



R.S.R. s.r.l. - via Montale, 3 - 20070 Bulgarograsso (CO) - Tel.: 031.3532168 - Fax: 031.891688

UCB Bootloader

Code: **RSR641**
Description UCB bootloader operations
Customer: VEII
File name: 641Bootloader04.doc

Revision	Date	Changes	Name
bozza	2014/03/14		Michele
revision	2014/04/10	0007 release	Michele
revision	2014/04/11	Code read protection (crp) added	Michele
revision	2014/04/18	Bootloader status description	Michele
revision	2014/05/16	Bootloader revision updated	Michele

1.	Project introduction	2
2.	0007 bootloader release description	2
2.1.	SD download	2
2.2.	Serial download	2
3.	Bootloader serial commands	3
4.	UPDATE BOOT	4
5.	Appendix	4

1. Project introduction

The bootloader project is identified with the IAR project file RSR641boot.eww: inside the preprocessor option is defined the macro BOOT to enable firmware features.

After the RSR641boot.hex file is generated before loading, it is necessary to run the boot.bat batch file:

```
Pchk RSR641boot.hex RSR641bootchk.hex RSR641 100 -m0x7F80="VENDORS-EXCHANGE" -m0x7F94=0x00
```

that generates the RSR641bootchk.hex, that includes all checksums.

The source of Pchk.exe is added to the appendix; this .exe add the producer signature at 0x7f80 address in the .hex file and language selection at 0x7f94.

After that the boot file is ready for upload: it could be loaded with FlashMagic tool or with the embedded application (described below).

The bootloader update project is identified with the IAR project file RSR641boot2.eww: inside the preprocessor option are defined the macro both BOOT and BOOT2 to enable firmware features.

After the RSR641boot2.hex file is generated before loading, it is necessary to run the boot2.bat batch file:

```
checksum RSR641boot2.hex RSR641aa.hex RSR641 100 -m0x8080="BOOT UPDATE" " -m0x8094=0x00
```

that generates the RSR641aa.hex, that includes all checksums.

The source of checksum.exe is the same of code of Pchk.exe, unless the offset; this .exe add the "BOOT UPDATE" signature at 0x8080 address in the .hex file and language selection at 0x8094.

After that the boot update file is ready for upload: it could be loaded with each bootloader like an ordinary vending machine fw.

2. 0007 bootloader release description

The bootloader release 0005 is the last implemented: it has two way to download the UCB firmware

1. the first (as usual) with the SD card
2. the second one with the serial line

At each startup the bootloader flash checksum is calculated: if it is correct and the SD card is present (with a valid file), a new download is started; if the SD card is not present (or is not present a valid file), the application present at 0x8000 address is run; if no valid application at 0x8000 flash address is present, the bootloader will wait the "LOAD" command on serial line to start a new download.

If at startup the bootloader flash checksum is not valid, it will try to signal the problem and the only possible operation is to update the bootloadeter

2.1. SD download

The bootloader will load the first file found on SD card with the following format:

RSR641aa<....>.hex

Inside the < > can be contained any string accepted by the OS (so it is possible identify each .hex file with the mnemonic preferred by each one); the prefix RSR641aa is kept to guarantee the compatibility with the past.

Before load the fw, the file checksum are checked with the checksum present in the UCB flash: if the fw is different from the new in the file, it will be loaded.

Before loading the file checksum is checked: if the checksum is correct the file is loaded; after the loading the flash checksum is verified. If all this verifications give a positive result, the fw is run.

2.2. Serial download

To start the serial download the Vending Machine/Uploader must send the "\x02LOAD?\x0d\x0a" command.

The com port used on the UCB is the COMM4; the com port configuration is always 9600n81 (to keep compatibility with the UCB-MIND).

After "LOAD" command is sent, the UCB will respond with "\x0d\x0a" string when ready to receive the file.

The file to send is the same used on the SD card and it has to be sent row by row: after each row the UCB will respond with a "\x0d\x0a" string.

If the "LOAD" command has been interrupted, the UCB will keep the status and at the next startup it is necessary to restart with the upload (the sending of command "LOAD" is not needed).

At the end of file, the application has to send a single "!" to end the procedure with an ACK (0x05) char.

If the file checksum (calculated during the download) and the flash checksum are correct, the fw is run.

3. Bootloader serial commands

Command structure

STXload?CR/LF

Reply

CR/LF

After receiving the **CR/LF** UCB waits for programming file records, in Intel .hex format

The download software has to send one record and wait for confirmation from UCB (**CR/LF**)

To close the programming the download software has to send the "!" (0x21) character.

When receiving the "!" character, the UCB will check the received file check-sum, and if ok the UCB is restarted and the new firmware is running.

If the check-sum is invalid, the UCB will send **NACK (0x15) CR/LF** and to restart a new download is necessary to send a new command "LOAD?".

Example :

PC > **STX load? CR/LF**

UCB > **CR/LF**

PC > **:1080E000525352363431202030313030D0B4ADF5D7 CR/LF**

UCB > **CR/LF**

PC > **:020000040000FACR/LF**

UCB > **CR/LF**

.....

PC > **!**

(UCB will check the received file, and if OK will restart)

Important: this command is present in the 7.5.H revision and following; before using it check the firmware revision!

If the serial download is interrupted/stopped, to restart a new download it is necessary to send newly the command "LOAD?".

Command structure

STXboot status?CR/LF

Reply

STXboot status xxx CR/LF

'xxx' represents the boot-loader status and can assumes these values:

- **0 - the fw has been loaded correctly:** if the fw is loaded correctly the bootloader jumps directly to the application address: for this reason the bootloader can't send back this status on the serial line;
- **1 - the bootloader can't open the mmc file:** load another file on the mmc;
- **2 - the mmc/serial file has a wrong format:** in this situation the download is stopped; to restart the mmc download you need to restart the board; to restart the serial download you must send a new "LOAD?" command;
- **3 - the download fw has a wrong dimension:** load another file on the mmc;
- **4 - the fw checksum is wrong:** in this situation the checksum of the file is wrong or different from the one calculated on the flash memory image; to restart the mmc download you need to restart the board; to restart the serial download you must send a new "LOAD?" command;

- **99 - the same fw is already loaded:** if the fw on the mmc is already loaded in the memory the bootloader jump directly to the application address: for this reason the bootloader can't send back this status on the serial line;
- **165 - serial download request:** this status is set from the UCB application when receives a "LOAD?" command on the serial line; the UCB application jumps directly to the bootloader (and then it will change its status), that waits for the file records: don't send a "BOOT STATUS?" request otherwise the download will fail with "WRONG FILE FORMAT" error (reason the bootloader can't send back this status on the serial line).
- **166 - serial download fail:** this status shows that a fw download was started and interrupted; this status can be send back from the bootloader only after a download interruption.

This command is used to check if the download has had success and the boot-loader is jumped to the UCB application and run it. If the response is the machine status

STX status;<status_word>;<credit>;<optional status1>;<optional status2>;<optional status2>; CR/LF

the UCB fw is correctly installed; if the response is one of the status shown above, a malfunction is happened.

Important: this command is present only in the bootloader revision 0007 and following; before using it check the bootloader revision!

4. UPDATE BOOT

To update the bootloader instead of the ordinary application at address 0x8000 must be loaded a specific application that load only from SD a new bootloader. The bootloader file present on the SD must be named RSR641bootchk.hex (see project introduction paragraph).

The SD card used for the update must contain the RSR641aa.hex file (the boot update application starting at 0x8000 address) and the RSR641bootchk.hex.

The bootloader present on the UCB (the one we need to replace) will load the downloader in the memory location used by the UCB Application. After that, the just installed *Downloader* will load the new boot-loader firmware, erasing the memory location of the old one. After that the UCB must be restarted: before restart it is necessary to insert the SD card with the UCB Application to load (overwriting the Downloader present in the memory), restoring the UCB functionality.

When the Downloader is flashing the new boot-loader, it adds:

- the flash signature at the address 0x00000014 (that contains the 2's complement of the check-sum of the remaining interrupt vectors);
- the code read protection (crp = 0x87654321) , to enable only erase with ISP command, after flash programming.

5. Appendix

```
//-----
#pragma hdrstop
//-----

#pragma argsused

#include <conio.h>
#include <ctype.h>
#include <io.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "crc16.h"

#define hexbin(a) (((a)<='9')?((a)-'0'):((a)-'A'+10))
#define binhex(a) (((a)<= 9)?((a)+'0'):((a)-10+'A'))

typedef unsigned char byte;
typedef unsigned int word;
typedef unsigned int WORD;
typedef unsigned long dword;
typedef unsigned long DWORD;

#define FIRMWARESTART 0x0000
#define FIRMWARE_VER 0x7FE0
#define FIRMWAREEND 0x7FF0

byte firmware[512*1024];
FILE *fp, *fout;
word nbyte;

byte flashIntelFile( DWORD _filelen );

void outHex( FILE *_fout, char *_buf )
{
```

```

int      i, crc, ver;
char     outbuf[128];

crc = 0;
for ( i = 1; _buf[i] != '\0'; i += 2 )
{
    sscanf( &_buf[i], "%2x", &ver );
    crc += ver;
}
//  sprintf( outbuf, "%s%02X\x0d\x0a", _buf, (-crc)&0xFF );
sprintf( outbuf, "%s%02X\n", _buf, (-crc)&0xFF );
fprintf( _fout, "%s", outbuf );

for ( i = 0; outbuf[i] != '\0'; i++ )
    crc_16( outbuf[i] );
}

int
{
    byte          i, inb;
    int           ch, ver;
    char          srec0[128], srec8[8+1], sbuf[256], *p;
    word          memCrc;

dword          addr, hiaddr;

printf("checksum #1.0.0\n" );
printf("in:  %s\n", argc[1] );
printf("out: %s\n", argc[2] );

memset( firmware, 0xFF, sizeof(firmware) );

if ( argv < 5 )
{
    printf( "Usage: checksum <input_file> <output_file> <project> <version> [<definition>]\n" );
    return 1;
}

    if( (fp=fopen(argc[1],"rb")) == NULL )
{
    printf( "Unable to open input file: %s\n", argc[1] );
    return 2;
}

nbyte = 0;
if ( strstr(argc[1],"bin") != NULL || strstr(argc[1],"Bin") != NULL || strstr(argc[1],"BIN") != NULL )
{
    // load binary file
    addr = FIRMWARESTART;
    while( !feof( fp ) ) {
        if ( (ch=getc(fp)) == EOF )
            break;

        firmware[addr++] = (byte)ch;
        nbyte++;
    }
}
else
{
    // load intel hex
    flashIntelFile( filelength(fileno(fp)) );
}

    fclose(fp);

// check memory options
if ( argv > 5 )
{
    clreol();
    for ( i = 5; i < argv; i++ )
    {
        printf( "opt: %s\n", argc[i] );
        // get address
        addr = 0;
        if ( sscanf( argc[i], "-m0x%x=%s", &addr, sbuf ) != 2 )
            continue;
        //printf( "\t'%s'='%s'\n", addr, sbuf );
        // get value
        if ( sbuf[0] == '0' && sbuf[1] == 'x' )
        {
            // hex
            p = &sbuf[0];
            while ( *p != '\0' )
            {
                if ( *(p+0) != '0' || *(p+1) != 'x' )
                    break;
                p += 2;
                if ( isxdigit(*p) )
                {
                    inb = (byte)((*p>='0'&&*p<='9')?(*p-'0'):(*p-'A'+10));
                    p++;
                }
                else
                    break;
                if ( isxdigit(*p) )
                {
                    inb = (byte)((inb<<4) + (*p>='0'&&*p<='9')?(*p-'0'):(*p-'A'+10));
                    p++;
                }
                else
                    break;
                if ( *p == ',' )
                    p++;
                firmware[addr++] = inb;
            }
        }
        else
        {
            // string copy
            strcpy( (char *)&firmware[addr], sbuf );
        }
    }
}

//

    if ( (fout=fopen(argc[2],"wb")) == NULL )
{
    printf( "Unable to open output file: %s\n", argc[2] );
    return 2;
}

BCC = 0;
for ( addr = FIRMWARESTART; addr < FIRMWAREEND; addr++ )
    crc_16(firmware[addr]);
memCrc = BCC;

ver = strlen(argc[3]);
if ( ver > 8 )
    ver = 8;
memset( srec8, ' ', 8 );

```

R. S. R. srl - via Montale,3 - 22070 Bulgarograsso (Como)
C.F. e P. IVA 02093840136 - Tel. 031.3532168 - Fax 031.891688- E mail: info@RSR.it

```

    _source += 2;
    *_addr |= dato;

    if ( hex2bin( &dato, _source ) != 0 )
        return 2;
    _source += 2;
    *_type = dato;

    *_len = 0;
    while( len-- > 0 )
    {
        if ( hex2bin( _dest++, _source ) != 0 )
            return 2;
        _source += 2;
        *_len = (byte) (*_len+1);
    }

    return 0;
} // intelHexConvert

/*-----
| FAT_getsFile:
|-----
| In:
| Out: byte    0   byte read
|       1   EOF
|       2   end of sector
|-----*/

byte FAT_getsFile( byte *_buf, byte *_len )
{
    byte i, dato;

    i = 0;
    *_buf = '\0';
    do {
        if ( (dato=(byte)getc(fp)) == (byte)EOF )
        {
            if ( i == 0 )
                return 1;
            *_len = i;
            return 0;
        }
        nbyte++;
        *_buf++ = dato;
        *_buf = '\0';
        i++;
        if ( i >= *_len )
            break;
    } while( dato >= ' ' ); // till cr/lf
    *_buf = '\0';
    *_len = i;
    return 0;
} // FAT_getsFile

byte flashIntelFile( DWORD _filelen )
{
    byte il, type, perc, oldperc, intelBuf[100], buflen;
    DWORD baseAddr, destAddr, len;

    // verify checksum and load file
    len = 0;
    BCC = 0;
    oldperc = 0;
    while ( len < _filelen )
    {
        buflen = sizeof(intelBuf);
        if ( FAT_getsFile(intelBuf,&buflen) != 0 )
            return 1;
        len += buflen;

        if ( buflen < 2 )
            continue;

        if ( intelHexConvert(intelBuf,&il,&type,&destAddr,intelBuf) != 0 )
            return 2; // wrong intel format

        perc = (byte) ((WORD) ((100L*(DWORD)len)/(DWORD)_filelen));
        if ( perc != oldperc )
        {
            oldperc = perc;
            printf( "LOADING %3d %% \r", perc );
        }

        if ( type == 4 )
        {
            // change base address
            baseAddr = (((DWORD)intelBuf[0]<<24)+(DWORD)intelBuf[1]<<16);
            continue;
        }
        if ( type != 0 )
            return 2; // wrong intel format

        memcpy( &firmware[baseAddr+destAddr], intelBuf, il );
    }

    if ( len != _filelen )
        return 3;
    return 0;
} // flashIntelFile

```