIn[3];
    length.v[1] = dataIn[4];
    length.v[0] = dataIn[5];

    if(address.Val<HOLDING_REGISTERS_LEN                                    &&
(address.Val+length.Val)<=HOLDING_REGISTERS_LEN && length.Val>0){
      // Holding registers
      for(i = 0; i < length.Val; i++){
        tempReg = getHoldingRegister(address.v[0] + i);
        dataOut[3 + 2*i] = 0;          //Data HI
        dataOut[3 + 2*i + 1] = tempReg;  //Data LO
      }

    }else{
      return modbusError(dataIn, dataOut, ERROR_ADDRESS);
    }
```

```
    // Response
    dataOut[0] = dataIn[0];
    dataOut[1] = dataIn[1];
    dataOut[2] = length.Val*2;
    crc.Val = modbusCRC(dataOut, length.Val*2 + 3);
    dataOut[length.Val*2 + 3] = crc.v[0];   //CRC LO
    dataOut[length.Val*2 + 4] = crc.v[1];   //CRC HI

    return length.Val*2 + 5;
}

UINT16 modbusF05(UINT8 *dataIn, UINT8 *dataOut){
    UINT16_VAL address;
    uint8_t state;

    address.v[1] = dataIn[2];
    address.v[0] = dataIn[3];
    state = dataIn[4];

    if(address.Val<=COIL_FLAGS_LEN){
        if(ValidateCoilRegister(state)){
            setCoil(address.v[0], state);
        }else{
            return modbusError(dataIn, dataOut, ERROR_VALUE);
        }
    }else{
        return modbusError(dataIn, dataOut, ERROR_ADDRESS);
    }

    // Response
    dataOut[0]=dataIn[0];
    dataOut[1]=dataIn[1];
    dataOut[2]=dataIn[2];
    dataOut[3]=dataIn[3];
    dataOut[4]=dataIn[4];
    dataOut[5]=dataIn[5];
    dataOut[6]=dataIn[6];
    dataOut[7]=dataIn[7];
   return 8;

}
```

```
UINT16 modbusF06(UINT8 *dataIn, UINT8 *dataOut){
  UINT16_VAL address, value;

  address.v[1]=dataIn[2];
  address.v[0]=dataIn[3];
  value.v[1]=dataIn[4];
  value.v[0]=dataIn[5];

  if(address.Val<=HOLDING_REGISTERS_LEN){
    if(ValidateHoldingRegister(address.Val, value.Val)){
      setHoldingRegister(address.v[0], value.v[0]);
    }else{
      return modbusError(dataIn, dataOut, ERROR_VALUE);
    }
  }else{
    return modbusError(dataIn, dataOut, ERROR_ADDRESS);
  }

 //  Response
  dataOut[0]=dataIn[0];
  dataOut[1]=dataIn[1];
  dataOut[2]=dataIn[2];
  dataOut[3]=dataIn[3];
  dataOut[4]=dataIn[4];
  dataOut[5]=dataIn[5];
  dataOut[6]=dataIn[6];
  dataOut[7]=dataIn[7];

  return 8;
}

UINT16 modbusError(UINT8 *data, UINT8 *dataOut, UINT8 code){
  UINT16_VAL crc;
  dataOut[0]=data[0];
  dataOut[1]=data[1] | 0x80;
  dataOut[2]=code;
  crc.Val = modbusCRC(dataOut,3);
  dataOut[3]=crc.v[0];
  dataOut[4]=crc.v[1];
  return 5;
}
```

```
UINT16 modbusCRC (UINT8 *data, UINT16 len){
  static const UINT16 wCRCTable[] = {
  0X0000, 0XC0C1, 0XC181, 0X0140, 0XC301, 0X03C0, 0X0280, 0XC241,
  0XC601, 0X06C0, 0X0780, 0XC741, 0X0500, 0XC5C1, 0XC481, 0X0440,
  0XCC01, 0X0CC0, 0X0D80, 0XCD41, 0X0F00, 0XCFC1, 0XCE81, 0X0E40,
  0X0A00, 0XCAC1, 0XCB81, 0X0B40, 0XC901, 0X09C0, 0X0880, 0XC841,
  0XD801, 0X18C0, 0X1980, 0XD941, 0X1B00, 0XDBC1, 0XDA81, 0X1A40,
  0X1E00, 0XDEC1, 0XDF81, 0X1F40, 0XDD01, 0X1DC0, 0X1C80, 0XDC41,
  0X1400, 0XD4C1, 0XD581, 0X1540, 0XD701, 0X17C0, 0X1680, 0XD641,
  0XD201, 0X12C0, 0X1380, 0XD341, 0X1100, 0XD1C1, 0XD081, 0X1040,
  0XF001, 0X30C0, 0X3180, 0XF141, 0X3300, 0XF3C1, 0XF281, 0X3240,
  0X3600, 0XF6C1, 0XF781, 0X3740, 0XF501, 0X35C0, 0X3480, 0XF441,
  0X3C00, 0XFCC1, 0XFD81, 0X3D40, 0XFF01, 0X3FC0, 0X3E80, 0XFE41,
  0XFA01, 0X3AC0, 0X3B80, 0XFB41, 0X3900, 0XF9C1, 0XF881, 0X3840,
  0X2800, 0XE8C1, 0XE981, 0X2940, 0XEB01, 0X2BC0, 0X2A80, 0XEA41,
  0XEE01, 0X2EC0, 0X2F80, 0XEF41, 0X2D00, 0XEDC1, 0XEC81, 0X2C40,
  0XE401, 0X24C0, 0X2580, 0XE541, 0X2700, 0XE7C1, 0XE681, 0X2640,
  0X2200, 0XE2C1, 0XE381, 0X2340, 0XE101, 0X21C0, 0X2080, 0XE041,
  0XA001, 0X60C0, 0X6180, 0XA141, 0X6300, 0XA3C1, 0XA281, 0X6240,
  0X6600, 0XA6C1, 0XA781, 0X6740, 0XA501, 0X65C0, 0X6480, 0XA441,
  0X6C00, 0XACC1, 0XAD81, 0X6D40, 0XAF01, 0X6FC0, 0X6E80, 0XAE41,
  0XAA01, 0X6AC0, 0X6B80, 0XAB41, 0X6900, 0XA9C1, 0XA881, 0X6840,
  0X7800, 0XB8C1, 0XB981, 0X7940, 0XBB01, 0X7BC0, 0X7A80, 0XBA41,
  0XBE01, 0X7EC0, 0X7F80, 0XBF41, 0X7D00, 0XBDC1, 0XBC81, 0X7C40,
  0XB401, 0X74C0, 0X7580, 0XB541, 0X7700, 0XB7C1, 0XB681, 0X7640,
  0X7200, 0XB2C1, 0XB381, 0X7340, 0XB101, 0X71C0, 0X7080, 0XB041,
  0X5000, 0X90C1, 0X9181, 0X5140, 0X9301, 0X53C0, 0X5280, 0X9241,
  0X9601, 0X56C0, 0X5780, 0X9741, 0X5500, 0X95C1, 0X9481, 0X5440,
  0X9C01, 0X5CC0, 0X5D80, 0X9D41, 0X5F00, 0X9FC1, 0X9E81, 0X5E40,
  0X5A00, 0X9AC1, 0X9B81, 0X5B40, 0X9901, 0X59C0, 0X5880, 0X9841,
  0X8801, 0X48C0, 0X4980, 0X8941, 0X4B00, 0X8BC1, 0X8A81, 0X4A40,
  0X4E00, 0X8EC1, 0X8F81, 0X4F40, 0X8D01, 0X4DC0, 0X4C80, 0X8C41,
  0X4400, 0X84C1, 0X8581, 0X4540, 0X8701, 0X47C0, 0X4680, 0X8641,
  0X8201, 0X42C0, 0X4380, 0X8341, 0X4100, 0X81C1, 0X8081, 0X4040 };

  UINT8 nTemp;
  UINT16 wCRCWord = 0xFFFF;

  while (len--){
   nTemp = *data++ ^ wCRCWord;
   wCRCWord >>= 8;
```

```
    wCRCWord  ^= wCRCTable[nTemp];
  }
  return wCRCWord;
}

BOOL ValidateHoldingRegister(UINT16 address, INT16 value){
  if(value>=0 && value<=255){
    switch(address){
      case HR_DEVICE_ID:
        return true;
      case HR_DEVICE_STATUS:
        return true;
      case HR_RTC_SEG:
        if(value<=59){return true;}
        break;
      case HR_RTC_MIN:
        if(value<=59){return true;}
        break;
      case HR_RTC_HOUR:
        if(value<=23){return true;}
        break;
      case HR_RTC_DAY:
        if(value>=1 && value<=7){return true;}
        break;
      case HR_RTC_DATE:
        if(value>=1 && value<=31){return true;}
        break;
      case HR_RTC_MONTH:
        if(value>=1 && value<=12){return true;}
        break;
      case HR_RTC_YEAR:
        if(value<=99){return true;}
        break;
      case HR_ESPECIAL_COUNT:
        if(value<=90){return true;}
        break;
      case HR_nEQUIPO:
        if(value<=7){return true;}
        break;
      case HR_nDIA:
        if(value>=1 && value<=7){return true;}
        break;
```

```
        case HR_nESPECIAL:
          if(value<=90){return true;}
          break;
        default:
          if(address>=20 && address<=60){     //Si es un registro de rutina
            return true;
          }
          if(address>=62 && address<=105){    //Si es un registro de dia especial
            return true;
          }
          if (address>=9 && address<=16){     //Si es un rgistro de los byte de control
            if(value == 0x00 || value == 0xF0){return true;}
            break;
          }
      }
    }
    return false;
}

BOOL ValidateCoilRegister(UINT8 state){
    if(state==0x00 || state==0xFF){return true;}
    return false;
}

void setHoldingRegister(uint8_t address, uint8_t value){
    switch(address){
      case HR_DEVICE_ID:
        SAVE_MODBUS_ID(value);
        break;
      case HR_DEVICE_STATUS:
        break;
      case HR_RTC_SEG:
        RTC_DATA[0] = value;
        break;
      case HR_RTC_MIN:
        RTC_DATA[1] = value;
        break;
      case HR_RTC_HOUR:
        RTC_DATA[2] = value;
        break;
      case HR_RTC_DAY:
        RTC_DATA[3] = value;
```

```
        break;
      case HR_RTC_DATE:
        RTC_DATA[4] = value;
        break;
      case HR_RTC_MONTH:
        RTC_DATA[5] = value;
        break;
      case HR_RTC_YEAR:
        RTC_DATA[6] = value;
        break;
      case HR_ESPECIAL_COUNT:
        SaveSpecialCount(value);
        break;
      case HR_nEQUIPO:
        EQUIPO_SELECT = value;
        break;
      case HR_nDIA:
        DIA_SELECT = value;
        break;
      case HR_nESPECIAL:
        ESPECIAL_SELECT = value;
        break;
      default:
        if(address>=20 && address<=60){     //Si es un registro de rutina
          SaveData_rutina(EQUIPO_SELECT, DIA_SELECT, address, value);
        }
        if(address>=62 && address<=105){    //Si es un registro de dia especial
          SaveData_especial(EQUIPO_SELECT, ESPECIAL_SELECT, address, value);
        }
        if (address>=9 && address<=16){     //Si es un rgistro de los byte de control
          SaveDada_control(address,value);
        }
        break;
    }
}

uint8_t getHoldingRegister(uint8_t address){
  switch(address){
    case HR_DEVICE_ID:
      return MODBUS_ID();
    case HR_DEVICE_STATUS:
      return GET_STATUS();
```

```
      case HR_RTC_SEG:
        return RTC_DATA[0];
      case HR_RTC_MIN:
        return RTC_DATA[1];
      case HR_RTC_HOUR:
        return RTC_DATA[2];
      case HR_RTC_DAY:
        return RTC_DATA[3];
      case HR_RTC_DATE:
        return RTC_DATA[4];
      case HR_RTC_MONTH:
        return RTC_DATA[5];
      case HR_RTC_YEAR:
        return RTC_DATA[6];
      case HR_ESPECIAL_COUNT:
        return ReadSpecialCount();
      case HR_nEQUIPO:
        return EQUIPO_SELECT;
      case HR_nDIA:
        return DIA_SELECT;
      case HR_nESPECIAL:
        return ESPECIAL_SELECT;
      default:
        if(address>=20 && address<=60){     //Si es un registro de rutina
          return ReadData_rutina(EQUIPO_SELECT, DIA_SELECT, address);
        }
        if(address>=62 && address<=105){    //Si es un registro de dia especial
          return ReadData_especial(EQUIPO_SELECT, ESPECIAL_SELECT, address);
        }
        if (address>=9 && address<=16){     //Si es un rgistro de los byte de control
          return ReadDada_control(address);
        }
      }

  return 0;
}

void setCoil(uint8_t address, uint8_t state){
  switch(address){
    case RELES_RESET_COIL:
      RESET_RELES();
      break;
```

```
      case RTC_SET_COIL:
        RTC_SET(RTC_DATA[0], RTC_DATA[1], RTC_DATA[2], RTC_DATA[3],
              RTC_DATA[4], RTC_DATA[5], RTC_DATA[6]);
        break;
      default:
        if(address>=0 && address<=7){     //Si es para forzar una salida
          SetReles(address,state);
        }
        break;
      }
}
```

Funciones de interrupción (UART_Rx, UART_Tx, Timer 2)

```
// Timeout timer for UART1
#define RS485_TIMEOUT        3ul // ms original 30 time out del modbus para espacio entre cadenas
#define RS485_PR2            (uint16_t)(_XTAL_FREQ*RS485_TIMEOUT/(8ul*1000ul))

// RS485 Buffers size
#define RS485_RX_BUFFER_SIZE    255
#define RS485_TX_BUFFER_SIZE    255

//Variables globales para la comunicacion Modbus
//RX
uint8_t rs485RxBuffer[RS485_RX_BUFFER_SIZE] = {0};    //Aqui se guardan los datos recibidos
uint16_t rs485RxBufferLen = 0;                  //Longitud actual de datos
bool rs485RxBufferError = false;
bool rs485RxTimeout = false;

//TX
uint8_t rs485TxBuffer[RS485_TX_BUFFER_SIZE] = {0};    //Aqui se guardan los datos a enviar
uint16_t rs485TxBufferLen = 0;                  //Longitud de los datos a enviar
uint8_t rs485TxState = 0;


void UART1_Initialize (void){
  //INICIALIZAR EL UART 1

  U1MODE = 0x8008;
  U1STA = 0x2000;
  //BaudRate = 9600; Frequency = 16000000 Hz; BRG 416;
  U1BRG = 0x01A0;

  U1STAbits.UTXEN = 1;
  IFS0bits.U1TXIF = 0;    //Bandera de Tx
  IEC0bits.U1TXIE = 1;    //Habilita interrupcion por Tx
  IFS0bits.U1RXIF = 0;    //Bandera de Rx
  IEC0bits.U1RXIE = 1;    //Habilita interrupcion por Rx

  //U1MODEbits.UARTEN = 1;  // enabling UART ON bit
```

```
  //INICIALIZAR EL TIMER 2
  //TMR2 0;
   TMR2 = 0x0000;
   //Period = Time out modbus; Frequency = 16000000 Hz; PR2 =
_XTAL_FREQ*RS485_TIMEOUT/(8ul*1000ul);
   PR2 = RS485_PR2;

   T2CONbits.TCKPS = 1;    //1:8
   IFS0bits.T2IF = false;  //Limpiar bandera
   IEC0bits.T2IE = true;   //Habilitar interrupcion del timer 2
   T2CONbits.TON = 1;

}

void __attribute__ ( ( interrupt, no_auto_psv ) ) _U1TXInterrupt ( void )
{

   static uint16_t txi=0;  //Numero de Bytes se han nenviado

   if((txi < rs485TxBufferLen) && (txi < RS485_TX_BUFFER_SIZE)){
      U1TXREG = rs485TxBuffer[txi++];
   }else{
      while(!U1STAbits.TRMT); //Espera que se emvie el ultimo dato
      RS485_TOGLE = 0;       //RS485 modo Rx
      txi = 0;
      rs485TxBufferLen = 0;
   }
   IFS0bits.U1TXIF = false;    //Limpia la bandera de interrupción
}

void __attribute__ ( ( interrupt, no_auto_psv ) ) _U1RXInterrupt( void )
{

   while(U1STAbits.URXDA == 1)
   {
      unsigned char data = U1RXREG;
      TMR2=0;                       // Clear timeout timer

      if(rs485RxBufferLen < RS485_RX_BUFFER_SIZE)
         rs485RxBuffer[rs485RxBufferLen++] = data;
      else
         rs485RxBufferError = true;    // Buffer overflow (hardware)
```

```
      if(U1STAbits.OERR){              // Buffer overflow (hardware)
        U1STAbits.OERR = 0;
        rs485RxBufferError = true;
      }

  }
  if(!U1STAbits.URXDA){
    IFS0bits.U1RXIF = false;
  }
}

void __attribute__ ( ( interrupt, no_auto_psv ) ) _U1ErrInterrupt ( void )
{

  if ((U1STAbits.OERR == 1))
  {
    U1STAbits.OERR = 0;
    rs485RxBufferError=true;    //Reportar error
  }

  IFS4bits.U1ERIF = false; //
}

void __attribute__ ( ( interrupt, no_auto_psv ) ) _T2Interrupt (  )
{

  IFS0bits.T2IF = 0;  //Limpiar bandera de interrupcion
  if(rs485RxBufferLen > 0){

    if(rs485RxBufferError){
      // Clear the buffer
      rs485RxBufferError = false;
    }else{
      IEC0bits.U1RXIE=0;  //Off UART RX
      T2CONbits.TON = 0;  //Off Timer2
      LED = 1;          //LED on

      if(rs485RxBuffer[0] == MODBUS_ID()){

        rs485TxBufferLen = evaluateModbusData(rs485RxBuffer, rs485RxBufferLen,
rs485TxBuffer);
```

```
            RS485_TOGLE = 1;       //RS485 modo TX
            IFS0bits.U1TXIF = 1;    //Forzar interrupcion de envio, Transmite respuesta
          }
        }
        LED = 0;    //LED Off
        rs485RxBufferLen = 0;
        // Enable RX & timeout timer
        IEC0bits.U1RXIE = 1;
        T2CONbits.TON = 1;

    }
    TMR2 = 0;
}
```

Manejo de DATA de la memoria

```
//DECLARACION DDE VARIABLES
int n; //Numero del equipo a accesar [0 ... 7]
int m; //Numero del evento a accesar por equipo [0 ... 9]

//DEFINICION DE LAS POSCIONES INICIALES DE MEMORIA
//PARA ACCEDER A UNA POSICION DE MEMORIA EXPESIFICA SE DEBEN SUMAR
//Ej: para acceder al dia martes del equipo #5 de la rutina senal
//Posicion = Address_rutina + Equipo_5 + Ma

//Posiciones iniciales de cada segemento de memoria
#define Address_ModbusID          0x0000   //Indica Byte de la direccion modbus de esta remota
#define Address_control          0x0028   //Indica Byte de control del equipo #0
#define Address_rutina          0x0032   //Indica N° de eventos del dia lunes del equipo #0
#define Address_especiales        0x0960   //Indica N° de dias especiales

#define Longitud_Dias            41      //Bytes entre dias o equipos de dias especiales
#define Longitud_Equipo           290     //Bytes entre equipos rutina
#define Longitud_Especial         335     //Bytes entre dias especiales

//ESTRUCTURAS DE DATOS PARA EL MANEJO HORARIO

typedef struct {
   uint8_t Hora;
   uint8_t Mins;
}Hour;

//Estructura de datos
struct {
   uint8_t Byte_Control;
   uint8_t n_Eventos;
   Hour Activar[10];
   Hour Desactivar[10];
}Equipo[8];

//Variables Globales
uint8_t ModbusID; //Direccion modbus de la remota

//FUNCIONES PARA EL MANEJO DE LA ESTRUCTURA DE DATOS HORARIA
```

```
void CONTROL_BYTES (void){
  uint8_t Data[1];
  uint16_t ADDRESS;

  for (n = 0; n < 8; n++){              //Desde el EQUIPO 0 al 7
    ADDRESS = Address_control + n;      //Byte de control de cada equipo
    MEM_LEER(0b1010000,ADDRESS,1,Data);  //Leer el Byte de cotrol
    Equipo[n].Byte_Control = Data[0];    //Almacenar en la estructura de datos
  }
}

void STRUCTURE_DAY (uint8_t Dia){
  //Variables locales
  uint8_t Info[1];
  uint8_t Data[40];
  uint16_t ADDRESS;                     //Posicion del Byte a leer de la memoria


  for (n = 0; n < 8; n++){              //Desde el EQUIPO 0 al 7
    ADDRESS = Address_rutina + (n*Longitud_Equipo);//Posicion inicial de cada equipo en la memoria                                  //290 es la cantidad de bytes de cada equipo

    if (Equipo[n].Byte_Control == 0xF0){
      ADDRESS = ADDRESS + Dia*Longitud_Dias - Longitud_Dias;//Posicion de la cantidad de eventos del "Dia"
      MEM_LEER(0b1010000,ADDRESS,1,Info);       //Leer cantidad de eventos del "Dia"

      if (Info[0] == 0xFF){
        Info[0] = 0;
      }

      if(Info[0] > 10){
        Equipo[n].n_Eventos = 10;           //Limitar la cantidad de eventos a 10
      }else{
        Equipo[n].n_Eventos = Info[0];      //Almacenar en la estructura de datos
      }

      for(m = 0; m < 40; m++){              //Vaciar el arreglo Data
        Data[m] = 0;
      }
```

```
        ADDRESS++;                              //Posicion inicial de los eventos
        MEM_LEER(0b1010000,ADDRESS,4*Equipo[n].n_Eventos,Data);  //Leer todos los Bytes de eventos

        for (m = 0; m < Equipo[n].n_Eventos; m++){          //Almacenar en la estructura de datos
          Equipo[n].Activar[m].Hora = Data[4*m];
          Equipo[n].Activar[m].Mins = Data[4*m+1];
          Equipo[n].Desactivar[m].Hora = Data[4*m+2];
          Equipo[n].Desactivar[m].Mins = Data[4*m+3];
        }
      }
   }
}

void STRUCTURE_SPECIAL (uint16_t Address){
   //Vaiables locales
   uint8_t Info[1];
   uint8_t Data[40];
   uint16_t ADDRESS;                    //Posicion del Byte a leer de la memoria


   for(n = 0; n < 8; n++){              //Desde el EQUIPO 0 al 7
      ADDRESS = Address + 2;

      if (Equipo[n].Byte_Control == 0xF0){
        ADDRESS = ADDRESS + n*Longitud_Dias;          //Posicion de la cantidad de eventos del "Dia"
        MEM_LEER(0b1010000,ADDRESS,1,Info);           //Leer cantidad de eventos del "Dia"

        if (Info[0] == 0xFF){
          Info[0] = 0;
        }

        if(Info[0] > 10){
          Equipo[n].n_Eventos = 10;             //Limitar la cantidad de eventos a 10
        }else{
          Equipo[n].n_Eventos = Info[0];        //Almacenar en la estructura de datos
        }
```

```
        ADDRESS++;                              //Posicion inicial de los eventos
        MEM_LEER(0b1010000,ADDRESS,4*Equipo[n].n_Eventos,Data);  //Leer todos
los eventos

        for (m = 0; m < Equipo[n].n_Eventos; m++){          //Almacenar en la estructura
de datos
          Equipo[n].Activar[m].Hora = Data[4*m];
          Equipo[n].Activar[m].Mins = Data[4*m+1];
          Equipo[n].Desactivar[m].Hora = Data[4*m+2];
          Equipo[n].Desactivar[m].Mins = Data[4*m+3];
        }
      }
    }
}

void STRUCTURE_FILL (uint8_t Dia, uint8_t Fecha, uint8_t Mes){
  //Variables locales
  uint8_t Info[1];
  uint8_t Date[2];
  uint16_t ADDRESS;                 //Posicion del Byte a leer de la memoria
  bool Dia_Especial = 0;            //Bandera de dia especial


  ADDRESS = Address_especiales;         //Posicion de la cantidad de dias especiales
  MEM_LEER(0b1010000,ADDRESS,1,Info);    //Leer cantidad de dias especiales

  if (Info[0] == 0xFF){
    Info[0] = 0;
  }
  if(Info[0] > 120){
    Info[0] = 120;                  //Limitar la cantidad de dias especiales a 120
  }

  ADDRESS = Address_especiales + 1;
  for(n = 0; n < Info[0]; n++){         //Desde cero hasta la cantidad de dias especiales
    ADDRESS = ADDRESS + n*Longitud_Especial;//Posicion de cada fecha especial
    MEM_LEER(0b1010000,ADDRESS,2,Date); //Leer el mes y la fecha

    if( (Mes == Date[0])&&(Fecha == Date[1]) ){
      Dia_Especial = 1;
      break;
    }
```

```
  }

  if(Dia_Especial == 1){
    STRUCTURE_SPECIAL(ADDRESS);    //Llenar la estructura de datos con la
    Dia_Especial = 0;              //informacion del dia especial
  }else{
    STRUCTURE_DAY(Dia);            //Llenar la estructura de datos con la
  }                                //informacion del dia de la semana
}

uint8_t ACTIVATION_HOURS (uint8_t Minutos, uint8_t Hora){
  uint8_t Bit_Equipo[8] = { 0b00000001, 0b00000010,
                    0b00000100, 0b00001000,
                    0b00010000, 0b00100000,
                    0b01000000, 0b10000000 };

  uint8_t Activar = 0;

  for(n = 0; n < 8; n++){    //Desde el Equipo 0 al 7

    if(Equipo[n].Byte_Control == 0xF0){            //Verificasi el equipo esta conectado
y activo
      for(m = 0; m < Equipo[n].n_Eventos; m++){   //Desde el Evento 0 al n

        //Si la hora de activacion es menor que la de apagado
        //3 casos pueden ocurrir
        if(Equipo[n].Activar[m].Hora < Equipo[n].Desactivar[m].Hora){

          //Caso 1: La hora actual igual a la hora de activacion
          if(Hora == Equipo[n].Activar[m].Hora){
            if(Minutos >= Equipo[n].Activar[m].Mins){ //Se verifican si los minutos
para decidir si se activa
              Activar = Activar | Bit_Equipo[n];    //Activar el equipo n
            }
          }

          //Caso 2 : La hora actua les mayor que la de activacion y menor que la de
apagado
          if((Hora            >            Equipo[n].Activar[m].Hora)&&(Hora            <
Equipo[n].Desactivar[m].Hora)){
            Activar = Activar | Bit_Equipo[n];  //Siempre se activa el equipo en este
caso
```

```
            }

            //Caso 3: La hora actual es igual a la hora de apagado
            if(Hora == Equipo[n].Desactivar[m].Hora){
              if(Minutos < Equipo[n].Desactivar[m].Mins){ //Se verifican los minutos
para decidir hasat cuando estara activo
                Activar = Activar | Bit_Equipo[n];     //Activar el equipo n
              }
            }
          }

          //Si la hora de Activacion y apagado son iguales
          if(Equipo[n].Activar[m].Hora == Equipo[n].Desactivar[m].Hora){
            if(Hora == Equipo[n].Activar[m].Hora){
              //Se verifican solo los minutos en los que estara activo el equipo
              if((Minutos >= Equipo[n].Activar[m].Mins)&&(Minutos <
Equipo[n].Desactivar[m].Mins)){
                Activar = Activar | Bit_Equipo[n];  //Activar el equipo n
              }
            }
          }

        }
      }
    }

    return Activar; //La funcion regresa un numero de 8 bits, cada bit en 1 indica que
equipo se activa
}

//FUNCIONES PARA EL MANEJO DE LOS REGISTROS MODBUS
void SAVE_MODBUS_ID (uint8_t ID){
    uint8_t data[1];

    data[0] = ID;
    ModbusID = ID;
    MEM_ESCRIBIR(0b1010000,Address_ModbusID,1,data);
    //DELAY
}

void GET_MODBUS_ID (void){
    uint8_t data[1];
```

```
  MEM_LEER(0b1010000,Address_ModbusID,1,data);
  ModbusID = data[0];
}

uint8_t MODBUS_ID(void){
  return ModbusID;
}

void SaveDada_control(uint8_t address, uint8_t value){
  uint8_t Data[1];
  uint16_t ADDRESS;

  Data[0] = value;
  ADDRESS = Address_control + address - 9;   //Byte de control de cada equipo
  MEM_ESCRIBIR(0b1010000,ADDRESS,1,Data);    //Escribir el Byte de cotrol
  //DELAY
  Equipo[address - 9].Byte_Control = value;           //Almacenar en la estructura de
datos

}

uint8_t ReadDada_control(uint8_t address){
  uint8_t Data[1];
  uint16_t ADDRESS;

  ADDRESS = Address_control + address - 9;   //Byte de control de cada equipo
  MEM_LEER(0b1010000,ADDRESS,1,Data);       //Leer el Byte de cotrol
  return Data[0];
}

void SaveData_rutina(uint8_t equipo, uint8_t dia, uint8_t address, uint8_t value){
  uint8_t data[1];
  uint16_t ADDRESS;                    //Posicion del Byte a guardar en la memoria

  if(dia>0 && dia<=7 && equipo>=0 && equipo<=7){
    data[0] = value;
    ADDRESS = Address_rutina + equipo*Longitud_Equipo + dia*Longitud_Dias -
Longitud_Dias; //Posicion del dia en el equipo seleccionado
    ADDRESS = ADDRESS + address - 20;       //Posicion exata en la meoria, -20 es la
traduccion del registro registro Modbus a EEPROM
    MEM_ESCRIBIR(0b1010000,ADDRESS,1,data);
```

```
      //DELAY
    }
}

uint8_t ReadData_rutina(uint8_t equipo, uint8_t dia, uint8_t address){
   uint8_t data[1];
   uint16_t ADDRESS;                    //Posicion del Byte a guardar en la memoria

   if(dia>0 && dia<=7 && equipo>=0 && equipo<=7){
      ADDRESS = Address_rutina + equipo*Longitud_Equipo + dia*Longitud_Dias -
Longitud_Dias; //Posicion del dia en el equipo seleccionado
      ADDRESS = ADDRESS + address - 20;       //Posicion exata en la meoria, -20 es la
traduccion del registro registro Modbus a EEPROM
      MEM_LEER(0b1010000,ADDRESS,1,data);
      return data[0];
   }
   return 0;
}

void SaveSpecialCount(uint8_t value){
   uint8_t Data[1];
   Data[0] = value;
   MEM_ESCRIBIR(0b1010000,Address_especiales,1,Data);       //Escribir el Byte de
cotrol
   //DELAY
}

uint8_t ReadSpecialCount(void){
   uint8_t Data[1];
   MEM_LEER(0b1010000,Address_especiales,1,Data);       //Leer el Byte de cotrol
   return Data[0];
}

void SaveData_especial(uint8_t equipo, uint8_t nDiaEsp, uint8_t address, uint8_t
value){
   uint8_t data[1];
   uint16_t ADDRESS;                    //Posicion del Byte a leer de la memoria

   data[0] = value;
   if(nDiaEsp>=0 && nDiaEsp<=120 && equipo>=0 && equipo<=7){
      if(address < 64){
```

```
        ADDRESS = Address_especiales + 1 + nDiaEsp*Longitud_Especial; //Posicion del equipo seleccionado del dia especial
        ADDRESS = ADDRESS + address - 62;      //Posicion exata en la meoria, -62 es la traduccion del registro registro Modbus a EEPROM
      }else{
        ADDRESS = Address_especiales + 1 + nDiaEsp*Longitud_Especial + equipo*Longitud_Dias + 2; //Posicion del equipo seleccionado del dia especial
        ADDRESS = ADDRESS + address - 64;      //Posicion exata en la meoria, -64 es la traduccion del registro registro Modbus a EEPROM
      }
      MEM_ESCRIBIR(0b1010000,ADDRESS,1,data);
      //DELAY
    }
}

uint8_t ReadData_especial(uint8_t equipo, uint8_t nDiaEsp, uint8_t address){
   uint8_t data[1];
   uint16_t ADDRESS;                    //Posicion del Byte a leer de la memoria

   if(nDiaEsp>=0 && nDiaEsp<=120 && equipo>=0 && equipo<=7){
      if(address < 64){
        ADDRESS = Address_especiales + 1 + nDiaEsp*Longitud_Especial; //Posicion del equipo seleccionado del dia especial
        ADDRESS = ADDRESS + address - 62;      //Posicion exata en la meoria, -62 es la traduccion del registro registro Modbus a EEPROM
      }else{
        ADDRESS = Address_especiales + 1 + nDiaEsp*Longitud_Especial + equipo*Longitud_Dias + 2; //Posicion del equipo seleccionado del dia especial
        ADDRESS = ADDRESS + address - 64;      //Posicion exata en la meoria, -64 es la traduccion del registro registro Modbus a EEPROM
      }
      MEM_LEER(0b1010000,ADDRESS,1,data);
      return data[0];
   }
   return 0;
}

//OTRAS RUTINAS PARA FUTURO USO
void MEMORY_ERASE(void){
   uint16_t address = 0x0000;
   uint8_t clear[1] = {0xFF};
```

```
    for(address = 0x0000; address < 0x7FFF; address++){
      MEM_ESCRIBIR(0b1010000,address,1,clear);
      delay_ms(10);
    }
}
```

Manejo de RELES

```
uint8_t Coils[8] = {0xFF,0xFF,0xFF,0xFF,
                    0xFF,0xFF,0xFF,0xFF};          //Coils; 0xFF=sin forzar, 0x00=forzar a cero,0xF0=forzar a uno
uint8_t Bits[8] = { 0b00000001, 0b00000010,
                    0b00000100, 0b00001000,
                    0b00010000, 0b00100000,
                    0b01000000, 0b10000000 };
uint8_t Status = 0;

void SetReles(uint8_t address, uint8_t state){
  if(state == 0xFF){Coils[address] = 0xF0;}
  if(state == 0x00){Coils[address] = 0x00;}

}
void RESET_RELES(void){
  uint8_t n;
  for (n = 0; n < 8; n++){
    Coils[n] = 0xFF;
  }
}

uint8_t RELES_FORZAR (uint8_t Bits_Equipos_Horario){
  uint8_t n;
  uint8_t Activar = 0;

  for (n = 0; n < 8; n++){              //Desde el EQUIPO 0 al 7
    if(Coils[n] == 0x00){
      Activar = Activar & (~Bits[n]);     //Forzar a OFF
    }
    if(Coils[n] == 0xF0){
      Activar = Activar | Bits[n];       //Frozar a ON
    }
    if(Coils[n] == 0xFF){
      Activar = Activar | (Bits_Equipos_Horario & Bits[n]);  //No forzar, modo automatico
    }
  }
  return Activar;
}
```

```
void RELES_ACTIVAR(uint8_t Bits_Equipos){
   int t = 100;
   Status = Bits_Equipos;

   //Equipo conectado al pin 0
   if( (Bits_Equipos & Bits[0]) == Bits[0]){
      PIN_0 = 1;
   }else{
      PIN_0 = 0;
   }
   delay_ms(t);

   //Equipo conectado al pin 1
   if( (Bits_Equipos & Bits[1]) == Bits[1]){
      PIN_1 = 1;
   }else{
      PIN_1 = 0;
   }
   delay_ms(t);

   //Equipo conectado al pin 2
   if( (Bits_Equipos & Bits[2]) == Bits[2]){
      PIN_2 = 1;
   }else{
      PIN_2 = 0;
   }
   delay_ms(t);

   //Equipo conectado al pin 3
   if( (Bits_Equipos & Bits[3]) == Bits[3]){
      PIN_3 = 1;
   }else{
      PIN_3 = 0;
   }
   delay_ms(t);

   //Equipo conectado al pin 4
   if( (Bits_Equipos & Bits[4]) == Bits[4]){
      PIN_4 = 1;
   }else{
      PIN_4 = 0;
   }
```

```
    delay_ms(t);

    //Equipo conectado al pin 5
    if( (Bits_Equipos & Bits[5]) == Bits[5]){
        PIN_5 = 1;
    }else{
        PIN_5 = 0;
    }
    delay_ms(t);

    //Equipo conectado al pin 6
    if( (Bits_Equipos & Bits[6]) == Bits[6]){
        PIN_6 = 1;
    }else{
        PIN_6 = 0;
    }
    delay_ms(t);

    //Equipo conectado al pin 7
    if( (Bits_Equipos & Bits[7]) == Bits[7]){
        PIN_7 = 1;
    }else{
        PIN_7 = 0;
    }
    delay_ms(t);
}

uint8_t GET_STATUS(void){
    return Status;
}
```

Comunicación I^2C

```
//Declaracion de variables globales
 uint8_t clear;

//DEFINICION DE LOS REGISTROS DEL I2C1 USADOS
#define I2C1_TRANSMIT                I2C1TRN            // Defines the transmit register used to send data.
#define I2C1_RECEIVE                 I2C1RCV            // Defines the receive register used to receive data.

//REGISTROS DEL I2C1 USADOS PARA REALIZAR LA COMUNICACION CON LOS PERIFERICOS
#define WRITE_COLLISION          I2C1STATbits.IWCOL       // Colicion al escribir en el buffer de Tx
#define SLAVE_ACK            I2C1STATbits.ACKSTAT    // Verifica que el esclavo ACK
#define MASTER_TRANSMITING    I2C1STATbits.TRSTAT    // Indica si el maestro esta trasmitiendo
#define RX_BUFFER_STATE        I2C1STATbits.RBF        // Idica si el Buffer de Rx esta vacio
#define TX_BUFFER_STATE        I2C1STATbits.TBF        // Idica si el Buffer de Tx esta vacio

#define MASTER_ACK_BIT        I2C1CONbits.ACKDT       // 1=NACK 0=ACK
#define MASTER_ACK          I2C1CONbits.ACKEN      // Envia el ACK al esclavo durante la Rx
#define MASTER_RECEIVE        I2C1CONbits.RCEN       // Habilita la Rx
#define STOP_CONDITION         I2C1CONbits.PEN         // Condicion de parada para la comunicacion
#define RESTART_CONDITION      I2C1CONbits.RSEN        // Condicion de reinicio para la comunicacion
#define START_CONDITION        I2C1CONbits.SEN         // Condicion de inicio para la comunicacion

void I2C1_Initialize(void)
{

   //INICIAL LAS VARIABLES QUE INDICAN EL ESTADO DEL BUFFER

   // initialize the hardware
   // Baud Rate Generator Value: I2CBRG 37; Con 16MHz - I2C frec de 400kHz
   I2C1BRG = 0x0025;
```

```
   // ACKEN disabled; STREN disabled; GCEN disabled; SMEN disabled; DISSLW disabled; I2CSIDL disabled; ACKDT Sends ACK; SCLREL Holds; RSEN disabled; IPMIEN disabled; A10M 7 Bit; PEN disabled; RCEN disabled; SEN disabled; I2CEN enabled;
   I2C1CON = 0x8200;
   // P disabled; S disabled; I2COV disabled; IWCOL disabled;
   I2C1STAT = 0x0000;

   /* MI2C1 - I2C1 Master Events */
   // clear the master interrupt flag
   IFS1bits.MI2C1IF = 0;
   // enable the master interrupt
   IEC1bits.MI2C1IE = 1;
}


void __attribute__ ( ( interrupt, no_auto_psv ) ) _MI2C1Interrupt ( void )
{

  //AGREGAR VARIABLES DE TIPO STATIC PARA USO LOCAL EN LA MAQUINA DE ETSADO

   IFS1bits.MI2C1IF = 0;

   // CHEQUEAR SI HAY COLICION EN EL BUFFER
   // Y DDEJAR LA COMUNICACION EN IDLE

   /* MAQUINA DE ESTADOSPARA IC2 */

}

/* FUNCIONES PARA ESCRIBIR/LEER EN EL RTC POR I2C*/

void RTC_ESCRIBIR( uint8_t Address,
                   uint8_t Reg_Address,
                   uint8_t Longitud,
                   uint8_t Data[])
{
   //Inicializacion de variables locales
   uint8_t contador;

   //Inicio del protocolo I2C
   START_CONDITION = 1;
```

```
  while (START_CONDITION);

  I2C1_TRANSMIT = (Address << 1)&(0b11111110); //Enviar direccion del RTC + W
  while(MASTER_TRANSMITING);

  if (SLAVE_ACK == 0){
    I2C1_TRANSMIT = Reg_Address;        //Enviar direccion de la poscion del registro
    while(MASTER_TRANSMITING);
  }

  if (SLAVE_ACK == 0){
    for (contador = 0; contador < Longitud; contador++){     //Envia cada Byte del
arreglo
      I2C1_TRANSMIT = Data[contador];
      while(MASTER_TRANSMITING);
      if (SLAVE_ACK == 1){break;}
    }
  }

  STOP_CONDITION = 1;
  while(STOP_CONDITION);
}

void RTC_LEER( uint8_t Address,
          uint8_t Reg_Address,
          uint8_t Longitud,
          uint8_t Data[])
{
  //Inicializacion de variables locales
  uint8_t contador;
  clear = I2C1_RECEIVE; //Limpia cualuquier informacion no deceada

  //Inicio del protocolo I2C
  START_CONDITION = 1;
  while (START_CONDITION);

  I2C1_TRANSMIT = (Address << 1)&(0b11111110); //Enviar direccion del equipo + W
  while (MASTER_TRANSMITING);

  if (SLAVE_ACK == 0){
    I2C1_TRANSMIT = Reg_Address;        //"Dummy write" en la psocion de memoria
a leer
```

```
    while (MASTER_TRANSMITING);
  }

  RESTART_CONDITION = 1;
  while(RESTART_CONDITION);

  I2C1_TRANSMIT = (Address << 1)|(0b00000001);    //Direccion del equipo a leer
  while (MASTER_TRANSMITING);

  if (SLAVE_ACK == 0){
    MASTER_ACK_BIT = 0; //ACK
    for (contador = 0; contador < (Longitud-1); contador++){ //Leer cada Byte a partir
de la posicion inicial
      MASTER_RECEIVE = 1;
      while(MASTER_RECEIVE);

      Data[contador] = I2C1_RECEIVE;
      if (MASTER_ACK_BIT == 0){
        MASTER_ACK = 1;
        while(MASTER_ACK);
      }
    }
    MASTER_ACK_BIT = 1; //NACK
    MASTER_RECEIVE = 1;
    while(MASTER_RECEIVE);
    Data[Longitud-1] = I2C1_RECEIVE;

    if (MASTER_ACK_BIT == 1){
        MASTER_ACK = 1;    //El maestro no debe reconocer el ultimo Byte para que
el quipo se detega
        while(MASTER_ACK);
      }
  }

  STOP_CONDITION = 1;    //Condicion de parada
  while(STOP_CONDITION);
}


/* FUNCIONES PARA ESCRIBIR/LEER EN LA EEPROM POR I2C*/

void MEM_ESCRIBIR( uint8_t Address,
```

```
            uint16_t Reg_Address,
            uint8_t Longitud,
            uint8_t Data[])
{
  //Inicializacion de variables locales
  uint8_t contador;

  //Inicio del protocolo I2C
  START_CONDITION = 1;
  while (START_CONDITION);

  I2C1_TRANSMIT = (Address << 1)&(0b11111110); //Direccion de la memoria + W
  while(MASTER_TRANSMITING);

  if (SLAVE_ACK == 0){
    I2C1_TRANSMIT = Reg_Address / 0x100;    // Byte mas significativo de direccion
de memoria
    while(MASTER_TRANSMITING);
  }

  if (SLAVE_ACK == 0){
    I2C1_TRANSMIT = Reg_Address % 0x100;      // Byte menos significativo de
direccion de memoria
    while(MASTER_TRANSMITING);
  }

  if (SLAVE_ACK == 0){
    for (contador = 0; contador < Longitud; contador++){
      I2C1_TRANSMIT = Data[contador];
      while(MASTER_TRANSMITING);
      if (SLAVE_ACK == 1){break;}
    }
  }

  STOP_CONDITION = 1;
  while(STOP_CONDITION);
}

void MEM_LEER( uint8_t Address,
          uint16_t Reg_Address,
          uint8_t Longitud,
          uint8_t Data[])
```

```
{
  //Inicializacion de variables locales
  uint8_t contador;
  clear = I2C1_RECEIVE;  //limpiar informacion no deceada


  //Inicio del protocolo I2C
  START_CONDITION = 1;
  while (START_CONDITION);

  I2C1_TRANSMIT = (Address << 1)&(0b11111110); //Comienzo del Dummy Write
  while(MASTER_TRANSMITING);

  if (SLAVE_ACK == 0){
    I2C1_TRANSMIT = Reg_Address / 0x100;    // Byte mas significativo de direccion
de memoria
    while(MASTER_TRANSMITING);
  }

  if (SLAVE_ACK == 0){
    I2C1_TRANSMIT = Reg_Address % 0x100;      // Byte menos significativo de
direccion de memoria
    while(MASTER_TRANSMITING);
  }

  RESTART_CONDITION = 1;                // Reiniciar para leer
  while(RESTART_CONDITION);

  I2C1_TRANSMIT = (Address << 1)|(0b00000001); //Direccion de la memoria + R
  while(MASTER_TRANSMITING);

  if (SLAVE_ACK == 0){
    MASTER_ACK_BIT = 0; //ACK
    for (contador = 0; contador < (Longitud-1); contador++){
      MASTER_RECEIVE = 1;
      while(MASTER_RECEIVE);
      Data[contador] = I2C1_RECEIVE;

      if (MASTER_ACK_BIT == 0){
        MASTER_ACK = 1;
        while(MASTER_ACK);
      }
```

```
        }

        MASTER_ACK_BIT = 1; //NACK
        MASTER_RECEIVE = 1;
        while(MASTER_RECEIVE);
        Data[Longitud-1] = I2C1_RECEIVE;

        if (MASTER_ACK_BIT == 1){
              MASTER_ACK = 1;
              while(MASTER_ACK);
            }
    }

    STOP_CONDITION = 1;
    while(STOP_CONDITION);
}
```

Función Delay

```
bool INTERRUPT_STATUS = false;


void TMR1_Initialize (void)
{
   //TMR1 0;
   TMR1 = 0x0000;
   //Period = 0.001 s; Frequency = 16000000 Hz; PR1 16000;
   PR1 = 0x3E80; // El timer 1 se desborda cada 1 ms
   //TCKPS 1:1; TON enabled; TSIDL disabled; TCS FOSC/2; TSYNC disabled; TGATE
disabled;
   T1CON = 0x8000;

   IFS0bits.T1IF = false;
   IEC0bits.T1IE = true;
}


void __attribute__ ( ( interrupt, no_auto_psv ) ) _T1Interrupt (  )
{
   /* FUNCION DE INTERRUPCION DEL TIMER 1 */

   INTERRUPT_STATUS = true;

   /* FIN DEL CODIGO DE INTERRUPCION */

   IFS0bits.T1IF = false; //Limpiar bandera de interrupcion
}

void delay_ms (int t){
   unsigned long tempo = 0;
   while (tempo < t){
      if (INTERRUPT_STATUS == true){
         INTERRUPT_STATUS = false;
         tempo++;
      }
   }
}
```