

**CODI DEL PROGRAMA**

```
/*Node ArtNet-DMX amb capacitat per a transmetre dos universos DMX i servidor web
per dur a terme la configuració*/
```

```
#include <capcalera.h>
```

```
/*Inicialització de les variables no volàtils a la memòria EEPROM*/
```

```
uint8_t EEMEM ip_node_eeprom[4] = {2,0,0,100};
uint8_t EEMEM subnet_mask_eeprom[4] = {255,0,0,0};
uint8_t EEMEM gateway_node_eeprom[4] = {2,0,0,1};
uint8_t EEMEM mac_node_eeprom[6] = {0xAA, 0x3D, 0xDE, 0x47, 0xFE, 0xAD};
uint8_t EEMEM net_eeprom = 0;
uint8_t EEMEM subnet_eeprom = 0;
uint8_t EEMEM univers0_eeprom = 0;
uint8_t EEMEM univers1_eeprom = 0;
char EEMEM nom_eeprom[18] = "Node 1 ArtNet-DMX";
```

```
/*Valors de fàbrica del node. En cas de reinici es tornaria a aquests valors*/
```

```
const uint8_t ip_node_fab[4] = {2,0,0,100};
const uint8_t subnet_mask_node_fab[4] = {255,0,0,0};
const uint8_t gateway_node_fab[4] = {2,0,0,1};
const uint8_t mac_node_fab[6] = {0xAA, 0x3D, 0xDE, 0x47, 0xFE, 0xAD};
const uint8_t net_fab = 0;
const uint8_t subnet_fab = 0;
const uint8_t univers0_fab = 0;
const uint8_t univers1_fab = 0;
const char nom_fab[18] = {"Node 1 ArtNet-DMX"};
```

```
/*Variables de configuració de la connexió i del node*/
```

```
unsigned char mac_node[6] = {0xAA, 0x3D, 0xDE, 0x47, 0xFE, 0xAD};
unsigned char ip_node[4] = {2,0,0,100};
unsigned char subnet_mask_node[4] = {255, 0, 0, 0};
unsigned char gateway_node[] = {2,0,0,1};
unsigned char net=0;
unsigned char subnet=0;
unsigned char univers0=0;
unsigned char univers1=0;
char nom[18]={ "Node 1 ArtNet-DMX"};
```

```
/*Variables de configuració del node. Mostren el valor a la web que predrà la
variable un cop s'efectuïn els canvis*/
```

```
uint16_t ip_node_web[4];
uint16_t subnet_mask_node_web[4];
uint16_t gateway_node_web[4];
uint16_t mac_node_web[6];
uint16_t net_web;
uint16_t subnet_web;
uint16_t univers0_web;
uint16_t univers1_web;
char nom_web[18];
```

```
/*IP font dels missatges UDP i TCP rebuts. Els missatges de resposta s'enviaran a
aquesta mateixa direcció*/
```

```
uint8_t ip_remota[4];
uint8_t ip_remota_broad[4];
```

```
/*Variables de recepció i enviament de paquets UDP i TCP*/
```

```
uint16_t restant;
uint16_t offset;
int mida_paquet_udp;
uint8_t buffer_udp [530];
uint16_t mida_paquet_tcp=0;
uint8_t estat_tcp;
uint8_t buffer_tcp[2000];
uint16_t tipus_paquet;

/*Variable de comptatge de temps de reset*/
uint8_t compta=0;

/*Variable que determina quan enviar missatges de tipus ArtPollReply*/
uint8_t talktome=0;

/*Cadena que marca l'inici de tots el paquets del protocol Art-Net*/
const uint8_t artnet_inici[8]={'A','r','t','-','N','e','t',0x00};

char conversio_str[15];
char * index;
int get_present,post_present;

/*Variables relatives a la funció d'unir (merge) paquets ArtDMX*/
volatile uint8_t estat_merge_0=0;
volatile uint8_t estat_merge_1=0;
uint8_t ip_remota_merge_00[4];
uint8_t ip_remota_merge_01[4];
uint8_t ip_remota_merge_10[4];
uint8_t ip_remota_merge_11[4];
volatile uint8_t identificacio_ip_00=0;
volatile uint8_t identificacio_ip_01=0;
uint8_t identificacio_ip_10=0;
uint8_t identificacio_ip_11=0;
uint8_t buffer_udp_merge_00[512];
uint8_t buffer_udp_merge_01[512];
uint8_t buffer_udp_merge_10[512];
uint8_t buffer_udp_merge_11[512];
volatile uint8_t segons_inactivitat_0=0;
volatile uint8_t segons_inactivitat_1=0;
volatile uint8_t torn_inactivitat_0=0;
volatile uint8_t torn_inactivitat_1=0;
uint8_t ip_remota_cancelacio[4];
uint8_t cancela_merge=0;
/*Variables relatives a la transmissió de DMX via el port USART*/

volatile int16_t ptr_dmx_0 = 0;
volatile int16_t ptr_dmx_1= 0;
volatile uint8_t estat_dmx_0 = 0;
volatile uint8_t estat_dmx_1= 0;
uint8_t buffer_dmx0[buffer_max];
uint8_t buffer_dmx1[buffer_max];
uint8_t dmx_llest_0=0;
uint8_t dmx_llest_1=0;

/*Variables de les estructures dels missatges del protocol Art-Net, fetes servir
per a la transmissió i interpretació d'aquests*/

paquet_pollreply_t paquet_pollreply;
paquet_pollreply_t *paquet_pollreply_p;

paquet_artaddress_t paquet_artaddress;
paquet_artaddress_t *paquet_artaddress_p;

paquet_artipprog_t paquet_artipprog;
paquet_artipprog_t *paquet_artipprog_p;

paquet_ipprogreply_t paquet_ipprogreply;
paquet_ipprogreply_t *paquet_ipprogreply_p;
```

```
int conta_prova=0;
int conta_prova1=0;

int main(void)
{
    _delay_ms(100);
    /*Els pins no fets servir es configuren com a sortides a nivell baix*/
    DDRA=0xFF;
    DDRB=0xFF;
    DDRC=0x7F;
    DDRD=0xFF;
    PORTA=0x00;
    PORTB=0x00;
    PORTC=0x00;
    PORTD=0x00;
    /*Seqüència de reset necessaria per inicialitzar el mòdul d'Ethernet*/
    PORTB=1;
    _delay_ms(1);
    PORTB=0;
    _delay_ms(1);
    PORTB=1;

    /*Inicialització del xip d'ethernet i els ports USART*/
    iniciar_dades_web();
    iniciar_sockets();
    iniciar_usart();
    /*S'habiliten les interrupcions globals*/
    sei();

    /*Configuració de les interrupcions dels timers*/
    TCCR1B|=(1<<WGM12);
    OCR1A=15624;
    TCCR1B|=((1<<CS12) | (1<<CS10));
    TCCR3B|=(1<<WGM32);
    OCR3A=15624;
    TCCR3B|=((1<<CS32) | (1<<CS30));

    /*Inicialització de buffers*/
    memset(buffer_udp_merge_00,0,512);
    memset(buffer_udp_merge_01,0,512);
    memset(buffer_udp_merge_10,0,512);
    memset(buffer_udp_merge_11,0,512);

    for(;;)
    {
        /*Si s'ha premut el polsador de reset, es retorna el node als valors de fàbrica*/
        /*Si s'ha premut durant aproximadament 3 segons el polsador de reset, es retorna el
        node als valors de fàbrica*/
        if((PINC&(1<<PINC7))==0)
        {
            while((PINC&(1<<PINC7))==0)
            {
                _delay_ms(500);
                compta++;
            }
            if (compta>6)
            {
                reinici_valors();
            }
            while((PINC&(1<<PINC7))==0)
            {
            }
        }
        /*Es deixa passar un quart de segon per evitar el rebot del polsador*/
        _delay_ms(250);
        compta=0;
    }
}
```

```

}
/*Es comprova si s'ha rebut un paquet UDP i si es de més de 14 bytes (missatge
d'Art-net més petit) es passa a analitzar*/
mida_paquet_udp = extreur_udp();
if(mida_paquet_udp>13)
{
/*Es comprova el tipus de missatge Art-Net i es procedeix en funció d'aquest*/
tipus_paquet= comprovar_tipus();
/*En cas de que sigui un missatge de tipus ArtDMX es comprova si el port és un dels
dos configurats al node i en cas afirmatiu es transmet la trama DMX via el port
USART*/
if ((tipus_paquet==codi_artdmx) && (mida_paquet_udp==mida_artdmx))
{
/*Si el paquet està dirigit al univers del port 0*/
if(((buffer_udp[14]==(univers0+(subnet<<4))) && buffer_udp[15]==net))
{
if(cancela_merge==0 || memcmp(ip_remota,ip_remota_cancelacio,4)==0)
{
if(dmx_llest_0==0)
{

if(identificacio_ip_00==0)
{
memcpy(ip_remota_merge_00,ip_remota,4);
identificacio_ip_00=1;
}
if((memcmp(ip_remota,ip_remota_merge_00,4)!=0) & (identificacio_ip_00==1) &
(identificacio_ip_01==0))
{
estat_merge_0=estat_merge_0+0x08;
memcpy(ip_remota_merge_01,ip_remota,4);
identificacio_ip_01=1;
}

if(estat_merge_0>=0x08)
{
/*S'habilita l'interruptió de comparació del timer 1*/
TIMSK1|=(1<<OCIE1A);
/*mode HTP de merge*/
if((estat_merge_0 & 0x02)==0)
{
if(memcmp(ip_remota,ip_remota_merge_00,4)==0)
{
if(torn_inactivitat_0==1)
{
segons_inactivitat_0=0;
torn_inactivitat_0=0;
}
for(int i=0;i<512;i++)
{
if(buffer_udp[i+18]>=buffer_udp_merge_01[i])
{
buffer_dmx0[i]=buffer_udp[i+18];
}
memcpy(buffer_udp_merge_00,buffer_udp+18,512);
}
}
if(memcmp(ip_remota,ip_remota_merge_01,4)==0)
{
if(torn_inactivitat_0==0)
{
segons_inactivitat_0=0;
torn_inactivitat_0=1;
}
for(int i=0;i<512;i++)
{
if(buffer_udp[i+18]>=buffer_udp_merge_00[i])
{

```

```
buffer_dmx0[i]=buffer_udp[i+18];
}
memcpy(buffer_udp_merge_01,buffer_udp+18,512);
}
}

}else
{
/*Mode de merge LTP no implementat*/
}
/*si es compten 10 segons seguits, es desactiva el mode merge i l'interrupció*/
if(segons_inactivitat_0>=10)
{

estat_merge_0=0;
identificacio_ip_00=0;
identificacio_ip_01=0;
TIMSK1&=~(1<<OCIE1A);
torn_inactivitat_0=0;
segons_inactivitat_0=0;
}
}else
{
memcpy(buffer_dmx0, &buffer_udp[18],512);
}
dmx_llest_0=1;
}

if (estat_dmx_0==0)
{
conta_prova++;
if (conta_prova==10)
{
inici_paquet_udp(ip_remota, port_artnet);
escriure_udp(0);
terminar_paquet_udp();
conta_prova=0;
}

dmx_llest_0=0;
envia_dmx_0();
}
}

/*Si el paquet ArtDMX està dirigit a l'univers el port 1*/
if ((buffer_udp[14]==(univers1+(subnet<<4))) && buffer_udp[15]==net)
{
if (cancela_merge==0 || memcmp(ip_remota,ip_remota_cancelacio,4)==0)
{
if (dmx_llest_1==0)
{

if (identificacio_ip_10==0)
{
memcpy(ip_remota_merge_10,ip_remota,4);
identificacio_ip_10=1;
}
if ((memcmp(ip_remota,ip_remota_merge_10,4)!=0) & (identificacio_ip_10==1) &
(identificacio_ip_11==0))
{
estat_merge_1=estat_merge_1+0x08;
memcpy(ip_remota_merge_11,ip_remota,4);
identificacio_ip_11=1;
}

if (estat_merge_1>=0x08)
{
```

```

TIMSK3|=(1<<OCIE3A);
if((estat_merge_1 & 0x02)==0)
{
if(memcmp(ip_remota,ip_remota_merge_10,4)==0)
{
if(torn_inactivitat_1==1)
{
segons_inactivitat_1=0;
torn_inactivitat_1=0;
}
for(int i=0;i<512;i++)
{
if(buffer_udp[i+18]>=buffer_udp_merge_11[i])
{
buffer_dmx1[i]=buffer_udp[i+18];
}
memcpy(buffer_udp_merge_10,buffer_udp+18,512);
}
}
if(memcmp(ip_remota,ip_remota_merge_11,4)==0)
{
if(torn_inactivitat_1==0)
{
segons_inactivitat_1=0;
torn_inactivitat_1=1;
}
for(int i=0;i<512;i++)
{
if(buffer_udp[i+18]>=buffer_udp_merge_10[i])
{
buffer_dmx1[i]=buffer_udp[i+18];
}
memcpy(buffer_udp_merge_11,buffer_udp+18,512);
}
}

}else
{
/*Mode merge LTP no implementat*/
}
/*si es compten 10 segons seguits, es desactiva el mode merge i l'interrupció*/
if(segons_inactivitat_1>=10)
{
estat_merge_1=0;
identificacio_ip_10=0;
identificacio_ip_11=0;
TIMSK3&=~(1<<OCIE3A);
torn_inactivitat_1=0;
segons_inactivitat_1=0;
}
}else
{
memcpy(buffer_dmx1, &buffer_udp[18],512);
}
dmx_llest_1=1;
}

if(estat_dmx_1==0)
{
dmx_llest_1=0;
envia_dmx_1();
}
}
}

/*En cas d'esser un missatge de tipus ArtPoll s'enviarà un missatge de resposta
ArtPollReply*/

```

```

if ((tipus_paquet==codi_poll) && (mida_paquet_udp==mida_poll))
{
talktome=(buffer_udp[12]&(0x02));
enviar_pollreply();
}
/*Si es un missatge de tipus ArtAddress s'efectuarian els canvis en el node en
funció del contingut del missatge*/
if ((tipus_paquet==codi_address) && (mida_paquet_udp==mida_address))
{
paquet_artaddress_p=&paquet_artaddress;
memcpy(paquet_artaddress_p,buffer_udp,mida_address);
if(paquet_artaddress_p->netswitch & 0x80)
{
net_web=(paquet_artaddress_p->netswitch & 0x7F);
}
if(paquet_artaddress_p->subswitch & 0x80)
{
subnet_web=(paquet_artaddress_p->subswitch & 0x7F);
}
if(paquet_artaddress_p->swout[0] & 0x80)
{
univers0_web=(paquet_artaddress_p->swout[0] & 0x7F);
}
if(paquet_artaddress_p->swout[1] & 0x80)
{
univers1_web=(paquet_artaddress_p->swout[1] & 0x7F);
}
/*En cas de rebre la comanda de parar el mode merge, es desa l'adreça d'on prové el
missatge per acceptar ArtDMX
només d'aquesta, i també es desactiven les interrupcions*/
if(paquet_artaddress_p->command & 0x01)
{
TIMSK1&=~(1<<OCIE1A);
TIMSK3&=~(1<<OCIE3A);
cancela_merge=1;
memcpy(ip_remota_cancelacio,ip_remota,4);
estat_merge_0=0;
estat_merge_1=0;
}

memcpy(nom_web,paquet_artaddress_p->shortname,18);
/*un cop desats els valors del missatge, s'actualitza l'EEPROM i s'envia un
missatge de tipus ArtPollReply de resposta*/
actualitzar_eeprom(0);
enviar_pollreply();
}

/*Si es un missatge de tipus ArtIpProg s'efectuarian els canvis en el node en funció
del contingut del missatge*/
if ((tipus_paquet==codi_ipprog) && (mida_paquet_udp==mida_ipprog))
{
paquet_artipprog_p=&paquet_artipprog;
memcpy(paquet_artipprog_p,buffer_udp,mida_ipprog);
if((paquet_artipprog_p->command & 0x82)==0x82)
{
subnet_mask_node[3]=paquet_artipprog_p->progs0;
subnet_mask_node[2]=paquet_artipprog_p->progs1;
subnet_mask_node[1]=paquet_artipprog_p->progs2;
subnet_mask_node[0]=paquet_artipprog_p->progs3;
}
if((paquet_artipprog_p->command & 0x84)==0x84)
{
ip_node[3]=paquet_artipprog_p->progip0;
ip_node[2]=paquet_artipprog_p->progip1;
ip_node[1]=paquet_artipprog_p->progip2;
ip_node[0]=paquet_artipprog_p->progip3;
}
}

```

```

/*Si l'objectiu del missatge era canviar l'IP o la màscara de subxarxa,
s'actualitzarà l'EEPROM amb aquests valors*/
if((paquet_artiprog_p->command & 0x82)==0x82 || (paquet_artiprog_p->command &
0x84)==0x84)
{
actualitzar_eeprom(1);
}
/*Si l'objectiu del missatge era tornar als valors de fàbrica, es realitza aquesta
acció*/
if((paquet_artiprog_p->command & 0x88)==0x88)
{
reinici_valors();
}
/*En cas d'haver rebut una comanda a realitzar, s'envia un missatge de tipus
ArtIpProgReply de resposta*/
if((paquet_artiprog_p->command & 0x82)==0x82 || (paquet_artiprog_p->command &
0x84)==0x84 || (paquet_artiprog_p->command & 0x84)==0x88)
{
omplir_ipprogreply();
memcpy(buffer_udp, &paquet_ipprogreply,34);
inici_paquet_udp(ip_remota, port_artnet);
for(int i=0;i<34;i++)
{
escriure_udp(buffer_udp[i]);
}
terminar_paquet_udp();
}
}

/*Es llegeix l'estat del socket TCP i en funció d'aquest es procedeix*/
estat_tcp=W5100.readSnSR(1);
/*Si el socket està tancat, s'obre*/
switch(estat_tcp) {
case SnSR::CLOSED:
iniciar_tcp();
break;
/*Si s'ha establert una comunicació, es comprova si hi ha missatges al buffer de
recepció per llegir*/
case SnSR::ESTABLISHED:
mida_paquet_tcp=W5100.getRXReceivedSize(1);
if (mida_paquet_tcp > 0) {
/*Es desa el buffer de recepció*/
rebre_tcp(buffer_tcp,mida_paquet_tcp);

/*Es comprova si al missatge hi ha un mètode de HTTP, en cas afirmatiu es pasa a
generar la plana web*/
get_present=cerca_str((char *)buffer_tcp,"GET /");
post_present=cerca_str((char *)buffer_tcp,"POST /");
if (get_present >= 0 || post_present >= 0) {
/*Si hi ha el mètode POST s'interpreten les dades que inclou i es desen a les
variables corresponents*/
if(post_present>=0)
{
index= strstr ((char *)buffer_tcp,"IP=");
sscanf
(index+3,"%u.%u.%u.%u",&ip_node_web[0],&ip_node_web[1],&ip_node_web[2],&ip_node_web
[3]);
index= strstr ((char *)buffer_tcp,"SM=");
sscanf
(index+3,"%u.%u.%u.%u",&subnet_mask_node_web[0],&subnet_mask_node_web[1],&subnet_ma
sk_node_web[2],&subnet_mask_node_web[3]);
index= strstr ((char *)buffer_tcp,"GW=");
sscanf
(index+3,"%u.%u.%u.%u",&gateway_node_web[0],&gateway_node_web[1],&gateway_node_web[
2],&gateway_node_web[3]);
index= strstr ((char *)buffer_tcp,"MAC=");

```



```

sscanf
(index+4, "%X%3A%X%3A%X%3A%X%3A%X%3A%X", &mac_node_web[0], &mac_node_web[1], &mac_
node_web[2], &mac_node_web[3], &mac_node_web[4], &mac_node_web[5]);
index= strstr ((char *)buffer_tcp, "NET=");
sscanf (index+4, "%u", &net_web);
if (net_web<0) net_web=0;
if (net_web>127) net_web=127;
index= strstr ((char *)buffer_tcp, "SNET=");
sscanf (index+5, "%u", &subnet_web);
if (subnet_web<0) subnet_web=0;
if (subnet_web>15) subnet_web=15;
index= strstr ((char *)buffer_tcp, "UNI0=");
sscanf (index+5, "%u", &univers0_web);
if (univers0_web<0) univers0_web=0;
if (univers0_web>15) univers0_web=15;
index= strstr ((char *)buffer_tcp, "UNI1=");
sscanf (index+5, "%u", &univers1_web);
if (univers1_web<0) univers1_web=0;
if (univers1_web>15) univers1_web=15;
index= strstr ((char *)buffer_tcp, "NOM=");
sscanf (index+4, "%18s", nom_web);

for(int i=0; i<18; i++)
{
if (nom_web[i]==43)
nom_web[i]=32;
}
/*SÍ s'ha premut la checkbox de efectuar canvis, s'actualitza l'EEPROM*/
if (cerca_str((char *)buffer_tcp, "canvis=si")>=0)
{
actualitzar_eeprom(0);
}
}
/*Es genera la plana web i es va desant en el buffer de transmissió*/
strcpy_P((char *)buffer_tcp, PSTR("HTTP/1.0 200 OK\r\nContent-Type:
text/html\r\n\r\n<!DOCTYPE HTML>\r\n"));

strcat_P((char *)buffer_tcp, PSTR("<html><form method=\"\"post\">\r\n <h1>Configuració
del node Artnet-DMX</h1> <p><b>Realitzar canvis?</b>"));

strcat_P((char *)buffer_tcp, PSTR("<input type=\"checkbox\" name=\"canvis\"
value=\"si\"></p><hr>\r\nIP actual: \r\n"));
formatejar_addr_8(ip_node, 4);
strcat_P((char *)buffer_tcp, PSTR("\r\n<br><b>IP</b> nova: <b>"));
formatejar_addr_16(ip_node_web, 4);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"post\">\r\n<p>IP:<input
type=\"text\" name=\"IP\" maxlength=\"15\"></p><hr>"));

strcat_P((char *)buffer_tcp, PSTR("\r\nSubnet Mask actual: "));
formatejar_addr_8(subnet_mask_node, 4);
strcat_P((char *)buffer_tcp, PSTR("<br>\r\n<b>Subnet Mask</b> nova: <b>"));
formatejar_addr_16(subnet_mask_node_web, 4);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"post\">\r\n<p>Subnet
Mask:<input type=\"text\" name=\"SM\" maxlength=\"15\"></p><hr>"));

strcat_P((char *)buffer_tcp, PSTR("\r\nDefault Gateway actual: "));
formatejar_addr_8(gateway_node, 4);
strcat_P((char *)buffer_tcp, PSTR("<br>\r\n<b>Default Gateway</b> nova: <b>"));
formatejar_addr_16(gateway_node_web, 4);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"post\">\r\n<p>Default
Gateway:<input type=\"text\" name=\"GW\" maxlength=\"15\"></p><hr>"));

strcat_P((char *)buffer_tcp, PSTR("MAC actual: "));
formatejar_addr_8(mac_node, 6);
strcat_P((char *)buffer_tcp, PSTR("<br>\r\n<b>MAC</b> nova: <b>"));
formatejar_addr_16(mac_node_web, 6);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"post\">\r\n<p>MAC:<input
type=\"text\" name=\"MAC\" maxlength=\"17\"></p><hr>"));

```

```

strcat_P((char *)buffer_tcp, PSTR("\r\nNet actual: "));
formatejar_addr_8(&net, 1);
strcat_P((char *)buffer_tcp, PSTR("<br>\r\n<b>Net</b> nova: <b>"));
formatejar_addr_16(&net_web, 1);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"\"post\"\">\r\n<p>Net (0-127):<input type=\"\"text\"\" name=\"\"NET\"\" maxlength=\"\"2\"\"></p><hr>"));

strcat_P((char *)buffer_tcp, PSTR("\r\nSubNet actual: "));
formatejar_addr_8(&subnet, 1);
strcat_P((char *)buffer_tcp, PSTR("<br>\r\n<b>SubNet</b> nova: <b>"));
formatejar_addr_16(&subnet_web, 1);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"\"post\"\">\r\n<p>Subnet (0-15):<input type=\"\"text\"\" name=\"\"SNET\"\" maxlength=\"\"2\"\"></p><hr>"));

strcat_P((char *)buffer_tcp, PSTR("\r\nUnivers port 0 actual: \r\n"));
formatejar_addr_8(&univers0, 1);
strcat_P((char *)buffer_tcp, PSTR("<br>\r\n<b>Univers port 0</b> nou: <b>"));
formatejar_addr_16(&univers0_web, 1);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"\"post\"\">\r\n<p>Univers p0 (0-15):<input type=\"\"text\"\" name=\"\"UNI0\"\" maxlength=\"\"2\"\"></p><hr>"));

strcat_P((char *)buffer_tcp, PSTR("\r\nUnivers port 1 actual: \r\n"));
formatejar_addr_8(&univers1, 1);
strcat_P((char *)buffer_tcp, PSTR("<br>\r\n<b>Univers port 1</b> nou: <b>"));
formatejar_addr_16(&univers1_web, 1);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"\"post\"\">\r\n<p>Univers p1 (0-15):<input type=\"\"text\"\" name=\"\"UNI1\"\" maxlength=\"\"2\"\"></p><hr>"));

strcat_P((char *)buffer_tcp, PSTR("\r\nNom actual: "));
strcat((char *)buffer_tcp, nom);
strcat_P((char *)buffer_tcp, PSTR("<br>\r\n<b>Nom</b> nou: <b>"));
strcat((char *)buffer_tcp, nom_web);
strcat_P((char *)buffer_tcp, PSTR("</b><form method=\"\"post\"\">\r\n<p>Nom:<input type=\"\"text\"\" name=\"\"NOM\"\" maxlength=\"\"18\"\"></p><hr>"));

strcat_P((char *)buffer_tcp, PSTR("<input type=\"\"submit\"\" value=\"\"Enviar\"\"></br>"));
strcat_P((char *)buffer_tcp, PSTR("\r\n</html>\r\n"));

/*Un cop omplert el buffer amb la plana web, s'envia*/
enviar_tcp(buffer_tcp, strlen((char *)buffer_tcp));

}

/*Es finalitza la connexió TCP*/
disconnect(1);
}
break;

/*si es troba en qualsevol altre estat, es tanca el socket*/
case SnSR::FIN_WAIT:
case SnSR::CLOSING:
case SnSR::TIME_WAIT:
case SnSR::CLOSE_WAIT:
case SnSR::LAST_ACK:
close(1);
break;
}
}
}

/*Funció que comprova la validesa dels missatges Art-Net i retorna el tipus del missatge*/
int comprovar_tipus()
{
/*Es desa el contingut del buffer de recepció*/
llegir_udp(buffer_udp, 530);

```

```

/*Comprova si el missatge comença amb el missatge Art-Net/0*/
if (!memcmp(buffer_udp, artnet_inici, 8))
{
/*Si ho fa, comprova que la versió del protocol es l'adequada (0x000E)*/
if (((buffer_udp[10] << 8) & 0xff00) | (buffer_udp[11] & 0x00FF))==0x000E)
{
/*retorna el tipus del missatge en cas afirmatiu*/
return (((buffer_udp[9] << 8) & 0xff00) | (buffer_udp[8] & 0x00FF));
}
}
return 0;
}

/*Funció que genera un missatge de tipus ArtPollReply fent servir les variables de
configuració del node i l'estructura del missatge*/
void omplir_pollreply ()
{
paquet_pollreply_p=&paquet_pollreply;
memcpy (paquet_pollreply_p->id, artnet_inici, sizeof(paquet_pollreply_p->id));
paquet_pollreply_p->opcode=codi_pollreply;
memcpy (paquet_pollreply_p->ip, ip_node, sizeof(paquet_pollreply_p->ip));
paquet_pollreply_p->port=port_artnet;
paquet_pollreply_p->verh=0;
paquet_pollreply_p->ver=0;
paquet_pollreply_p->subh=net;
paquet_pollreply_p->sub=subnet;
paquet_pollreply_p->oemh=0x00;
paquet_pollreply_p->oem=0xFF;
paquet_pollreply_p->ubea=0;
paquet_pollreply_p->status1=0xE0;
memset (paquet_pollreply_p->etsaman, 0xFF, sizeof(paquet_pollreply_p->etsaman));
memcpy (paquet_pollreply_p->shortname, nom, sizeof(paquet_pollreply_p->shortname));
memset (paquet_pollreply_p->longname, 0, sizeof(paquet_pollreply_p->longname));
memset (paquet_pollreply_p->nodereport, 0, sizeof(paquet_pollreply_p->nodereport));
paquet_pollreply_p->numbportsh=0;
paquet_pollreply_p->numbports=2;
memset (paquet_pollreply_p->porttypes, 0x80, 2);
memset (paquet_pollreply_p->porttypes+2, 0x00, 2);
memset (paquet_pollreply_p->goodinput, 0x08, sizeof(paquet_pollreply_p->goodinput));
paquet_pollreply_p->goodoutput[0]=0x80+estat_merge_0;
paquet_pollreply_p->goodoutput[1]=0x80+estat_merge_1;
memset (paquet_pollreply_p->goodoutput+2, 0x00, 2);
paquet_pollreply_p->swout[0]=univers0;
paquet_pollreply_p->swout[1]=univers1;
paquet_pollreply_p->swvideo=0;
paquet_pollreply_p->swmacro=0;
paquet_pollreply_p->swremote=0;
paquet_pollreply_p->sp1=0;
paquet_pollreply_p->sp2=0;
paquet_pollreply_p->sp3=0;
paquet_pollreply_p->style=0;
memcpy (paquet_pollreply_p->mac, mac_node, sizeof(paquet_pollreply_p->mac));
memset (paquet_pollreply_p->bindip, 0x00, sizeof(paquet_pollreply_p->bindip));
paquet_pollreply_p->bindindex=0;
paquet_pollreply_p->status2=0x09;
memset (paquet_pollreply_p->filler, 0x00, sizeof(paquet_pollreply_p->filler));
}

/*Funció que genera un missatge de tipus ArtIpProgReply fent servir les variables
de configuració del node i l'estructura del missatge*/
void omplir_ipprogreply()
{
paquet_ipprogreply_p=&paquet_ipprogreply;
memcpy (paquet_ipprogreply_p->id, artnet_inici, sizeof(paquet_ipprogreply_p->id));
paquet_ipprogreply_p->opcode=codi_ipprogreply;
paquet_ipprogreply_p->protverhi=0;

```

```

paquet_ipprogreply_p->protverlo=0x0E;
paquet_ipprogreply_p->progip3=ip_node[0];
paquet_ipprogreply_p->progip2=ip_node[1];
paquet_ipprogreply_p->progip1=ip_node[2];
paquet_ipprogreply_p->progip0=ip_node[3];
paquet_ipprogreply_p->progs3=subnet_mask_node[0];
paquet_ipprogreply_p->progs2=subnet_mask_node[1];
paquet_ipprogreply_p->progs1=subnet_mask_node[2];
paquet_ipprogreply_p->progs0=subnet_mask_node[3];
paquet_ipprogreply_p->progporthi=net;
paquet_ipprogreply_p->progportlo=(subnet<<4)+univers0;
}

/*Funció que inicia la transmissió d'una trama DMX al port USART0*/
void envia_dmx_0() {
estat_dmx_0 = 1;
/*La transmissió s'inicia a una freqüència més baixa (80.000kHz) per generar el
senyal de break*/
UBRR0H = ((F_CPU/80000/16) - 1) >> 8;
UBRR0L = ((F_CPU/80000/16) - 1);
/*S'habilita la interrupció de transmissió complerta*/
UCSR0B |= 1<<TXCIE0;
/*S'envia un byte buit per genera el senyal de break*/
UDR0 = 0;
}

/*Funció que inicia la transmissió d'una trama DMX al port USART1*/
void envia_dmx_1() {
estat_dmx_1 = 1;
UBRR1H = ((F_CPU/80000/16) - 1) >> 8;
UBRR1L = ((F_CPU/80000/16) - 1);
UCSR1B |= 1<<TXCIE1;
UDR1 = 0;
}

/*Interrupció que es genera cada cop que la transmissió d'un byte es compleix*/
ISR(USART0_TX_vect) {
/*Es comprova si està en estat d'enviar totes les dades DMX primer ja que es
l'estat més freqüent*/
if (estat_dmx_0 == 3) {
/*S'envia la resta de dades de la trama DMX*/
UDR0 = buffer_dmx0[ptr_dmx_0];
ptr_dmx_0++;
if (ptr_dmx_0 >= 512) {
estat_dmx_0 = 4;
}
}
else if (estat_dmx_0 == 1) {
/*Es configura el baudrate a 250.000 kHz*/
UBRR0H = ((F_CPU/250000/16) - 1) >> 8;
UBRR0L = ((F_CPU/250000/16) - 1);
/*S'envia el byte d'start*/
UDR0 = 0;
estat_dmx_0 = 2;
}
else if (estat_dmx_0 == 2) {
/*S'envia el primer byte de dades DMX i s'inicialitza el punter*/
UDR0 = buffer_dmx0[0];
ptr_dmx_0 = 1;
estat_dmx_0 = 3;
}
else {
/*Un cop enviats tots el bytes, es desabilita l'interrupció, es posa a 0 el flag
d'aquesta i la variable d'estat de transmissió de DMX*/
UCSR0B &= ~(1<<TXCIE0);
UCSR0A |= 1<<TXC0;
}
}

```

```

estat_dmx_0 = 0;
}
}

ISR(USART1_TX_vect) {
if (estat_dmx_1 == 3) {
UDR1 = buffer_dmx1[ptr_dmx_1];
ptr_dmx_1++;
if (ptr_dmx_1 >= 512) {
estat_dmx_1 = 4;
}
}
else if (estat_dmx_1 == 1) {
UBRR1H = ((F_CPU/250000/16) - 1) >> 8;
UBRR1L = ((F_CPU/250000/16) - 1);
UDR1 = 0;
estat_dmx_1 = 2;
}
else if (estat_dmx_1 == 2) {
UDR1 = buffer_dmx1[0];
ptr_dmx_1 = 1;
estat_dmx_1 = 3;
}
else {
UCSR1B &= ~(1<<TXCIE1);
UCSR1A |= 1<<TXC1;
estat_dmx_1 = 0;
}
}

/*Funció d'inicialització dels registres dels ports USART. Es configuren amb 8 bits
de dades i 2 bits de stop*/
void iniciar_usart()
{
UBRR0H = ((F_CPU/250000/16) - 1) >> 8;
UBRR0L = ((F_CPU/250000/16) - 1);
UCSR0A = 1<<UDRE0;
UCSR0C = 1<<USBS0 | 1<<UCSZ01 | 1<<UCSZ00; // 2 stop bits,8 data bitss
UCSR0B = 1<<TXEN0; // turn it on

UBRR1H = ((F_CPU/250000/16) - 1) >> 8;
UBRR1L = ((F_CPU/250000/16) - 1);
UCSR1A = 1<<UDRE1;
UCSR1C = 1<<USBS1 | 1<<UCSZ11 | 1<<UCSZ10; // 2 stop bits,8 data bitss
UCSR1B = 1<<TXEN1; // turn it on
}

/*Inicialització del xip w5100*/
void iniciar_w5100(uint8_t *mac, uint8_t *ip_address, uint8_t *gateway, uint8_t
*mask)
{
W5100.init();
W5100.setMACAddress(mac);
W5100.setIPAddress(ip_address);
W5100.setGatewayIp(gateway);
W5100.setSubnetMask(mask);
}

/*Inicialització del socket UDP*/
void iniciar_udp(uint16_t port)
{
socket(0, SnMR::UDP, port, 0);
}

/*Funció que comprova la recepció de paquets UDP i en cas afirmatiu desa la IP font
i retorna la mida del paquet*/
int extreure_udp()
{

```

```
/*Es descarten bytes pertanyents al paquet anteriorment rebut*/
flush_udp();

if (W5100.getRXReceivedSize(0) > 0)
{
    uint8_t tmpBuf[8];
    int ret =0;
    /*Llegeix els 8 primer bytes del paquet UDP, capçalera que posa el xip amb la
    direcció IP font, el port destí i la mida del paquet*/
    ret = recv(0,tmpBuf,8);
    if (ret > 0)
    {
        memcpy(ip_remota,tmpBuf,4);
        /*Desa la mida del paquet i la retorna*/
        restant = tmpBuf[6];
        restant = (restant << 8) + tmpBuf[7];
        ret = restant;
        /*la variable d'identificació serà 0 al iniciar el programa i quan passin 10 segons
        al mode merge
        desde el moment en que es va rebre l'últim paquet d'una de les dues IP*/
    }
    return ret;
}
return 0;
}

/*En cas de que quedin bytes per llegir d'un paquet anterior, aquesta funció els
llegeix per tal d'actualitzar el punter de recepció del xip i fer
que apunti a la direcció del nou paquet rebut*/
void flush_udp()
{
    while (restant)
    {
        llegir_udp();
    }
}

/*Funció que llegeix el buffer de recepció UDP byte per byte*/
int llegir_udp()
{
    uint8_t byte;

    if ((restant > 0) && (recv(0, &byte, 1) > 0))
    {
        // We read things without any problems
        restant--;
        return byte;
    }

    // If we get here, there's no data available
    return -1;
}

/*Funció que llegeix un número de bytes especificat del buffer de recepció UDP i
els desa en el array especificat*/
int llegir_udp(unsigned char* buffer, size_t len)
{
    if (restant > 0)
    {
        int got;
        if (restant <= len)
        {
            got = recv(0, buffer, restant);
        }
        else
        {
            got = recv(0, buffer, len);
        }
    }
}
```

```

}

if (got > 0)
{
    restant -= got;
    return got;
}
return -1;
}

/*Funció per preparar la transmissió d'un paquet UDP. Escriu en els registres
indicats la IP i port destí*/
int inici_paquet_udp(uint8_t *ip, uint16_t port)
{
    offset = 0;
    return startUDP(0, ip, port);
}

/*Funció que escriu la comanda d'enviar el paquet UDP al xip d'ethernet*/
int terminar_paquet_udp()
{
    return sendUDP(0);
}

/*Funció que escriu un byte al buffer de transmissió UDP*/
size_t escriure_udp(uint8_t byte)
{
    return escriure_udp(&byte, 1);
}

/*Funció que passa els continguts d'un array al buffer de transmissió UDP*/
size_t escriure_udp(const uint8_t *buffer, size_t size)
{
    uint16_t bytes_written = bufferData(0, offset, buffer, size);
    offset += bytes_written;
    return bytes_written;
}

/*Funció que obre el socket TCP*/
void iniciar_tcp()
{
    socket(1, SnMR::TCP, 80, 0);
    listen(1);
}

/*Funció per transmetre bytes via el socket TCP*/
void enviar_tcp(const uint8_t *buffer_tcp, uint16_t buflen)
{
    uint16_t ptr, realaddr, txsize, timeout;

    txsize= W5100.readSnTX_FSR(1);
    timeout=0;
    while (txsize < buflen) {
        _delay_ms(1);

        txsize= W5100.readSnTX_FSR(1);

        /*si al cap d'un segon el buffer de transmissió no te prou espai per el paquet, es
        tanca la connexió*/
        if (timeout++ > 1000) {
            disconnect(1);
        }
    }
    /*Es calcula la posició del punter del buffer de transmissió*/
    ptr = W5100.readSnTX_WR(1);
    while(buflen) {
        buflen--;
    }
}

```

```

realaddr = 0x4800 + (ptr & 0x07FF);
/*S'escriu byte a byte el contingut del paquet al buffer de transmissió*/
W5100.write(realaddr,*buffer_tcp);
ptr++;
buffer_tcp++;
}
/*Es dóna la comanda de transmissió al xip d'ethernet*/
W5100.writeSnTX_WR(1, ptr);
W5100.execCmdSn(1, Sock_SEND);

}

/*Funció per rebre bytes del socket TCP*/
void rebre_tcp(uint8_t *buffer_tcp,uint16_t buflen)
{
uint16_t ptr,realaddr;
/*Si el paquet té una mida superior a 1000 bytes, es trunca*/
if (buflen > 1000)
buflen=1000 - 2;
ptr = W5100.readSnRX_RD(1);
/*Es calcula la posició del punter del buffer de recepció*/
while(buflen) {
buflen--;
realaddr= 0x6800 + (ptr & 0x07FF);
*buffer_tcp = W5100.read(realaddr);
ptr++;
buffer_tcp++;
}
/*Es finalitza el paquet rebut amb un caràcter nul*/
*buffer_tcp='\0';
W5100.writeSnRX_RD(1, ptr);
/*Es dóna la comanda de recepció complerta al xip d'ethernet*/
W5100.execCmdSn(1, Sock_RECV);
}

/*Funció que cerca un string en un altre. Si el troba retorna 0 o un número
positiu*/
int cerca_str(char *s,char *t)
{
uint16_t i,n;

n=strlen(t);
for(i=0;*(s+i); i++) {
if (strncmp(s+i,t,n) == 0)
return i;
}
return -1;
}

/*Funcio que actualitza les dades de les variables de la web*/
void iniciar_dades_web()
{
eeprom_read_block((void*) ip_node, ip_node_eeprom, 4);
eeprom_read_block((void*) subnet_mask_node, subnet_mask_eeprom, 4);
eeprom_read_block((void*) gateway_node, gateway_node_eeprom, 4);
eeprom_read_block((void*) mac_node, mac_node_eeprom, 6);
net=eeprom_read_byte (&net_eeprom);
subnet=eeprom_read_byte (&subnet_eeprom);
univers0=eeprom_read_byte (&univers0_eeprom);
univers1=eeprom_read_byte (&univers1_eeprom);
eeprom_read_block((void*) nom, nom_eeprom, 18);

for(int i=0;i<6;i++)
{
if(i<4)
{
ip_node_web[i]=ip_node[i];

```



```

subnet_mask_node_web[i]=subnet_mask_node[i];
gateway_node_web[i]=gateway_node[i];
}
mac_node_web[i]=mac_node[i];
}
net_web=net;
subnet_web=subnet;
univers0_web=univers0;
univers1_web=univers1;
memcpy(nom_web,nom,18);
}

/*Funció que desa els valors de fàbrica a les variables del nodes i després
actualitza la EEPROM amb aquests, provocant un reinici del xip ethernet*/
void reinici_valors()
{
memcpy(ip_node,ip_node_fab,4);
memcpy(subnet_mask_node,subnet_mask_node_fab,4);
memcpy(gateway_node,gateway_node_fab,4);
memcpy(mac_node,mac_node_fab,6);
net_web=net_fab;
subnet_web=subnet_fab;
univers0_web=univers0_fab;
univers1_web=univers1_fab;
memcpy(nom_web,nom_fab,18);
actualitzar_eeprom(1);
}

/*S'actualitzen les dades de configuració del node a la EEPROM i es reseteja el xip
Ethernet per tal d'aplicar els canvis*/
/*En funció del mode, les dades a desar provenen de la variable de la web o no*/
void actualitzar_eeprom(uint8_t mode)
{
if (mode==0)
{
for(int i=0;i<6;i++)
{
if(i<4)
{
ip_node[i]=ip_node_web[i];
subnet_mask_node[i]=subnet_mask_node_web[i];
gateway_node[i]=gateway_node_web[i];
}
mac_node[i]=mac_node_web[i];
}
}
eeprom_update_block((const void*)ip_node,(void*)ip_node_eeprom,4);
eeprom_update_block((const void*)subnet_mask_node,(void*)subnet_mask_eeprom,4);
eeprom_update_block((const void*)gateway_node,(void*)gateway_node_eeprom,4);
eeprom_update_block((const void*)mac_node,(void*)mac_node_eeprom,6);
eeprom_update_byte(&net_eeprom,(uint8_t)net_web);
eeprom_update_byte(&subnet_eeprom,(uint8_t)subnet_web);
eeprom_update_byte(&univers0_eeprom,(uint8_t)univers0_web);
eeprom_update_byte(&univers1_eeprom,(uint8_t)univers1_web);
eeprom_update_block((const void*)nom_web,(void*)nom_eeprom,18);
iniciar_dades_web();
iniciar_sockets();
if(talktome==2)
{
enviar_pollreply();
}
}

/*Funció per a convertir les addreces de 8 bits a text*/
void formatjar_addr_8(unsigned char* addr, uint8_t num)

```

```

{
for(int i=0;i<num;i++)
{
if(num!=6)
{
sprintf(conversio_str,"%d",addr[i]);
}else
{
sprintf(conversio_str,"%X",addr[i]);
}
strcat((char *)buffer_tcp,conversio_str);
if(num==4)
{
if(i<3) strcat_P((char *)buffer_tcp,PSTR("."));
}
if(num==6)
{
if(i<5) strcat_P((char *)buffer_tcp,PSTR(":"));
}
}

}

/*Funció per a convertir les addreces de 16 bits a text*/
void formatejar_addr_16(uint16_t* addr, uint8_t num)
{
for(int i=0;i<num;i++)
{
if(num!=6)
{
sprintf(conversio_str,"%d",addr[i]);
}else
{
sprintf(conversio_str,"%X",addr[i]);
}
strcat((char *)buffer_tcp,conversio_str);
if(num==4)
{
if(i<3) strcat_P((char *)buffer_tcp,PSTR("."));
}
if(num==6)
{
if(i<5) strcat_P((char *)buffer_tcp,PSTR(":"));
}
}

}

/*Inicialització de tots dos sockets a utilitzar del xip ethernet*/
void iniciar_sockets()
{
iniciar_w5100(mac_node,ip_node,gateway_node,subnet_mask_node);
iniciar_udp(port_artnet);
iniciar_tcp();
}

/*Funció que envia un missatge de tipus ArtPollReply via UDP*/
void enviar_pollreply()
{
omplir_pollreply();
memcpy(buffer_udp, &paquet_pollreply,239);
for (int i=0;i<4;i++)
{
ip_remota_broad[i]=(ip_remota[i]&subnet_mask_node[i])+(~subnet_mask_node[i]);
}
inici_paquet_udp(ip_remota_broad, port_artnet);
for(int i=0;i<240;i++)

```

```
{
escriure_udp(buffer_udp[i]);
}
terminar_paquet_udp();
}

/*Rutines d'interruptió de comparació. Cada segon incrementen una variable per
determinar
si cal desactivar el mode merge*/

ISR(TIMER1_COMPA_vect)
{
segons_inactivitat_0++;
}
ISR(TIMER3_COMPA_vect)
{
segons_inactivitat_1++;
}
```