**Общее описание командной станции MD\_Prog**

В программе есть 2 модуля:

1. Интерфейс с юзером. Этот модуль обрабатывает контролы на форме и собирает информацию – какие кнопки нажал юзер, изменение адреса лока и движок скорости лока.  
   Этот модуль на основании этих действий формирует DCC команды и записывает их в буфер (массив команд)
2. Интерфейс с программатором. Там есть таймер, он включается, когда юзер нажимает кнопку «включить станцию», этот таймер делает прерывания каждые 50мс. В прерывании рутина залезает в массив команд и смотрит – есть ли готовые DCC команды, если нет, то ничего не происходит, если есть команда, то берем ее из массива команд и отправляем программатору.

Массив команд – это не просто массив. Он состоит из 3-х частей, каждая из которых является массивом с командами:

1. Массив приоритетных команд. Если в нем есть команда, то она отправляется в очередном прерывании таймера не зависимо от того есть ли какие то команды в других массивах. После передачи команда удаляется из массива.
2. Массив с командами аксессуарным декодерам (переключение стрелок). Он имеет меньший приоритет чем массив в п.1, но больший чем массив с локомотивными командами. После передачи команда удаляется из массива.
3. Массив с командами для локомотивных декодеров. Низший приоритет. Рутина из прерывания таймера возьмет из него команду только если в 2-х предыдущих массивах ничего нет. После отправки команды программатору она не удаляется из массива. Таким образом, команды для лок декодеров циклически повторяются.

**Интерфейс с программатором**

Вот кусок кода из прерывания таймера, он определяет – какая команда сейчас будет отправлена программатору:

// получить следующую команду

// а есть приоритетные команды ?

Len=HiPriorDCCCommand.GetNextCommandAndDel((byte \*)&Command); //если в HiPriorDCCCommand есть команда - она скопируется в Command и удалится из списка

if( Len == 0xFF )

{// нет приоритетных

// берем команду для аксессуарных декодеров

Len=AccDecoders.GetNextCommand((byte \*)&Command);

if( Len == 0xFF )

{// нет команд для аксессуарных декодеров

// берем обычную

Len=Decoders.GetNextCommand((byte \*)&Command);

}

}

В результате его выполнения в буфере

byte Command[MAX\_BYTES\_IN\_COMMAND];

мы получаем команду для передачи сейчас.

Эти байты соответствуют байтам DCC команды 1 в 1, за исключение того, что отсутствует контрольная сумма (ее будет считать сам программатор перед передачей команды на выход станции)

Ниже рутина, которая отправляет эти байты программатору, она добавляет в начало пакета 2 байта:

1. код команды: PR\_OPCODE\_OM\_COMMAND (значение констант в конце документа)
2. кол-во байт в этом пакете включая эти 2 байта заголовка.

// byte \*Command, там байты DCC команды БЕЗ PR\_OPCODE\_OM\_COMMAND и без 2-го байта длинны посылки

// byte Len - количество байт в Command

int PROG::OP\_Mode\_Command(byte \*Command, byte Len)

{

char Comm[10];

byte AckProgrammer=0xff;

DWORD cntr=0xff;

unsigned int tmp,tmp1;

LPVOID lpMsgBuf;

// проверяем готовность программера

if( 0!=check\_programmer\_ready() )

return -1;

// очистить внутренние буфера и отменить все операции ввода/вывода

// очистка добавлена 24.08.2010

// PurgeComm(ComPort,PURGE\_TXCLEAR | PURGE\_RXCLEAR | PURGE\_TXABORT | PURGE\_RXABORT );

// до отправки команды поставить на чтение

ResetEvent(MOvr1->hEvent);

ReadFile(ComPort,&AckProgrammer,1,&cntr,MOvr1);

// отправить команду

Comm[0]=PR\_OPCODE\_OM\_COMMAND;

Comm[1]=Len+2; // количество байт в посылке - это для программера

memcpy(&(Comm[2]),Command,Len);

// Comm[2]=LocAddr; // 1-й байт DCC команды (0AAA.AAAA)

// Comm[3]=0x62; // 2-й байт DCC команды (01DHSSSS) D-Direction, H-HeadLight, SSSS-Speed

SendBytesToCOM(Comm,Comm[1]); // отправить байты

// ждем подтверждения приема команды от программера

tmp = WaitForSingleObject(MOvr1->hEvent,1000);

ResetEvent(MOvr1->hEvent);

// очистить внутренние буфера и отменить все операции ввода/вывода

PurgeComm(ComPort,PURGE\_TXCLEAR | PURGE\_RXCLEAR | PURGE\_TXABORT | PURGE\_RXABORT );

if( WAIT\_OBJECT\_0 != tmp )

{// нет подтверждения от программера

INFO("OP\_Mode\_Command->Нет подтверждения от программера")

tmp1 = GetLastError();

FormatMessage(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL, tmp1, MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), // Default language

(LPTSTR) &lpMsgBuf, 0, NULL);

INFO("GetLastError: "+AnsiString(tmp1)+" - "+(const char \*)lpMsgBuf)

SetLastError(0);

return -1;

}

if( AckProgrammer != PR\_ACK\_DCC\_STOP )

{// нет подтверждения приема посылки

INFO("PROG::OP\_Mode\_Command->Вместо подтверждения приема команды получен байт: 0x" + IntToHex(AckProgrammer,2) )

return -1;

}

INFO("OP\_Mode\_Command->Programmer принял команду")

return 0; // ok

}

Здесь используется простенькая рутина check\_programmer\_ready()

Она вызывается каждый раз перед отправкой любого пакета программатору, вот ее код:

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

// проверяем готовность программера

// шлем PR\_READY

// ждем в ответ PR\_ACK\_DCC\_READY

int PROG::check\_programmer\_ready(void)

{

char Comm[10];

byte AckProgrammer=0;

DWORD cntr=0xff;

unsigned int tmp,tmp1,nTry=0;

LPVOID lpMsgBuf;

// очистить внутренние буфера и отменить все операции ввода/вывода

PurgeComm(ComPort,PURGE\_TXCLEAR | PURGE\_RXCLEAR | PURGE\_TXABORT | PURGE\_RXABORT );

while( nTry<20)

{

// до отправки команды поставить на чтение

ResetEvent(MOvr1->hEvent);

ReadFile(ComPort,&AckProgrammer,1,&cntr,MOvr1); //

// отправить команду

Comm[0]=PR\_READY;

Comm[1]=2; // количество байт в посылке - это для программера

SendBytesToCOM(Comm,Comm[1]); // отправить байты

// ждем приема от программера

tmp = WaitForSingleObject(MOvr1->hEvent,200);

ResetEvent(MOvr1->hEvent);

// очистить внутренние буфера и отменить все операции ввода/вывода

PurgeComm(ComPort,PURGE\_TXCLEAR | PURGE\_RXCLEAR | PURGE\_TXABORT | PURGE\_RXABORT );

if( WAIT\_OBJECT\_0 != tmp )

{// нет подтверждения от программера

INFO("check\_programmer\_ready->Нет подтверждения от программера")

}

else

{// что-то принято

if( AckProgrammer == PR\_ACK\_DCC\_READY )

{// программер готов

INFO("check\_programmer\_ready->Programmer готов")

return 0;

}

// принято не PR\_ACK\_DCC\_READY

// а может это перегрузка ?

if( AckProgrammer == PR\_OVERLOAD )

{// да

INFO("")

INFO("--------------- OVERLOAD -----------------------")

INFO("")

OnDetectOverload();

return OVERLOAD\_RETURN\_CODE;

}

}

nTry++;

} // от while

INFO("check\_programmer\_ready->Programmer НЕ готов !!!!!!!!")

Form1->Programmer\_Edit->Text="Нет связи с программатором";

return -1; // программер не готов

}

Формат DCC команд смотрите на сайте NMRA.org

**Интерфейс с юзером**

Ниже кусок кода, который вызывается, когда юзер изменил скорость лока или нажал какую то функциональную кнопку.

Этот код опрашивает все контролы на форме, относящиеся к этому локу и формирует DCC команды этому локу.

Команды формируются в промежуточных буферах.

Здесь 4 рутины, одна – отвечает за скорость и 3 других за группы функциональных кнопок.

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

// формирует байты команды исходя из (int Speed,int Dir,int Light)

// Instruction Packet for Multi Function Decoders

// 1 или 2 байта адреса декодера

// 0AAA.AAAA (адрес декодера 1-127) #############################################################################

// или

// 11AA.AAAA AAAA.AAAA (64-9999)

// Speed and direction Instruction

// 2-й байт

// 01DHSSSS

// D - Direction =0 <-Reverse, =1 <-Forward

// H - HeadLight =1 <-On

// SSSS <-Speed

// если без параматров - функция возмет их из установленных

int DECODER::MakeSpeedInsruction(void)

{

int i=0;

// проверить адрес на допустимый максимум для Long Addr

if( DecAddr > 9999 )

return -1; // не будем

// а адресс Short или Long ?

if( DecAddr <=127 )

{// SHORT

CommandS[0] = DecAddr; // адрес

i=1; // адрес байта, следующего за адресом

}

else

{// LONG

CommandS[0] = 0xc0; // два старших бита всегда =1

CommandS[0] = CommandS[0] + (DecAddr>>8); // старший байт адреса

CommandS[1] = DecAddr & 0x00ff; // младший

i=2;

}

if( Decoders.SpeedStep14 )

{//\_\_\_\_\_\_\_\_\_ 14

INFO("SpeedStep-14")

// 01DCSSSS C - Head Light для 14

if( Speed==-1)

// это Immediate Stop

CommandS[i] = 0x40 | (byte)1;

else

if( Speed>0 )

CommandS[i] = 0x40 | (byte)Speed+1; // какая то скорость >0

else // предполагаю, что Speed=0

CommandS[i] = 0x40 | (byte)0; // плавная остановка

// добавляем Direction

if( Dir==1 )

// вперед - установить бит D в 01DHSSSS

CommandS[i] = CommandS[i] | 0x20;

else

// назад - сбросить бит D в 01DHSSSS

CommandS[i] = CommandS[i] & 0xDF;

// добавляем Light

if( Light==0 )

// выключить свет - сбросить бит H в 01DHSSSS

CommandS[i] = CommandS[i] & 0xEF;

else

// включить свет - установить бит H в 01DHSSSS

CommandS[i] = CommandS[i] | 0x10;

i++;

} // от if( Decoders.SpeedStep14 )

if( Decoders.SpeedStep28 )

{//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 28

// 01DCSSSS 2-й байт

INFO("SpeedStep-28")

// 01DCSSSS C - полушаг скорости для 28/128 (C - Head Light для 14)

//CommandS[1] = 0x62;

if( Speed==-1)

// это Immediate Stop

CommandS[i] = 0x40 | (byte)1;

else

if( Speed>0 )

{// какая то скорость >0

CommandS[i] = 0x40 | (byte)((Speed+1)>>1)+1; // делаем SSSS

CommandS[i] = CommandS[i] | (((Speed-1) & 0x01)<<4); // добавляем полушаг

}

else // предполагаю, что Speed=0

CommandS[i] = 0x40 | (byte)0; // плавная остановка

// добавляем Direction

/\*

if( Dir==0 )

// вперед - сбросить бит D в 01DHSSSS

CommandS[i] = CommandS[i] & 0xDF;

else

// назад - установить бит D в 01DHSSSS

CommandS[i] = CommandS[i] | 0x20;

\*/

if( Dir==1 )

// вперед - установить бит D в 01DHSSSS

CommandS[i] = CommandS[i] | 0x20;

else

// назад - сбросить бит D в 01DHSSSS

CommandS[i] = CommandS[i] & 0xDF;

i++;

} // от if( Decoders.SpeedStep28 )

if( Decoders.SpeedStep128 )

{//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 128

// 001CCCCC 0 DDDDDDDD CCCCC = 11111: 128 Speed Step Control

/\* DDDDDDDD - The subsequent single byte shall define speed and direction with bit 7 being direction ("1" is forward and "0"

is reverse) and the remaining bits used to indicate speed. The most significant speed bit is bit 6. A data-byte value

of U0000000 is used for stop, and a data-byte value of U0000001 is used for emergency stop. \*/

CommandS[i] = 0x3f;

i++;

if( Speed==-1)

// это Immediate Stop

CommandS[i] = (byte)1;

else

if( Speed>0 )

{// какая то скорость >0

CommandS[i] = (byte)(Speed+1); // делаем DDDDDDDD

}

else // предполагаю, что Speed=0

CommandS[i] = (byte)0; // плавная остановка

// добавляем Direction

if( Dir==0 )

// вперед - сбросить бит U в UDDDDDDD

CommandS[i] = CommandS[i] & 0x7F;

else

// назад - установить бит U в UDDDDDDD

CommandS[i] = CommandS[i] | 0x80;

i++;

}

//LenS=2; для Short Address

LenS=i; // длинна посылки (без Err байта)

return 0; //ok

}

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

/\*

// 1 или 2 байта адреса декодера

// 0AAA.AAAA (адрес декодера 1-127) #############################################################################

// или

// 11AA.AAAA AAAA.AAAA (64-9999)

100HDDDD 2-й байт

H - HeadLight =1 <-On

DDDD -> F4,F3,F2,F1

D=1 - on

D=0 - off

If Bit 1 of CV#29 has a value of one (1), then bit 4 controls function FL,

otherwise bit 4 has no meaning.

\*/

// формирует байты команды исходя из (int Fs,int Dir,int Light)

int DECODER::MakeFuncGroupOneInstruction(void)

{

int i;

// проверить адрес на допустимый максимум для Long Addr

if( DecAddr > 9999 )

return -1; // не будем

// а адресс Short или Long ?

if( DecAddr <=127 )

{// SHORT

CommandFGrOne[0] = DecAddr; // адрес

i=1; // адрес байта, следующего за адресом

}

else

{// LONG

CommandFGrOne[0] = 0xc0; // два старших бита всегда =1

CommandFGrOne[0] = CommandFGrOne[0] + (DecAddr>>8); // старший байт адреса

CommandFGrOne[1] = DecAddr & 0x00ff; // младший

i=2;

}

CommandFGrOne[i] = 0x80 | (Fs & 0x0F); // младшие 4 бита определяют состояние F4-F1

if( Decoders.SpeedStep28 || Decoders.SpeedStep128)

{// 28

// добавляем бит света

CommandFGrOne[i] = CommandFGrOne[i] | ((Light & 0x01) <<4); // 4-й бит свет

}

//LenFGrOne=2;

LenFGrOne= i+1;

return 0; //ok

}

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

/\*Up to 8 additional auxiliary functions (F5-F12) can be controlled by a Function Group Two instruction.

Bit 4 defines the use of Bits 0-3.

When bit 4 (S) is ‘1’, Bits 0-3 (DDDD) shall define the value of functions F5-F8 with

function F5 being controlled by bit 0 and function F8 being controlled by bit 3.

When bit 4 (S) is ‘0’, Bits 0-3 (DDDD) shall define the value of functions F9-F12

with function F9 being controlled by bit 0 and function F12 being controlled by bit 3.

A value of "1" shall indicate that the function is "on" while a value of "0" shall indicate that the function is "off".

// 1 или 2 байта адреса декодера

// 0AAA.AAAA (адрес декодера 1-127) #############################################################################

// или

// 11AA.AAAA AAAA.AAAA (64-9999)

// 101SDDDD

S=1 -> DDDD - F8-F5 MakeFuncGroupTwo1Instruction

S=0 -> DDDD - F12-F9 MakeFuncGroupTwo2Instruction

\*/

int DECODER::MakeFuncGroupTwo1Instruction(void)

{

int i;

// а адресс Short или Long ?

if( DecAddr <=127 )

{// SHORT

CommandFGrTwo1[0] = DecAddr; // адрес

i=1; // адрес байта, следующего за адресом

}

else

{// LONG

CommandFGrTwo1[0] = 0xc0; // два старших бита всегда =1

CommandFGrTwo1[0] = CommandFGrTwo1[0] + (DecAddr>>8); // старший байт адреса

CommandFGrTwo1[1] = DecAddr & 0x00ff; // младший

i=2;

}

CommandFGrTwo1[i] = 0xB0 | ((Fs & 0xF0)>>4); // старшие 4 бита Fs определяют состояние F8-F5

LenFGrTwo1= i+1;

return 0; //ok

}

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

// 1 или 2 байта адреса декодера

// 0AAA.AAAA (адрес декодера 1-127) #############################################################################

// или

// 11AA.AAAA AAAA.AAAA (64-9999)

// 101SDDDD

/\*

S=1 -> DDDD - F8-F5 MakeFuncGroupTwo1Instruction

S=0 -> DDDD - F12-F9 MakeFuncGroupTwo2Instruction

\*/

int DECODER::MakeFuncGroupTwo2Instruction(void)

{

int i;

// а адрес Short или Long ?

if( DecAddr <=127 )

{// SHORT

CommandFGrTwo2[0] = DecAddr; // адрес

i=1; // адрес байта, следующего за адресом

}

else

{// LONG

CommandFGrTwo2[0] = 0xc0; // два старших бита всегда =1

CommandFGrTwo2[0] = CommandFGrTwo2[0] + (DecAddr>>8); // старший байт адреса

CommandFGrTwo2[1] = DecAddr & 0x00ff; // младший

i=2;

}

CommandFGrTwo2[i] = 0xA0 | ((Fs & 0x0F00)>>8); // биты 11-8 Fs определяют состояние F12-F9

LenFGrTwo2= i+1;

return 0; //ok

}

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

// 1 или 2 байта адреса декодера

// 0AAA.AAAA (адрес декодера 1-127) #############################################################################

// или

// 11AA.AAAA AAAA.AAAA (64-9999)

// 110C.CCCC DDDD.DDDD, где CCCCC=11110

// 3-й байт - это значения для F13-F20 (F13 в бите 0)

int DECODER::MakeFuncGroupThree1Instruction(void)

{

int i;

// а адрес Short или Long ?

if( DecAddr <=127 )

{// SHORT

CommandFGrThree1[0] = DecAddr; // адрес

i=1; // адрес байта, следующего за адресом

}

else

{// LONG

CommandFGrThree1[0] = 0xc0; // два старших бита всегда =1

CommandFGrThree1[0] = CommandFGrThree1[0] + (DecAddr>>8); // старший байт адреса

CommandFGrThree1[1] = DecAddr & 0x00ff; // младший

i=2;

}

CommandFGrThree1[i] = 0xDE;

i++;

CommandFGrThree1[i] = ((Fs & 0xFF000)>>12); // биты 12-19 Fs определяют состояние F13-F20

LenFGrThree1= i+1;

return 0; //ok

}

**COM port**

Ниже код рутины – открытия COM порта. В режиме командной станции порт должен быть открыт как 19200 8бит без четности, никакое управление потоком не используется. В программатор заведены только 2 линии RS232: Tx и Rx

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

void PROG::doOpenPort(bool ShowErrMSG)

{

// номер порта Config.ComPortN может быть=0 - тогда ничего не делаем

if( Config.ComPortN <= 0 )

{

INFO("PROG::doOpenPort->1: Config.ComPortN <= 0 -> return");

return;

}

char ComN[10];

AnsiString AS;

//To specify a COM port number greater than 9, use the following syntax: "\\.\COM10". This syntax works for all port numbers and hardware that allows COM port numbers to be specified.

AS = "\\\\.\\COM" + AnsiString( Config.ComPortN );

memcpy(ComN,AS.data(),AS.Length()+1);

ComPort=CreateFile(ComN,GENERIC\_READ|GENERIC\_WRITE,0,NULL,OPEN\_EXISTING,FILE\_FLAG\_OVERLAPPED,NULL);

if(ComPort==INVALID\_HANDLE\_VALUE)

{

AS ="Не возможно открыть последовательный порт COM"+ AnsiString(Config.ComPortN)+"\nвозможно он занят другой программой";

if( ShowErrMSG )

{

Application->MessageBox(AS.c\_str(),"ОШИБКА", MB\_OK);

}

INFO("Не возможно открыть последовательный порт Error ")

// установить состояние порта - закрыт

portState = 0;

UpdateProgControls(portState); // когда порт открыт разрешает кнопки или запрещает если закрыт

return;

}

else

{

INFO("Порт "+AnsiString(ComN)+" открыт")

}

dcb->DCBlength=sizeof(DCB);

// получить текущие настройки

GetCommState(ComPort,dcb);

// скорость

//dcb->BaudRate=CBR\_115200;

//dcb->BaudRate=CBR\_57600;

//dcb->BaudRate=CBR\_38400;

dcb->BaudRate=CBR\_19200;

//dcb->BaudRate=28800;

// Количество бит

dcb->ByteSize=8;

//

dcb->fBinary = true;

//Задает игнорирование всех операций чтения/записи

//при возникновении ошибки. Если это поле равно TRUE,

//драйвер прекращает все операции чтения/записи для

//порта при возникновении ошибки.

//Продолжать работать с портом можно будет только после

//устранения причины ошибки и вызова функции ClearCommError.

dcb->fAbortOnError = false;

// четность

dcb->fParity = false; // off

dcb->Parity = NOPARITY; //Бит четности отсутствует

// CTS

dcb->fOutxCtsFlow = false; // off

// DSR

dcb->fOutxDsrFlow = false; // off

// DTR

dcb->fDtrControl = DTR\_CONTROL\_DISABLE; // off

dcb->fDsrSensitivity = false;

// Задает использование XON/XOFF управления потоком при передаче

dcb->fOutX=false; //off

// Задает использование XON/XOFF управления потоком при приеме

dcb->fInX=false; //off

// Задает режим управления потоком для сигнала RTS.

// dcb->fRtsControl=RTS\_CONTROL\_DISABLE; //off

dcb->fRtsControl=RTS\_CONTROL\_ENABLE; //2017 Enables the RTS line when the device is opened and leaves it on.

// это не handshaking, это просто дает переход в 0 линии RTS, после открытия порта

// можно сделать и DISABLE - это ничего не меняет. В программаторе линии RST и CTS

// никуда не ведут

// Задает количество стоповых бит.

dcb->StopBits=ONESTOPBIT; //1

// убрано при переходе на CBuilder6 dcb->wReserved = 0;

dcb->fNull = false;

//если TRUE, то передача продолжается, несмотря на то,

// что приемный буфер содержит более XoffLim символов

dcb->fTXContinueOnXoff = true;

//--- установить настройки ---

if( 0 == SetCommState(ComPort,dcb))

{

INFO("SetCommState-> Ошибка установки параметров")

// закрываем порт HANDLE ComPort

CloseHandle(Form1->Prog->ComPort);

// установить состояние порта - закрыт

portState = 0;

UpdateProgControls(portState); // когда порт открыт разрешает кнопки или запрещает если закрыт

return;

}

// --- Тайм ауты ---

// Максимальное время, в миллисекундах, допустимое

//между двумя последовательными символами

//считываемыми с коммуникационной линии

//ct.ReadIntervalTimeout=MAXDWORD;

ct.ReadIntervalTimeout=0;

// Задает множитель, в миллисекундах, используемый

//для вычисления общего тайм-аута операции чтения.

//Для каждой операции чтения данное значение умножается

// на количество запрошеных для чтения символов.

ct.ReadTotalTimeoutMultiplier=0;

//Задает константу, в миллисекундах, используемую для

//вычисления общего тайм-аута операции чтения.

// Для каждой операции чтения данное значение

//прибавляется к результату умножения

//ReadTotalTimeoutMultiplier на количество запрошеных

//для чтения символов. Нулевое значение полей

//ReadTotalTimeoutMultiplier и ReadTotalTimeoutConstant

//означает, что общий тайм-аут для операции чтения

//не используется.

ct.ReadTotalTimeoutConstant=0;

// Задает множитель, в миллисекундах, используемый

//для вычисления общего тайм-аута операции записи.

// Для каждой операции записи данное значение

//умножается на количество записываемых символов.

ct.WriteTotalTimeoutMultiplier=0;

//Задает константу, в миллисекундах, используемую для

//вычисления общего тайм-аута операции записи.

// Для каждой операции записи данное значение прибавляется

// к результату умножения WriteTotalTimeoutMultiplier

//на количество записываемых символов.

//Нулевое значение полей WriteTotalTimeoutMultiplier и

//WriteTotalTimeoutConstant означает, что общий тайм-аут

//для операции записи не используется.

ct.WriteTotalTimeoutConstant = 0;

// Установить тайм ауты

SetCommTimeouts(ComPort,&ct);

// получить текущие настройки

GetCommState(ComPort,dcb1);

INFO("SetCommState-> Установлена скорость :"+AnsiString(dcb->BaudRate))

// очистить внутренние буфера

PurgeComm(ComPort,PURGE\_TXCLEAR | PURGE\_RXCLEAR | PURGE\_TXABORT | PURGE\_RXABORT );

// установить событие - порт открыт

openPortEvent->SetEvent();

// установить состояние порта - открыт

portState = 1;

// послать break программеру - он должен перейти в DCC

SetCommBreak(ComPort);

WaitForSingleObject(MOvr1->hEvent,200);

ClearCommBreak(ComPort);

ResetEvent(MOvr1->hEvent);

UpdateProgControls(portState); // когда порт открыт разрешает кнопки или запрещает если закрыт

//---------------------------- это тестовый код, искал причину синего экрана на Win7 64bit SP1. Проблема проявляется только с новыми драйверами FTDI.

// если COM порт аппаратный – синего экрана нет. Баг в драйвере.

#if 0

// эта последовательность вызывает синий экран когда i==300

// т.е. вторая операция чтения если после первой не делать PurgeComm(), а делать после второй

int i=0;

DWORD cntrR=0xff, cntrW=0xff;

byte bbb, bbb1;

unsigned int tmp;

do {

bbb= PR\_READY;

ResetEvent(MOvr1->hEvent);

ReadFile(ComPort,&bbb,1,&cntrR,MOvr1); //

if( i==300 )

{

ResetEvent(MOvr1->hEvent);

ReadFile(ComPort,&bbb1,1,&cntrW,MOvr1); //

}

PurgeComm(ComPort,PURGE\_TXCLEAR | PURGE\_RXCLEAR | PURGE\_TXABORT | PURGE\_RXABORT );

Sleep(50);

INFO("PROG::doOpenPort-> "+AnsiString(ComN)+" call ReadFile+ Purge) "+AnsiString(i));

Form1->ProcessWinMessages(1);

i++;

}

while(1);

#endif

INFO("PROG::doOpenPort-> END -OK");

}

**Включение программатора в режиме командной станции**

Вот рутина, которая вызывается при включении программатора.

Она устанавливает таймаут связи с ПК и отправляет первую DCC команду – команду reset всем декодерам: 00 00 00

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

// перевести программер и декодер в DCC mode

int PROG::Switch\_to\_DCC\_mode(void)

{

char Comm[10];

byte AckProgrammer=0xff;

int rez;

DWORD cntr=0xff;

unsigned int tmp,tmp1;

LPVOID lpMsgBuf;

// Reset\_Programmer(); // посылает BREAK программеру (он выключит POWER на RAIL и перезагрузится)

// дать декодеру включиться

INFO("Switch\_to\_DCC\_mode->шлем DCC Reset 00 00 00")

Comm[0] = 0;

Comm[1] = 0;

if( 0!=OP\_Mode\_Command((byte \*)&Comm,2) )

{// наверно нет связи

return -1;

}

//--------------------- SET DCC TIMEOUT ----------------------------

// установить тайм аут приема команд от PC

// если нет команд за это время - программер выключит POWER

INFO("Switch\_to\_DCC\_mode->устанавливаем тайм аут приема DCC команд")

// проверяем готовность программера

if( 0!=check\_programmer\_ready() )

return -1;

// до отправки команды поставить на чтение

ResetEvent(MOvr1->hEvent);

ReadFile(ComPort,&AckProgrammer,1,&cntr,MOvr1);

// отправить команду

Comm[0] = PR\_SET\_DCC\_TIMEOUT;

Comm[1] = 4; // количество байт в посылке - это для программера

Comm[2] = (Config.DCCRxTimeOut & 0xFF00) >> 8;

Comm[3] = Config.DCCRxTimeOut & 0x00FF;

SendBytesToCOM(Comm,Comm[1]); // отправить байты

// ждем подтверждения приема команды от программера

tmp = WaitForSingleObject(MOvr1->hEvent,1000);

ResetEvent(MOvr1->hEvent);

// очистить внутренние буфера и отменить все операции ввода/вывода

PurgeComm(ComPort,PURGE\_TXCLEAR | PURGE\_RXCLEAR | PURGE\_TXABORT | PURGE\_RXABORT );

if( WAIT\_OBJECT\_0 != tmp )

{// нет подтверждения от программера

INFO("Switch\_to\_DCC\_mode->Нет подтверждения от программера")

tmp1 = GetLastError();

FormatMessage(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL, tmp1, MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), // Default language

(LPTSTR) &lpMsgBuf, 0, NULL);

INFO("GetLastError: "+AnsiString(tmp1)+" - "+(const char \*)lpMsgBuf)

SetLastError(0);

return -1;

}

if( AckProgrammer != PR\_ACK\_DCC\_STOP )

// нет подтверждения приема посылки

return -1;

INFO("Switch\_to\_DCC\_mode->Programmer принял команду SET DCC TIMEOUT")

//--------------------- END SET DCC TIMEOUT ----------------------------

INFO("Switch\_to\_DCC\_mode->Программер перезагружен")

return 0; // ok

}

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

Если перед включением программатора в DCC mode он мог быть в режиме программирования, то перед выполнением Switch\_to\_DCC\_mode(void)

нужно сделать сброс программатора, для этого нужно передать по RS232 сигнал break длительностью 200мс, пример кода:

volatile LPOVERLAPPED MOvr1;

MOvr1 = new OVERLAPPED;

MOvr1->Internal=0;

MOvr1->InternalHigh=0;

MOvr1->Offset=0;

MOvr1->OffsetHigh=0;

MOvr1->hEvent = CreateEvent(NULL,true,false,"My1\_NameMOvr1"); // manual reset

ResetEvent(MOvr1->hEvent);

SetCommBreak(ComPort);

WaitForSingleObject(MOvr1->hEvent,200);

ClearCommBreak(ComPort);

ResetEvent(MOvr1->hEvent);

// декодер будет отключать питание - подождать пока не закончит

Sleep(200);

**Аксессуарные команды (переключение стрелки)**

рутина для переключения стрелки делает команду **Basic Accessory Decoder Packet Format**

(**Extended Accessory Decoder Control Packet Format** не реализован, но это легко добавить в код)

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

// создает команду для переключения выхода

int T\_ACC\_DECODER::SwitchOutput(int NOutput) // NOutput – это номер выхода декодера, может быть = 0,1,2,3

{

byte NewState,b;

if( NOutput >3 )

return -1;

//--- делаем команду

// 10aa.aaaa 1AAA.CDDD ##################################################################################

// ||||

// ||||--- бит\_0 байта2 направления (Red=0, Green=1)

// |||

// |+---- биты\_1 и 2 - номер выхода

// +-----ДОЛЖЕН БЫТЬ=1

// LSB MSB ( MSB инвертированы)

DCC\_Command[NOutput].Command[0] = 0x80 | (DecAddr & 0x3F);

b = (DecAddr & 0x01c0)>>2; // 3 старших бита в старшую тетраду

b = (0xff - b) & 0xF0; // инвертировать их

b = 0x88 | b;

DCC\_Command[NOutput].Command[1] = b | (NOutput<<1) | NewState;

DCC\_Command[NOutput].Len = 2;

DCC\_Command[NOutput].Prioritet = 0;

return NewState; // >0 ok

}

**Константы**

// команды программеру

#define PR\_WHY\_ARE\_YOU 0x99

#define PR\_SET\_RS232\_SPEED 0x91 // изменить скорость RS232, 3-й байт - скорость. (2-й - это кол-во байт в команде)

// Servise mode

#define PR\_READY 0x7a // проверка готовности (только если в DCC\_MODE) - в ответ программер посылает PR\_ACK\_DCC\_READY

#define PR\_OPCODE\_SM\_COMMAND 0x70 // выполнить команду Servise Mode Direct CV Addressing

#define DEC\_POSITIVE\_ACK 0x78 // положительно

#define DEC\_NEGATIVE\_ACK 0x79 // отрицательно

// Operational mode

#define PR\_OPCODE\_OM\_COMMAND 0x60

#define PR\_SET\_DCC\_TIMEOUT 0x61

#define DCC\_TIMEOUT 2500 // 2 байта (тики по 0.4096мс) 2500~ 1сек

#define PR\_ACK\_DCC\_NEXT 0x61 // команда принята и можно следующую

#define PR\_ACK\_DCC\_STOP 0x62 // команда принята следующую нельзя

#define PR\_ACK\_DCC\_READY 0x63 // команда выполнена и можно следующую

#define PR\_OVERLOAD 0xDE // программер посылает 2 байта ee,ee

#define OVERLOAD\_RETURN\_CODE -100

// SCI mode

#define PR\_SWITCH\_TO\_SCI\_MODE 0x80

#define PR\_SEND\_A0B0 0x8A // 3-й байт содержит A0 или B0

#define PR\_SET\_POLARITY 0x81 // 3-й байт содержит A0 или B0 - это полярность которая правильная