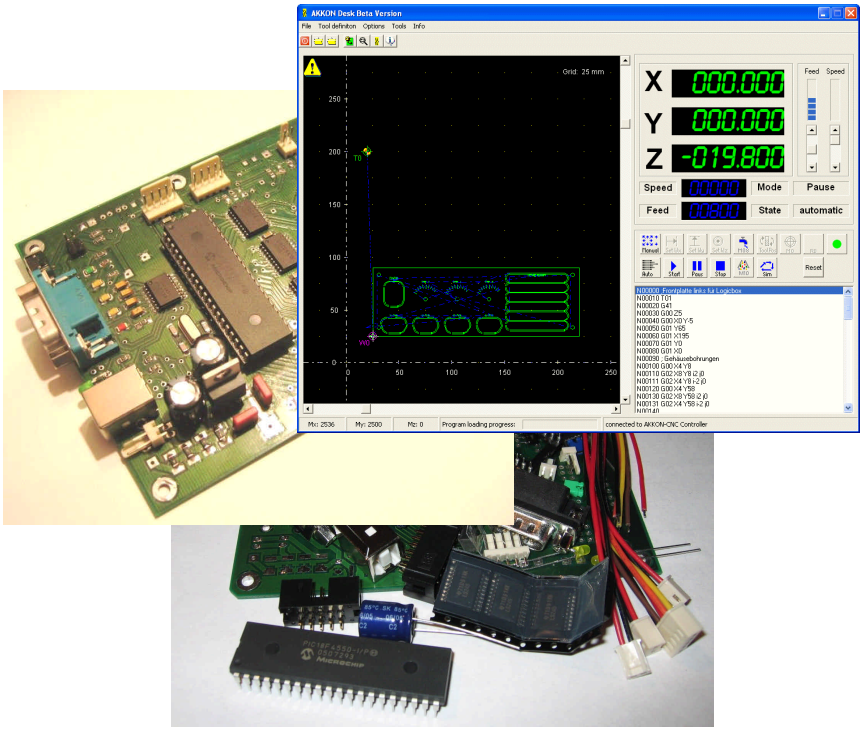
TN007

The AKKON USB CNC controller kernel

Programming interface for CNC applications and automation

Designed for the AKKON USB controller board



Authors: Gerhard Burger
Version: 1.0
Last update: 22.01.2006
File:
Attachments: -

Version table

Version	Date	Remarks
1.0	20.01.2006	First version
	23.01.2006	Extending document with a description of available commands and some programming examples

*PIC18F4550 is a trademark of the microchip corporation (www.microchip.com).

Table of content

1 Introduction

This document relates to the AKKON USB Controller board. The document outlines a description how to communicate with the AKKON USB CNC controller kernel for realizing CNC applications or to solve problems in the area of measurement and automation. The AKKON USB CNC controller kernel is a firmware that can be used with the AKKON USB controller board for realizing CNC applications. The kernel has the tasks to receive commands from a host system, decodes the commands, creates the output for controlling the hardware and returns information about the current state of the CNC-controller and hardware.

The document starts with a general description of the different command classes that can be sent to the controller. After that the protocol for exchanging information between a host system and the controller board will be discussed by code sections. In a further section a list of available commands that are currently implemented in the AUCCK will be discussed. The document ends with some programming examples that show how to communicate with the AKKON USB CNC controller kernel with PC applications. Examples are held in the C and Delphi programming language but the code should easily be adapted to other compilers.

Please read the disclaimer at the end of the document

Pleas note: As the AUCCK is currently under construction information that will be outlined in this document could already be not up to date. For actual information please visit www.burger-web.com

2 Command classes

The AUCB operates with pre defined commands that are sent by a host system (PC) and decoded in the controller. The AUCCK principally distinguishes between two classes of commands. Real time commands and sequence commands. Independent of the command class the structure of a command is comprised of 1 byte command identifier, 1 byte length of parameters in bytes and the parameters.

CMD_ID	PARAM_LEN	PARAM 0	PARAM 1	PARAM 2	...	...	PARAM n

Figure 1: Structure of a command that will be understood by the AKKON USB CNC controller kernel

2.1 Real time commands

Real time commands are commands that will be immediately executed after encoding in the receiver routine (ProcessInput). An example of a real time command is e.g. to switch on the cooling system in manual mode or step motor if the machine is in manual mode but real time commands will also be executed like e.g. to reduce the feed in automatic mode by the user or to pause and restart a current running CNC-program.

2.2 Sequence commands

Sequence commands are used for command sequences being processed by the AKKON USB CNC controller kernel. Sequence commands are stored in a FIFO-ring buffer to achieve continuous output data stream for controlling the stepper motors. Sequence commands have the same data format as real time commands. Only data transmission to the AUCB is optimized by sending larger data packages.

3 Principals of USB communication

The AKKON USB controller board operates as a device with a PC as host. Communication is always initiated by the PC. All data transmission can be done by using the one interface method:

“sendReceivePackage(sendbuf,sendbuflen,inbuf,inbuflen,timeout)

ParamNo	Name	Type	Description
1	Sendbuf	Pointer to send buffer	holds a buffer to the data
2	Sendbuflen	Unsigned int	the length of the data that has to be sent
3	Inbuf	Pointer to receive buffer	holds a buffer to the received data
4	Inbuflen	Unsigned int	holds the length to the received buffer
5	Timeout	Unsigned int	specifies the allowed time for sending and receiving of the data.

Figure 2: Parameters of the send and receive method

The implementation of sendReceiveBuf is shown in Figure 3.

```
Function SendReceivePacket(SendData :PByteBuf; SendLength : DWORD ; ReceiveData : PByteBuf;
         var ReceiveLength : DWORD; SendDelay :Word; ReceiveDelay:Word):DWORD;
var
   SentDataLength:  DWORD ;
   ExpectedReceiveLength: DWORD;
Begin
  ExpectedReceiveLength := ReceiveLength;
  if(myOutPipe <> INVALID_HANDLE_VALUE) and ( myInPipe <> INVALID_HANDLE_VALUE) then
  Begin
    if MPUSBWrite(myOutPipe,SendData,SendLength,@SentDataLength,SendDelay) <> 0 then
    Begin
      if(MPUSBRead(myInPipe,ReceiveData, ExpectedReceiveLength,@ReceiveLength,ReceiveDelay)) <> 0 then
      Begin
        if (ReceiveLength = ExpectedReceiveLength) Then
        Begin
          result:=1;   // Success!
          Exit;
        End
        else
        Begin
          if (ReceiveLength < ExpectedReceiveLength) then
          begin
            result:=2; // Partially failed, incorrect receive length
          End;
        End
      End
      else
          CheckInvalidHandle();
        End
        else
            CheckInvalidHandle();
    End;
  result:=0;  // Operation Failed
End;//end SendReceivePacket
```

Figure 3: Implementation of SendReceivePackage

Programming of the AKKON USB CNC controller kernel can be done by using the AKKON USB CNC controller library. The library is written in Delphi but should easily be adapted to other programming languages. Following interface routines are available. Routines with the Prefix SendRT relate to functions that can be used to send real time commands whereas routines with the Prefix SendSQ relate to functions for sending blocks of data to the sequence buffer.

```
Function GetSQ_Buf_Space():Integer;
Function SendSQBuf(Buf:Pointer;len:Integer;var FreeBufspace:Word):Integer;
Function SendSQ_Cmd_Int(Cmd:Byte;Param:Word):Integer;
Function SendSQ_Cmd_Long(Cmd:Byte;Param:Integer):Integer;
Function SendSQ_Cmd_Byte_Long(Cmd:Byte;Param1:Byte;Param2:Integer):Integer;
Function SendSQ_Cmd_Long_Long(Cmd:Byte;x,y:Integer):Integer;

Function SendRT_Cmd(Cmd:Byte):Integer;
Function SendRT_Cmd_Byte(Cmd:Byte;Param:Byte):Integer;
Function SendRT_Cmd_Byte_Byte(Cmd:Byte;Param1,Param2:Byte):Integer;
Function SendRT_Cmd_Int(Cmd:Byte;Param:SmallInt):Integer;
Function SendRT_Cmd_Word(Cmd:Byte;Param:Word):Integer;
Function SendRT_Cmd_long(Cmd:Byte;Param:Integer):Integer;
Function SendRT_Cmd_byte_long(Cmd:Byte;Param1:Byte; Param2:SmallInt):Integer;
Function SendRT_Cmd_byte_long_Long(Cmd:Byte;Param1:Byte;Param2,Param3:SmallInt):Integer;
Function SendRT_Cmd_Long_Long_Long(Cmd:Byte;x,y,z:Integer):Integer;

Function SendRT_Cmd_long_GetLong(cmd:Byte;Param1:Integer):integer;
Function SendRT_Cmd_long_GetVec(cmd:Byte;var Param1:VectorType):integer;
```

Figure 4: Interface for communication with the AKKON USB CNC controller kernel

4 Protocol for data transmission between PC and AUCB

Before sending new data to the AUCB the output buffer data has to be prepared for communication. For communication between PC and AUCB the protocol is a follow:

4.1 Protocol for real time commands

Real time commands are always sent as single commands to the AUCCK by the application to the controller. The protocol is shown in Figure 5.

Figure 5: Protocol of a real time command

An example of an implementation for sending a real time command to the AUCCK is shown in Figure 6.

```
Function SendRT_Cmd_Word(Cmd:Byte;Param:Word):Integer;
Var Selection : DWORD;
    RecvLength : DWORD;
    send_buf : TByteBuf;
    receive_buf :TByteBuf;
Begin
    Selection:=0; // according to our firmware, only index 0 is available!
    myOutPipe := MPUSBOpen(selection,vid_pid, out_pipe, MP_WRITE, 0);
    myInPipe  := MPUSBOpen(selection,vid_pid, in_pipe,  MP_READ,  0);
    If (myOutPipe = INVALID_HANDLE_VALUE) or (myInPipe = INVALID_HANDLE_VALUE) then
    Begin
//      memo1.lines.add('Failed to open data pipes.');
      result:=1;
      Exit;
    End;
    send_buf[0] := Cmd; // Command for Read Firmware version
    send_buf[1] :=  2;          // Expected length of the DATA result (Except 2 bytes header).
                          // change it as needed,here 0x02 indicate firmware return 2 bytes data.
    send_buf[2] := Param and $00FF;
    send_buf[3] := (Param shr 8) and $00FF;
    RecvLength := 2;  // Command + Datalen = 2 Byte
    if (SendReceivePacket(@send_buf,4,@receive_buf,RecvLength,1000,1000) = 1) Then
    Begin
      if (receive_buf[0] = Cmd) then
      Begin
        result:=0;
      End
      Else
        result:=1;
//        Memo1.lines.add('upper cerita is error!');
    ;
    End
    Else
  //    Memo1.lines.add('USB Operation Failed');
      result:=1;
    MPUSBClose(myOutPipe);
    MPUSBClose(myInPipe);
    myOutPipe := INVALID_HANDLE_VALUE;
    myInPipe := INVALID_HANDLE_VALUE;
End;
```

Figure 6: Example implementation for sending a real time command to the AUCCK

In the example above SendRT_Cmd_Word sends a command byte plus two byte parameter to the AUCCK. This function could be used for example to send the command set feed rate. In this case the first byte represents the command set feed rate, the second byte the length of the parameter (=2 byte of a word parameter) and the two bytes parameter (feed rate). If the USB operation succeed the AUCB echo the received command. Otherwise SendRT_Cmd_WORD returns with a 1 indicating that an error occurred that can be handled by the application pram.

4.2 Protocol for sequence commands

The difference between sending real time commands and sequence commands is that a sequence command contains one ore more commands that will be send in one block. The protocol of a package of sequence commands that will be sent to the AUCCK is shown in Figure 7.

CMD_ID_SQ	USB PARAM_LEN	sequence command 1	sequence command 2	sequence command 3	…	…	sequence command n

Figure 7: Protocol of a package of sequence commands.

4.3 Getting information about sequence buffer state

The sequence buffer is designed as FIFO ring buffer. On the AUCB the sequence buffer is able to hold 128 bytes. Sequence commands can be sent in larger packages as they will be buffered in the AUCCK. Every time when data are written to the sequence buffer the receivebuf contains information about the available free buffer space. The data format is as follow:

Response by the AUCB after sending block of sequence commands

CMD_ID_SQ	USB PARAM_RESPONSELLEN=2	High byte of free sequence buffer size	Low byte of free sequence buffer size

Figure 8: Return values after sending data to the sequence buffer

An implementation of SendSQBuf is shown in Figure 9.

```
Function SendSQBuf(Buf:Pointer;len:Integer;var FreeBufspace:Word):Integer;

type DataRec = Record
                 Cmd:Byte;
                 DataLen:Byte;
                 Param:Integer;
               End;
Var Selection : DWORD;
    RecvLength : DWORD;
    send_buf : TByteBuf;
    receive_buf :TByteBuf;
    myData:DataRec;

Begin
    Selection:=0; // according to our firmware, only index 0 is available!
    myOutPipe := MPUSBOpen(selection,vid_pid, out_pipe, MP_WRITE, 0);
    myInPipe  := MPUSBOpen(selection,vid_pid, in_pipe,  MP_READ,  0);
    If (myOutPipe = INVALID_HANDLE_VALUE) or (myInPipe = INVALID_HANDLE_VALUE) then
    Begin
//      memo1.lines.add('Failed to open data pipes.');
      Exit;
    End;
    fillchar(send_buf,1024,0);
    move(Buf^, send_buf[0], len);
    RecvLength := 4;  // Command + Datalen + Minor version + Major version= 4 Byte
    if (SendReceivePacket(@send_buf,len,@receive_buf,RecvLength,1000,1000) = 1) Then
    Begin
      if (receive_buf[0] = send_buf[0]) then
      Begin
        move(receive_buf[2],FreeBufSpace,2);
        result:=0;
//        Memo1.Lines.Add('Echo from Controller');
      End
      Else
      begin
        FreeBufSpace:=0;
        result:=1;
      End;
//        Memo1.lines.add('upper cerita is error!');
    End
    Else
  //    Memo1.lines.add('USB Operation Failed');
      result:=1;
    MPUSBClose(myOutPipe);
    MPUSBClose(myInPipe);
    myOutPipe := INVALID_HANDLE_VALUE;
    myInPipe := INVALID_HANDLE_VALUE;
End;
```

Figure 9: Implementation for sending sequence commands to the AUCCK

The function above send a buffer to the AUCCK. If the operation succeed the controller echo the sent command and also returns the free sequence buffer space.

5 Reading controller state

Information that includes the current state of the hardware and the controller kernel can be gathered by sending the command CNC_GET_TOTAL_STATER. In this case the controller returns a record of information of the type ControllerInfo.

```
1611 Procedure TfrmCnc.ReadControllerInfo();
1612 Var Selection : DWORD;
1613     RecvLength : DWORD;
1614     send_buf : TByteBuf;
1615     receive_buf :TByteBuf;
1616     CI:ControllerInfo;
1617
1618 Begin
1619     Selection:=0; // according to our firmware, only index 0 is available!
1620     myOutPipe := MPUSBOpen(selection,vid_pid, out_pipe, MP_WRITE, 0);
1621     myInPipe  := MPUSBOpen(selection,vid_pid, in_pipe,  MP_READ,  0);
1622     If (myOutPipe = INVALID_HANDLE_VALUE) or (myInPipe = INVALID_HANDLE_VALUE) then
1623     Begin
1624 //      memo1.lines.add('Failed to open data pipes.');
1625       Exit;
1626     End;
1627     send_buf[0] := CNC_GET_TOTAL_STATER; //
1628     send_buf[1] := 2; // send data length
1629     RecvLength := 2 + sizeof(ControllerInfo); // receive data lenth = Command + Datalen + param length
1630     if (SendReceivePacket(@send_buf,2,@receive_buf,RecvLength,1000,1000) = 1) Then
1631     Begin
1632       if (receive_buf[0] = CNC_GET_TOTAL_STATER) then
1633       Begin
1634         move(receive_buf[2],CI,sizeof(CI));
1635         MPUSBClose(myOutPipe);
1636         MPUSBClose(myInPipe);
1637         myOutPipe := INVALID_HANDLE_VALUE;
1638         myInPipe := INVALID_HANDLE_VALUE;
1639 //         ProcessControllerInfo(ci);
1640         Memo1.Lines.Add('Current Command  ' + IntToStr(Ci.LimitSwitchState));
1641         Memo1.Lines.Add('Feed  ' + IntToStr(Ci.Feed));
1642         Memo1.Lines.Add('State  ' + IntToStr(Ci.ProcessingState));
1643         Memo1.lines.add('Sequence buf count ' + inttoStr(Ci.BufCount));
1644         Memo1.Lines.Add('Current program line ' + intToStr(Ci.CurrentProgramNo));
1645         Memo1.Lines.Add('Current feed ' + IntToStr(Ci.Feed));
1646         Memo1.Lines.Add('Controlbuf count ' + IntToStr(Ci.SpindleSpeed));
1647          //... Ci. .. further parameters
1648       End
1649       Else
1650 {        Memo1.lines.add('upper cerita is error!');}
1651         ;
1652     End
1653     Else
1654 //      Memo1.lines.add('USB Operation Failed');
1655     MPUSBClose(myOutPipe);
1656     MPUSBClose(myInPipe);
1657     myOutPipe := INVALID_HANDLE_VALUE;
1658     myInPipe := INVALID_HANDLE_VALUE;
1659 End;
```

Figure 10: Implementation of ReadControllerInfo

This command can be sent by the application e.g. every 500 ms to the AUCCK. The type “ControllerInfo” is defined as shown in

```
174  Type
175     ControllerInfo = Record
176                         Feed :Word;
177                         SpindleSpeed: Word;
178                         ProcessingState:Byte;
179                         ProcessingState2:Byte;
180                         CurrentProgramNo:Word;
181                         OutCtrlState:Byte;
182                         LimitSwitchState : Byte;
183                         ApplicationTimeoutFlag :Byte;
184                         BufCount:Word;
185                         MO:PICVectorType;
186                         CurrentTool:Byte;
187                         FeedFactor:Word;
188                         SpeedFactor:Word;
189                      End;
```

Figure 11: Definition of data type ControllerInfo

5.1 Information about limit switch state

Information about limit switch state can be accessed de-masking the variable LimitSwitchState.

Bit No	Hexadecimal value	Meaning
0		not used
1		not used
2	0x04	END_POSX_MIN
3	0x08	END_POSY_MIN
4	0x10	END_POSZ_MIN
5	0x20	END_POSX_MAX
6	0x40	END_POSY_MAX
7	0x80	END_POSZ_MAX

Figure 12: Description of control bits for controlling x-, y-, z- and c-axis

5.2 Flags of indicating the controller state

Each time the command "**CNC_GET_TOTAL_STATER**" is sent to the controller it returns a structure of data type "ControllerInfo" (see above). Following table outlines the meaning of the flag of the state variables.

State variable	Bit No	Flag	Meaning
ProcessingState	0	CNC_CTRL_INT_ENABLED	Indicates if controller is in idle state
	1	CNC_APPLICATION_TIMOUT	Application time out has occurred if the PC-program sends no command to the controller during this specified time. If an application timeout occurs, the controller will go in a save state (from automatic to manual mode, set feed to 0, set speed to 0).
	2	Internal use	
	3	CNC_CTRL_AUTOMATIC	0: machine is in manual mode 1: machine is in automatic mode
	4	CNC_PROCESSING_RUNNING	Geometry processing is running (just for internal use)
	5	CNC_RUN	controller in on state
	6	CNC_SIMULATION	0: controller in standard output mode 1: controller in simulation mode
	7	Internal use	
ProcessingState2	0	CNC_R0_REACHED	Reference point has been reached
	1	CNC_T0X_REACHED	Tool change position has been reached
	2	CNC_M30	NC-Program has been finished
	3	CNC_EMERGENCY_PRESSED	Emergency button pressed

	4	Internal use	
	5	Internal use	
	6	Internal use	
	7	Internal use	
OutCtrlState	0	CNC_M08_MASK	Cooling system on
	1	CNC_M10_MASK	Vacuum cleaner on
	2	CNC_M03_MASK	Main spindle on
	3		

Figure 13: Flags for indicating controller state

Pleas note, as the AUCCK is currently under construction, the definition can change. Please look at the web for getting the actual definition.

6 List of available commands supported by the AKKON CNC kernel

Please note: commands with the ending S e.g. CNC_F00**S** are from type sequence command, commands with the ending R e.g. CNC_F00**R** are from type real time command.

No.	Command name	ID	Meaning	Parameters	Return parameters
2	CNC_F00S	5	set feed	unsigned int feed	
3	CNC_S00S	7	set speed	unsinged int speed	
4	CNC_M03S	9	spindle on clockwise		
5	CNC_M04S	10	spindle on anticlockwise		
6	CNC_M05S	11	spindle off		
7	CNC_M08S	12	cooling system on		
8	CNC_M09S	13	cooling system off		
9	CNC_M30S	14	program end		
12	CNC_PAUS	17	pause a running program by NC-program		
13	CNC_KORS	18	get position		
14	CNC_AUTS	19	set automatic mode		
15	CNC_RUNS	20	run program		
16	CNC_MANS	21	set manual mode by a NC-program		
17	CNC_InitCmdCor	22	Initialize coordinate system		
20	CNC_R00S	25	initialize R0		
21	CNC_M10S	26	vacuum cleaner on		
22	CNC_M11S	27	vacuum cleaner off		
23	CNC_SET_PROGRAM_NOS	28	set the current program number	unsigned int ProgramNumber	
24	CNC_G01_001S	101	linear interpolation in x	long x	
25	CNC_G01_010S	102	linear interpolation in y	long y	
26	CNC_G01_011S	103	linear interpolation in xy	long x, long y	
27	CNC_G01_100S	104	linear interpolation in z	long z	
28	CNC_G01_101S	105	linear interpolation in xz	long x, long z	
29	CNC_G01_110S	106	linear interpolation in yz	long y, long z	
30	CNC_G01_111S	107	linear interpolation in xyz	long x, longy, long z	
31	CNC_G02S	108	Circle G02	long x, longy, long i, long j	
32	CNC_G03S	109	Circle G03	long x, longy, long i, long j	
33	CNC_T0XS	110	Indicate that tool has to be changed		
34	CNC_SET_IMAXR	202	set all steppers to IMAX		
35	CNC_SET_vParams R	203	Set feed parameter of axis	unsinged int vxmin, unsigned int vyMin, unsigned int vzMin, unsinged int vxMax, unsinged int vyMax,	

No.	Command name	ID	Meaning	Parameters	Return parameters
				unsigned int vzMax	
47	CNC_TIMR	216	set timeout (set machine cursor timeout. Indicates the delay after a step. Specifies how long the maximum current will be hold after a step on the stepper motor card)	unsigned int Timeout	
48	CNC_PAUR	217	pause a running program by user		
50	CNC_AUTR	219	set automatic mode		
51	CNC_RUNR	220	run program		
52	CNC_MANR	221	set manual mode by user		
53	CNC_APPTIMEOUTR	222	set timeout	unsinged int Timeout	
54	CNC_WARMBOOTR	223	warm reset	reboot controller	
55	CNC_SIMULATIONR	225	set simulation mode (no output to the stepper motor cards)		
56	CNC_REALMODER	226	set standard mode (output to the stepper motor cards)		
57	CNC_GETVER	227	Read firmware version		string
64	CNC_SETWXR	236	Set W0.x	unsigned long Wx	
65	CNC_SETWYR	237	Set W0.y	unsigned long Wy	
66	CNC_SETWZR	238	Set W0.z	unsigned long Wz	
67	CNC_GETWR	239	Read W0		long Wx, long Wy, long Wz
70	CNC_SET_M0R	242	Set Machine zero point	long M0x, long M0y, long M0z, long M0w	
71	CNC_RESET_PROGRAMFILER	243	clear sequence buffer		
74	CNC_M10R	246	milk on		
75	CNC_M11R	247	milk off		
76	CNC_GET_TOTAL_STATER	248	read machine state		struct ControllerInfo
77	CNC_GETMR	249	read current coordinate of machine		long M0x, long M0y, long M0z, long M0w
78	CNC_DEBUGR	250	for debugging (reads currently the first 32 bytes of the sequence buffer)		
79	CNC_SET_FLAGS	251	set or clear bit of ProcessingsState value 2	unsigned char On, unsigned char BitNumber	
80	CNC_SETMAXFEED	252	set maximum feed of the machine		
81	CNC_ENABLE_STEPPERS	253	enable stepper motor cards		
82	CNC_SIMPLE_STEP	254	move a single step		

Figure 14: Available commands an its meaning of the AKKON CNC kernel

7 Speed and feed setup

Feed and speed are controlled each by two values.

7.1 The feed parameters

Feed is controlled by two values of type unsigned int. Once the value “MaxFeed” and twice the value “Feed”. “MaxFeed” is set by the command CNC_SETMAXFEED and saved on the controller. It represents the maximal possible feed that is currently allowed on the machine. The second value is value feed represents the feed value itself. This value is set by the command CNC_SETFEED.

```
unsigned int SetFeed(unsigned int feed)
{
    long myFeed;

    CNCparams.Feed = feed;
    if (feed == 0) return feed;
    myFeed = (long)feed * CNCparams.FeedFactor;
    myFeed = myFeed / CNCparams.MaxFeed;

    ..snip
    // myFeed holds the new set point value of the feed
}
```

Figure 15: Source code example for setting feed value

7.2 The speed parameters

Speed value works with the same principal as the feed values. Depending on these variables the Pulse Width Module output of the controller is set. Possible values are between 0..900.

Pulse-Pause Value of PWM [%] =Speed / MaxSpeed * 1024; If the output value is larger than 900 then the value will be set to 900. Values between 0..980 generate a pulse of around 95% and pause of 14% during one period.

```
// MaxSpeed holds the current maxium speed that was set by SetMaxSpeed
// CNCparams.SpeedFactor holds the current speed value
// => speed control can be done between 0..100% of maximum speed value
//  PWM-Value = speed/MaxSpeed * 1024;   0..1024
// please note, PWM values should be not bigger than 980 if the
// phase controller module is used
unsigned int SetSpeed(unsigned int speed)
{
    long mySpeed;
    unsigned int PhaseValue;

    CNCparams.SpindleSpeed = speed;
    mySpeed = (long)speed * CNCparams.SpeedFactor;
    mySpeed = mySpeed / CNCparams.MaxSpeed;
    PhaseValue = (unsinged int) mySspeed / 32;

    CCPR1L = (PhaseValue >> 2) & 0x00FF;
    if ((PhaseValue & 0x0002) != 0) CCP1CONbits.DC1B1 = 1;
                                    CCP1CONbits.DC1B1 = 0;
    if ((PhaseValue & 0x0001) != 0) CCP1CONbits.DC1B0 = 1;
                                    CCP1CONbits.DC1B0 = 0;
    return speed;
}
```

Figure 16: Source code for setting phase controller value on AKKON CNC kernel

8 Example source for using CNC commands

8.1 Switch on cooling system

SendRT_Cmd(CNC_M08R);

8.2 Move to tool change position

```
CmdGen.ClearBuf();
SendRT_Cmd(RS_RESET_PROGRAMFILER);
SendRT_Cmd(CNC_AUTR);
CmdGen.Add_Byte_Long(101,0,Round(zUp/Resolution));
CmdGen.Add(CNC_MO5S);
CmdGen.Add_Byte_Long_Long(106,0, Round(xBack/Resolution), Round(yBack/Resolution));
CmdGen.Add_Word(CNC_TOXS,0);
CmdGen.Add(CNC_MANS);
size:=CmdGen.GetBufSize();
SendSQBuf(@CmdGen.Buffer,size,  CncParams.FreeBufSpace);
```

Figure 17: Source code example for moving to tool change position

In this example CmdGen is an object that creates the data for the sequence buffer commands.

The process for moving to the tool change position is as follow:

1. Reset sequence buffer
2. switch to automatic mode
3. linear interpolation move in z-axis zUp/Resolution [steps]
4. switch spindle off
5. linear interpolation in x- and y-axis xback/Resolution and yback/Resolution [steps]
6. tell PC application that tool change position has been reached
7. set to manual mode

8.3 Set speed

```
procedure TfrmCNC.sbSpeedChange(Sender: TObject);
begin
  SendRT_Cmd_Word(CNC_S00R,Round(sbSpeed.Position));
  pbSpeed.Position:= sbSpeed.Position;
end;
```

Figure 18: Source code example for setting speed command

8.4 Movement in manual mode

Bit No	Hexadecimal value	Meaning
0	$01	Step in X
1	$02	Step in Y
2	$04	Step in Z
3	$08	Step in C (currently not implemented)
4	$10	Direction bit for X
5	$20	Direction bit for Y
6	$40	Direction bit for Z
7	$80	Direction bit for C (currently not implemented)

Figure 19: Description of control bits for controlling x-, y-, z- and c-axis

Example:

```
SendRT_Cmd_Byte(CNC_STPR,$11 + $02);
```

Figure 20: Source code example for movement in manual mode

In the example above, the AKKON CNC kernel generates output for the x-axis in left direction and in the right direction for the y-axis. This command e.g. could be send every 250 ms to the controller. As long as the parameter values not change the controller will generate an output for the stepper motor card until zero values as parameters are send; The feed rate depends on the feed that was selected before.

8.5 Set machine zero point

Following piece of code shows how to initialize the coordinate system held in the AKKON CNC kernel. In example it is assumed that the main spindle has reached the maximum position in x-, y- and z-axis. Certainly M0 can be set on every other position, according to the specific machine

```
procedure TfrmCNC.btnM0Click(Sender: TObject);
begin
  SendRT_Cmd_long_long_long(CNC_SETM0,Round(XMAX/Resolution),
                                      Round(YMax/Resolution),
                                      Round(ZMax/Resolution));
end;
```

Figure 21: Source code example for setting machine zero point

9 Disclaimer:

9.1 Limited Warranty and Disclaimer of Warranty

THIS SOFTWARE AND ACCOMPANYING WRITTEN MATERIALS (INCLUDING INSTRUCTIONS FOR USE) ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. FURTHER, the author DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF USE, OF THE SOFTWARE OR WRITTEN MATERIALS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. IF THE SOFTWARE OR WRITTEN MATERIALS ARE DEFECTIVE YOU, AND NOT the author OR ITS DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES, ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION.

THE ABOVE IS THE ONLY WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, THAT IS MADE BY the author, ON THIS PRODUCT. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY the author, ITS DEALERS, DISTRIBUTORS, AGENTS OR EMPLOYEES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY AND YOU MAY NOT RELY ON ANY SUCH INFORMATION OR ADVICE.

NEITHER the author NOR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION OR DELIVERY OF THIS PRODUCT SHALL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE SUCH PRODUCT EVEN IF the author HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

9.2 ACKNOWLEDGMENT

BY USING THIS PRODUCT YOU ACKNOWLEDGE THAT YOU HAVE READ THIS LIMITED WARRANTY, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS' TERMS AND CONDITIONS. YOU ALSO AGREE THAT THE LIMITED WARRANTY IS THE COMPLETE AND EXCLUSIVE STATEMENT OF AGREEMENT BETWEEN THE PARTIES AND SUPERSEDE ALL PROPOSALS OR PRIOR AGREEMENTS, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN THE PARTIES RELATING TO THE SUBJECT MATTER OF THE LIMITED WARRANTY.