APPLICATION NOTE

AN022: The Bootloader Protocol

Document Revision V2.00 • 2018-APR-04

This application note describes the protocol used to update the firmware of a Trinamic module or a Trinamic evaluation board. Under most cicumstances the firmware update function of the TMCL-IDE should be used to perform a firmware update. But sometimes it might be desired to integrate the module firmware update into custom PC software / host software.

Contents

TRINAMIC
MOTION CONTROL

1 Introduction

With most TRINAMIC modules the software running on the microprocessor of a module consists of two parts: the bootloader and the firmware itself. Whereas the bootloader is installed during production and testing at TRINAMIC and usually remains untouched throughout the whole lifetime of the module, the firmware can be updated by the user. During firmware updates the bootloader supports communication with the host and transfer of the new firmware to the internal flash memory of the microcontroller on the module.

After power-on or reset the bootloader code will be executed first. The bootloader will verify the checksum of the firmware in the flash memory in order to decide whether a valid firmware is installed. In case the checksum is invalid the bootloader will remain active and so the module will stay in bootloader mode. In case the checksum is valid the bootloader will start the application firmware and the module enters normal mode. The bootloader can also be activated from the firmware (command TMCL_Boot) in order to initiate a firmware update.

The bootloader commands used for firmware updates are described in the following chapters. The basic communication protocol supported by the bootloader is the same as used for standard TMCL communication with the application firmware. We assume that the reader of this application note already is familiar with the TMCL communication protocol.

On most modules the bootloader supports communication via all interfaces the module is equipped with. When using the RS232, RS485 or USB interface in bootloader mode the module ID will always be 1 and the reply ID will always be 2 (regardless of other module settings). With RS232 and RS485 the baud rate will always be 115200bps in bootloader mode. When using the CAN interface in bootloader mode, the CAN receive ID of the module will always be 1, the CAN send ID of the module will always be 2 and the CAN bit rate will always be 1MBit/s.

2 Bootloader Commands

2.1 TMCL_Boot

Parameter	Value / Description
Opcode	242
Type	81_h (Magic Code)
Motor/Bank	92_h (Magic Code)
Value	$A3B4C5D6_h$ (Magic Code)

This command switches the module into bootloader mode. This is necessary in order to perform a firmware update as the update procedure is done by the bootloader. In case the module already is in bootloader mode this command will have no effect.

In contrast to most other TMCL commands there will be no reply with this command.

2.2 TMCL_GetVersion

Parameter	Value / Description
Opcode	136
Type	0 - as string 1 - binary
Motor/Bank	Not used, always set to 0.
Value	Not used, always set to 0.

This command returns the bootloader software version. It works just the same way (and in fact is the same command) as the GetVersion command in normal TMCL firmware. The version number of the bootloader and the module type can either be returned as an ASCII string or in binary format.

In binary format the value field of the reply will contain the module type and version number as follows:

- First byte: module number, high byte.
- Second byte: module number, low byte.
- Third byte: version number, high byte.
- Fourth byte: version number, low byte.

In case a string is requested a special format will be used for the reply. The reply then consists of the reply ID (one byte) and eight bytes containing the ASCII string. This ASCII string contains the module number (four digits), the letter B and the version number of the boot loader (three digits), for example "1110B102".

2.3 TMCL_BootEraseAll

Parameter	Value / Description
Opcode	200
Type	Not used, always set to 0.
Motor/Bank	Not used, always set to 0.
Value	Not used, always set to 0 .

This command erases the currently installed firmware. Execute this command before installing any new firmware. Please note that erasing the flash memory may take some time (depending on the module between one second and 30 seconds).

2.4 TMCL_BootWriteBuffer

Parameter	Value / Description
Opcode	201
Type	Index into buffer, low byte.
Motor/Bank	Index into buffer, high byte.
Value	4 bytes (one 32-Bit-word).

This command is used to fill the temporary page programming buffer. It writes a 32 bit word to the buffer at the position specified by the type and the motor/bank parameter. The resulting index is Motor/Bank*256+Type.

Use this command to fill the temporary buffer. After the temporary buffer has been filled a flash memory page can be programmed using the command TMCL_BootWritePage (section 2.5).

The page size depends on the MCU used on the module. Inquire the page size using the TMCL_BootGetInfo command (section 2.9) before starting to download a firmware to the module in order to be sure to fill the buffer with the correct number of bytes.

2.5 TMCL_BootWritePage

Parameter	Value / Description
Opcode	202
Type	Not used, always set to 0.
Motor/Bank	Not used, always set to 0.
Value	Physical memory address (start address of page to be programmed).

The TMCL_BootWritePage command copies the contents of the temporary buffer to the flash memory page specified by the physical flash memory address given in the value field. Please note that programming a memory page may take some time (up to a few seconds depending on the MCU used on the module).

2.6 TMCL_BootGetChecksum

Parameter	Value / Description
Opcode	203
Type	Not used, always set to 0.
Motor/Bank	Not used, always set to 0 .
Value	Last address

This command calculates the checksum of the specified flash memory area. The calculation starts at the lowest possible application firmware address and ends at the specified address. The specified address has to be an absolute memory address. The calculated checksum is a 32 bit value. All memory values (bytes) inside the specified address range will be added up and the lower 32 bits of the result will be returned in the value field of the reply.

The result of this command can then be compared to the checksum of the firmware file. These two checksums must be equal.

2.7 TMCL_BootReadMemory

Parameter	Value / Description
Opcode	204
Type	Not used, always set to 0.
Motor/Bank	Not used, always set to 0.
Value	Address (has to be a multiple of 4).

Using this instruction the contents of the flash memory can be read back. Four bytes (a 32 bit word) will be read back and returned in the value field of the reply. The address supplied with this command has to be

an absolute memory address, starting from the first possible application memory address. The address must be a multiple of four.

This command is normally not needed when performing a firmware update. It is just there for debugging purposes (for example when developing an own update tool).

2.8 TMCL_BootStartAppl

Parameter	Value / Description
Opcode	205
Type	Not used, always set to 0.
Motor/Bank	Not used, always set to 0.
Value	Not used, always set to 0.

This command starts the application firmware in the flash memory. After the entire firmware has successfully been written to the flash memory and also length and checksum have been written to the flash memory, this command can be used to start the new firmware (i.e. to switch the module back from bootloader mode to normal mode).

2.9 TMCL_BootGetInfo

Parameter	Value / Description
Opcode	206
Type	0 - get size of one flash memory page 1 - get start address of application memory area 2 - get overall size of flash memory
Motor/Bank	Not used, always set to 0.
Value	Not used, always set to 0.

Depending on the type field, this command returns some important information about the flash memory of the module. The following information can be inquired:

- Type=0: The size of one flash memory page will be returned. This also is the number of bytes that can be written into the temporary buffer using the TMCL_BootWriteBuffer command.
- Type=1: The first address of the application memory area will be returned. The first address used in the HEX file must be the same as this value. Otherwise the HEX file cannot be used for this module.
- Type=2: The overall size of the flash memory will be returned. This value can be used to check whether the contents of the HEX file will fit into the memory of the module.

Always use this command to inquire these values and never rely on any fixed values. Different modules might be equipped with different MCUs and hence also the flash memory parameters might be different.

2.10 TMCL_BootWriteLength

Parameter	Value / Description
Opcode	208
Type	0 - write length 1 - write checksum
Motor/Bank	Not used, always set to 0.
Value	Length or checksum.

Depending on the type parameter, this command either writes the length or the checksum of the application to the flash memory. Always write the length first and then the checksum, as on most modules (depending on the MCU used on the specific module) it is only possible to write the two values in this order.

3 Firmware Update Procedure

The following steps describe the proposed firmware update procedure.

1. Load the new firmware into the PC memory and calculate the checksum (add-up all byte values and use the lower 32 bits of the result). It should be verified that the new firmware file is suitable for the connected module. This can be done by searching for the version string inside the firmware file. The format of this version string is "####V###" or "###V#.##", where # stands for any number (0...9).
2. Send the TMCL_Boot command to the module. Please note: there will be no response from the module for this command.
3. Get the size of the flash memory and the page size using TMCL_BootGetInfo commands.
4. Erase the flash memory using the TMCL_BootEraseAll command. Erasing the flash memory might take several seconds. The reply for this command will be sent at the end of this process.
5. Transfer the new firmware to the module using TMCL_BootWriteBuffer and TMCL_BootWritePage commands. The flash memory has to be filled page by page. First, fill the temporary buffer with as many 32 bit words as a page can hold (page size read using TMCL_BootGetInfo divided by four). Then, issue a TMCL_BootWritePage command with the start address of the page to be programmed. Repeat these steps until the entire firmware has been transferred.
6. Get the checksum from the module using the TMCL_BootGetChecksum command. The end address being sent with this command has to be the start address of the application area (inquired using a TMCL_BootGetInfo command) plus the length of the new firmware.
7. Verify the checksum from the module against the checksum calculated when the firmware has been loaded from the HEX file (in the first step).
8. In case both checksums are identical the update process has been successful. Now write the length of the application and the checksum of the application to the flash memory using two TMCL_BootWriteLength commands.
9. The new firmware can now be started using the TMCL_BootStartAppl command.
10. In case the checksums are not identical do not write the length and checksum to the memory and also do not start the new firmware. Instead, erase the flash memory and repeat downloading the new firmware.

In case the firmware update procedure has been interrupted (for example due to a power glitch during the update process) the module will stay in bootloader mode, as the bootloader detects that there is no valid firmware in the flash memory. The update procedure can then be repeated.

4 Code Examples

Here are two code examples. Section 4.1 shows how to interpret a HEX file, and section 4.2 illustrates the boot procedure. Please note that these two examples are not complete. They are just provided to show how loading a HEX file and the bootloader protocol can be implemented.

4.1 Loading a HEX File

The following example shows how to interpret a HEX file and how to load its contents into an array of bytes. It uses the Qt environment (in fact it is an excerpt from the TMCL-IDE).

```
int FirmwareUpdate::compareWildcard(char c1, char c2)
{
    if(c1 == c2)
        return 1;

    if((c2 == '#') && (c1 >= '0') && (c1 <= '9'))
        return 1;

    return 0;
}

bool FirmwareUpdate::findVersionString(unsigned char *data, int size, char *
    mask)
{
    int i, j, l, v;

    i = 1;
    j = 0;
    l = strlen(mask);
    v = 0;
    memset(version, '_', 10);
    while((i < size) && (j < l))
    {
        if(compareWildcard(data[i+j], mask[j]))
        {
            version[v++] = data[i+j];
            j++;
        } else {
            i++;
            j = 0;
            v = 0;
        }
    }
    version[v] = '\0';

    return (v != 0);
}

QString FirmwareUpdate::loadHexFile(QString fileName)
{
    QFile file(fileName);

    int recordLength, recordAddress;
```


```
43      unsigned char recordChecksum;
        unsigned int checksum, progChecksum;
45      int lineCount = 1;
        unsigned char recordType = 0;
47
        bool firstAddress = true;
49      bool ok = false;
        int progEnd;
51
        memset(progData, 0, sizeof(progData));
53      progEnd = 0;
        progStart = 0;
55      extendedLinearAddress = 0;
        segmentAddress = 0;
57      if(!file.open(QIODevice::ReadOnly))
        {
59          qDebug() <<"could not open file!";
            return "";
61      }

63      while (!file.atEnd() && recordType !=1)
        {
65          //each line contains at least 11 characters and starts with a ':'
            QByteArray line = file.readLine();
67          if(line.count()<11 || line.at(0)!=':')
            {
69              qDebug() << "Illegal line in hex file" << endl;
                break;
71          }

73          //extract record values
            recordLength  = line.mid(1, 2).toInt(&ok,16);
75          recordAddress = line.mid(3, 4).toInt(&ok,16);
            recordType    = line.mid(7, 2).toInt(&ok,16);
77          //Record Type: 0=normal data, 1=EOF, 2=Extended Segment Address
            //4=Extended Linear Address
79          recordChecksum = line.mid(9+recordLength*2, 2).toInt(&ok,16);
            checksum = recordLength + line.mid(3, 2).toInt(&ok,16) + line.mid(5,
    2).toInt(&ok,16) + recordType;
81
            if (recordType == 0) //Record type 0: Data
83          {
                if (extendedLinearAddress < 1048576)
85              {
                  //ignore data with Extended Linear Address >1MB
87                //(these are superflous entries with some CPUs)
                    if (firstAddress)
89                  {
                        progStart = recordAddress + segmentAddress +
    extendedLinearAddress;
91                      firstAddress = false;
                    }
93                  else progStart = qMin(progStart, recordAddress +
    segmentAddress + extendedLinearAddress);
```


```
95                  for (int i = 0; i <= recordLength-1; i++)
                    {
97                      if ((recordAddress + segmentAddress +
    extendedLinearAddress) > MAX_FIRMWARE_SIZE)
                            qDebug() << "address too high";
99
                        unsigned char d = line.mid(9+i*2, 2).toInt(&ok,16) & 0
    xff;
101                     progData[recordAddress + segmentAddress +
    extendedLinearAddress] = d;
                        checksum += progData[recordAddress + segmentAddress +
    extendedLinearAddress];
103                     recordAddress++;

105                     progEnd = qMax(progEnd, recordAddress + segmentAddress +
     extendedLinearAddress);
                    }
107
                    //Calculate and compare record check sum
109                 checksum &= 0xff;
                    checksum = (256-checksum) & 0xff;
111                 if (checksum != recordChecksum)
                    {
113                     qDebug() << "record checksum error in line " <<
    lineCount;
                        return "";
115                 }
                }
117         }
            else if (recordType == 2)
119         {  //Record type 2: Segment address
                segmentAddress = line.mid(9, 4).toInt(&ok,16) << 4;
121         }
            else if (recordType == 4)
123         {  //Record type 4: Extended linear address
                extendedLinearAddress = line.mid(9, 4).toInt(&ok,16) << 16;
125             if(extendedLinearAddress>=0x1fff0000)
                {
127                 //Superfluous entry with some CPUs:
                    //Just ignore this data.
129             }
                else if ((extendedLinearAddress >= 0x100000) && (
    extendedLinearAddress <0x140000))
131             { //ARM7 address range (module with AT91SAM7 CPU)
                    extendedLinearAddress = extendedLinearAddress-0x100000;
133             }
                else if (extendedLinearAddress >= 0x8000000)
135             {  //Cortex-M0/M3/M4 address range (module with Cortex-Mx CPU)
                    extendedLinearAddress = extendedLinearAddress-0x8000000;
137             }
            }
139         else if ((recordType!=1) && (recordType !=3) && (recordType!=5))
            {  //Ignore record types 1, 3 and 5
```


```
            qDebug() << "Record type not supported!";
            return "";
        }
        lineCount++;
    }
    file.close();

    //Calculate checksum of the firmware file
    progChecksum = 0;
    progEnd--;
    for (int i = progStart; i <= progEnd; i++)
        progChecksum += progData[i];

    //Display the Checksum somewhere if desired.
    //progChecksum now contains the checksum that must be the same in the
    //module (when using TMCL_BootGetChecksum).

    //Length should not be odd.
    //Else make it one byte longer (then one dummy null byte will be sent).
    if ((progEnd-progStart+1) & 1)
        progEnd++;

    programSize = progEnd-progStart+1;
    //programSize now contains the overall size of the firmware
    //(which should at the end of programming be set using
    //TMCL_BootWriteLength).

    //Check version and module number of the firmware file
    if(findVersionString(progData, sizeof(progData),(char*)"####V###") ||
       findVersionString(progData, sizeof(progData),(char*)"###V#.##") ||
    {
        return version;
    }
    return "";
}
```

4.2 Downloading the Firmware

This is an example for implementing the boot procedure itself. It is quite lean and so can be used either on a PC or also on a microcontroller. The user will have to add some functions that are not shown in this example:

- GetSysTimer(): returns the value of an up-counting millisecond timer (at least 32 bit).
- ReadUART(): Tries to read a character from the UART or some other interface. Returns TRUE if a character could be read and FALSE if no character could be read. The character that has been read will be put into the given variable.
- WriteUART(): Writes a character to the UART.

```
#define TMCL_GetVersion       136
#define TMCL_BootEraseAll     200
#define TMCL_BootWriteBuffer  201
#define TMCL_BootWritePage    202
```


```
#define TMCL_BootGetChecksum  203
#define TMCL_BootStartAppl    205
#define TMCL_BootGetInfo      206
#define TMCL_BootWriteLength  208
#define TMCL_Boot             242


uint8_t TMCLReply[8];
int32_t TMCLReplyInt;

static uint8_t SendTMCLAndWait(uint8_t Address, uint8_t Opcode, uint8_t Type
    , uint8_t Motor, int32_t Value)
{
  uint8_t Checksum;
  uint32_t Timeout;
  uint32_t Count;
  uint8_t Byte;

  //Calculate checksum of TMCL command
  Checksum=Address+Opcode+Type+Motor;
  Checksum+=Value >> 24;
  Checksum+=Value >> 16;
  Checksum+=Value >> 8;
  Checksum+=Value & 0xff;

  //Send the command
  WriteUART(Address);
  WriteUART(Opcode);
  WriteUART(Type);
  WriteUART(Motor);
  WriteUART(Value >> 24);
  WriteUART(Value >> 16);
  WriteUART(Value >> 8);
  WriteUART(Value & 0xff);
  WriteUART(Checksum);

  //Wait for reply
  Timeout=GetSysTimer();
  Count=0;
  while(Count<9 && abs(GetSysTimer()-Timeout)<30000)
  {
    if(ReadUART((char*) &Byte)) TMCLReply[Count++]=Byte;
  }

  //got less than nine bytes => Timeout
  if(Count<9) return FALSE;

  //Decode reply value
  TMCLReplyInt=TMCLReply[4];
  TMCLReplyInt<<=8;
  TMCLReplyInt|=TMCLReply[5];
  TMCLReplyInt<<=8;
  TMCLReplyInt|=TMCLReply[6];
  TMCLReplyInt<<=8;
```


```
  TMCLReplyInt|=TMCLReply[7];

  return TRUE;
}

static void SendTMCL(uint8_t Address, uint8_t Opcode, uint8_t Type, uint8_t
    Motor, int32_t Value)
{
  uint8_t Checksum;

  //Calculate checksum of TMCL command
  Checksum=Address+Opcode+Type+Motor;
  Checksum+=Value >> 24;
  Checksum+=Value >> 16;
  Checksum+=Value >> 8;
  Checksum+=Value & 0xff;

  //Send the command, do not wait for reply
  WriteUART(Address);
  WriteUART(Opcode);
  WriteUART(Type);
  WriteUART(Motor);
  WriteUART(Value >> 24);
  WriteUART(Value >> 16);
  WriteUART(Value >> 8);
  WriteUART(Value & 0xff);
  WriteUART(Checksum);
}

static void FirmwareUpdate(void)
{
  uint32_t PageSize;
  uint32_t MemSize;
  uint32_t StartAddress;
  uint32_t PageAddress;
  uint32_t PageOffset;
  uint32_t i;
  char Dummy;

  //The following variables should be defined after loading the HEX file:
  //  progData: data to be programmed
  //  progStart: first index into progData to be programmed
  //  progEnd: last index into progData to be programmed
  //  programSize: overall size of the program
  //  progChecksum: checksum of the program

  //Check module type
  if(!SendTMCLAndWait(1, TMCL_GetVersion, 0, 0, 0))
  {
  //Compare, abort if wrong module type.
  }

  //Put module into bootloader mode
  SendTMCL(1, TMCL_Boot, 0x81, 0x92, 0xA3B4C5D6);
```


```
111
  //Just wait a short time here (one second).
113 i=GetSysTimer();
  while(abs(GetSysTimer()-i)<1000);
115
  //Get flash page size
117 if(!SendTMCLAndWait(1, TMCL_BootGetInfo, 0, 0, 0))
  {
119   return;
  }
121 PageSize=TMCLReplyInt;

123 //Get start address
  if(!SendTMCLAndWait(1, TMCL_BootGetInfo, 1, 0, 0))
125 {
    return;
127 }
  StartAddress=TMCLReplyInt;
129
  //Get size of memory
131 if(!SendTMCLAndWait(1, TMCL_BootGetInfo, 2, 0, 0))
  {
133   return;
  }
135 MemSize=TMCLReplyInt;

137 //Erase flash
  if(!SendTMCLAndWait(1, TMCL_BootEraseAll, 0, 0, 0))
139 {
    return;
141 }

143 //Page by page programming
  PageAddress=StartAddress;
145 PageOffset=0;
  i=progStart;
147 while(i<progEnd)
  {
149   //Send one 32 bit word
    if(!SendTMCLAndWait(1, TMCL_BootWriteBuffer, PageOffset/4, 0,
151     (progData[i+3]<<24)|(progData[i+2]<<16)|(progData[i+1]<<8)|progData[i
 ]))
    {
153   return;
    }
155
    PageOffset+=4;
157 i+=4;

159 //One page full or last page?
    if(PageOffset%PageSize==0 || i>=progEnd)
161 {
      if(!SendTMCLAndWait(1, TMCL_BootWritePage, 0, 0, PageAddress))
163   {
```


```
        return;
      }

      PageOffset=0;
      PageAddress+=PageSize;
    }
  }

  //Check the check sum
  if(!SendTMCLAndWait(1, TMCL_BootGetChecksum, 0, 0, StartAddress+
   programSize-1))
  {
    return;
  }

  if(programChecksum!=(uint32_t) TMCLReplyInt)
  {
    return;
  }

  //Send length and checksum
  if(!SendTMCLAndWait(1, TMCL_BootWriteLength, 0, 0, programSize))
  {
    return;
  }
  if(!SendTMCLAndWait(1, TMCL_BootWriteLength, 1, 0, programChecksum))
  {
    return;
  }

  //Start new firmware
  if(!SendTMCLAndWait(1, TMCL_BootStartAppl, 0, 0, 0))
  {
    return;
  }
}
```

5 Disclaimer

TRINAMIC Motion Control GmbH & Co. KG does not authorize or warrant any of its products for use in life support systems, without the specifc written consent of TRINAMIC Motion Control GmbH & Co. KG. Life support systems are equipment intended to support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided, can be reasonably expected to result in personal injury or death.

Information given in this application note is believed to be accurate and reliable. However, no responsibility is assumed for the consequences of its use nor for any infringement of patents or other rights of third parties that may result from its use.

Specifications are subject to change without notice. All trademarks used are property of their respective owners.

6 Revision History

Version	Date	Author	Description
V1.00	2012-MAR-19	SD	First release version, valid for stepRocker V1.xx only.
V2.00	2018-APR-04	OK	Completely revised version, valid for all Trinamic modules.

Table 1: Document Revision