FRONTGRADE

APPLICATION NOTE

UT32M0R50x

CAN Protocol for Updating Firmware in the UT32M0R50x NVM

12/20/2017
Version #: 1.0.3

Product Name	Manufacturer Part Number	SMD #	Device Type	Internal Pic Number
Arm Cortex M0+	UT32M0R50x	5962-17212	---	QS30

Table 1: Cross Reference of Applicable Products

1.0 Overview

This document details the CAN Update communications protocol between the UT32M0R500 microcontroller internal BootROM and a host system for uploading firmware to the NVM. This operation is only supported with BOOTCFG = 2'b11.

2.0 References

Refer to the 'C' header file can_update_protocol.h for the constants and data structures involved.

3.0 Protocol Basics

The CAN Update communications protocol is a message-based, master/slave (host/UT32M0R500) protocol. With a single exception, each message is 'atomic' and messages can be sent in any order. *[That said, there is a recommended sequence of messages, though that sequence is not enforced by the BootROM. Refer to section 6.0 for further details.]* The protocol behaves in a "half-duplex" fashion: commands are sent by the host and replies are sent by the BootROM. The host should wait for a reply to each command before sending a successive command. The standard-mode, 11bit CAN address of the UT32M0R500 BootROM is 0x555. The BootROM will reply to CAN address 0x000. The BootROM utilizes CAN0 for the update port.

All messages fit within the 8-byte CAN message payload. The CUP_CMD_ASCII_HEX_RECORD_COMPONENT message carries an ASCII HEX record component, with the entire record capable of spanning two or more messages, each message carrying a component of the record. This is explained further below.

Each message is comprised of a message header and a data block. The message header – defined by the structure CAN_MSG_HEADER – is common to all messages – commands and replies – and contains (a) the message type and

(b) the message sequence number. The message types are defined in can_update_protocol.h as CUP_MT_xyz defines. The sequence number is a (rolling) 8- bit value used to further associate replies with commands. The first message will be sequence '0' and the host must increment the sequence number with each successive message. *[There is a command to reset the sequence number to '0' should the host and UT32M0R500 ever lose 'sync'…]*

There are eight command messages, defined by the CUP_CMD_xyz data structures. Typically, the command messages are employed to effect actions/changes within the UT32M0R500 BootROM. However, some commands offer a 'Query' option, instructing the BootROM to report a setting pertinent to the command.

There is a single reply message – CUP_REPLY – with a status code and the option to return data in response to a 'Query' command. The status codes are defined in can_update_protocol.h as CUP_STAT_xyz defines. *[There may be several 'placeholder' status values that are unassigned and available for future use…]*

> **It's important to note the use of the *"pack (1)"* pragma. This pragma enforces proper byte alignment/packing of the message structures across platforms. Failure to use this pragma during the development of host software could result in a host system that's unable to properly communicate with the UT32M0R500 BootROM.**

4.0 Message Details – Commands

4.1 CUP_CMD_Device_To_Process

Message: CUP_CMD_DEVICE_TO_PROCESS

MessageType: CUP_MT_DEVICE_TO_PROCESS

This command is used to instruct the UT32M0R500 BootROM to associate any follow-on commands with the specified device: Nor Flash.

Field: Destination

Which device will be associated with any follow-on commands. Valid values are:

CUP_DEST_NOR_FLASH

Field: DisplayProgressToUART

A Boolean field that instructs the BootROM to display – or not – any status information to UART0. '1' is true, '0' is false. Default BootROM state is false.

Only the **Status** field of the reply will contain valid data.

4.2 CUP_CMD_IMAGE_TO_PROCESS

Message: CUP_CMD_IMAGE_TO_PROCESS

MessageType: CUP_MT_IMAGE_TO_PROCESS

This command is used to instruct the UT32M0R500 BootROM to associate any follow-on commands with the specified image.

Field: **Image**

Which image will be associated with any follow-on commands. Valid values are: 0..3 Only the Status field of the reply will contain valid data.

4.3 CUP_CMD_ERASE_IMAGE

Message: CUP_CMD_ERASE_IMAGE

MessageType: CUP_MT_ERASE_IMAGE

This command is used to instruct the UT32M0R500 BootROM to erase the (previously-assigned) image within the (previously-assigned) device.

Field: **Query**

A Boolean field that instructs the BootROM to query – or not – the validity of the image by performing a CRC check on the image. '1' indicates that a query is to be performed and that the erase is NOT to be performed. '0' indicates that the CRC check is NOT to be performed and that the erase is to be performed.

The `Status` field of the reply will contain valid data.

If a query is requested, the ReplyDataLSB field of the reply will indicate a '1' for a valid image, a '0' for an invalid image. If no query is requested both ReplyDataMSB/LSB fields are to be ignored by the host.

4.4 CUP_CMD_BEGIN_ACCEPTING_IMAGE

Message: CUP_CMD_BEGIN_ACCEPTING_IMAGE

MessageType: CUP_MT_BEGIN_ACCEPTING_IMAGE

This command is used to inform the UT32M0R500 BootROM that update (or compare) messages – containing ASCII HEX record data – are forthcoming.

Field: **Action**

Specifies if the subsequent messages are for update (programming) or for comparison. The valid values are:

CUP_ACTION_WRITE

CUP_ACTION_VERIFY

Field: **FileFormat**

Specifies if the record data will be in Intel HEX or Motorola S record format. The valid values are:

CUP_FORMAT_INTEL_HEX

Only the Status field of the reply will contain valid data.

4.5 CUP_CMD_ASCII_HEX_RECORD_COMPONENT

Message: CUP_CMD_ASCII_HEX_RECORD_COMPONENT

MessageType: CUP_MT_ASCII_HEX_RECORD_COMPONENT

This command is used to send the UT32M0R500 BootROM a 'component' of an ASCII HEX record. Since a single CAN message can only support an 8-byte payload, a single record may – and likely will – span two or more messages.

Field: **Data[]**

Contains a component of an ASCII HEX record. For example, an Intel HEX record of the format…

```
:10000000040130120B9010020C1010020C3010020DC<cr><lf>
```

Note: the carriage return and line feed will be 'hidden' in the record

…will be packed into **eight** CUP_CMD_ASCII_HEX_RECORD_COMPONENT messages, the Data[] fields containing hexadecimal values representing all the ASCII digits in the HEX record…

```
0x3A 31 30 30 30 30 /* :10000 */
0x30 30 30 34 30 31 /* 000401 */
0x33 30 31 32 30 42 /* 30120B */
0x39 30 31 30 30 32 /* 901002 */
0x30 43 31 30 31 30 /* 0C1010 */
0x30 32 30 43 33 30 /* 020C30 */
0x31 30 30 32 30 44 /* 10020D */
0x43 0D 0A 00 00 00      /* C<cr><lf><null><null><null> */
```

…each message containing a component of the record, sent separately. Note: the last message of the record is padded with NULL (0x00) values.

Only the **Status** field of the replies will contain valid data.

4.6 CUP_CMD_CRC_STAMP_IMAGE

Message: CUP_CMD_CRC_STAMP_IMAGE

MessageType: CUP_MT_CRC_STAMP_IMAGE

This command is used to send the UT32M0R500 BootROM a CRC16-CCITT value to be embedded into the image for future verification.

Field: **Query**

A Boolean field that instructs the BootROM to query – or not – the image's CRC. '1' indicates that a query is to be performed and that embedding the included CRC is NOT to be performed. '0' indicates that the CRC check is NOT to be performed and that embedding the included CRC is to be performed.

Field: **Calculate**

A Boolean field that instructs the BootROM ***– if a query has been requested –*** which CRC to return in the reply. '1' indicates that a calculated CRC is to be returned. '0' indicates that the embedded CRC is to be returned.

Fields: CRC16_CCITT_MSB and CRC16_CCITT_LSB

The most significant byte and least significant byte (respectively) of the CRC16-CCITT CRC to be embedded into the image. Ignored by the BootROM if Query is equal to '1'.

For example, if the image's CRC is 0x2C9F, the fields are assigned as follows:

CRC16_CCITT_MSB = 0x2C
CRC16_CCITT_LSB = 0x9F

The Status field of the reply will contain valid data.

If Query is '1', the ReplyDataMSB field of the reply will contain the most significant byte of the embedded *[or calculated, if so requested]* CRC and the ReplyDataLSB field of the reply will contain the least significant byte of the embedded *[or calculated, if so requested]* CRC. If Query is '0', both ReplyDataMSB/LSB fields are to be ignored by the host.

4.7 CUP_CMD_OVERRIDE_IMAGE

Message: CUP_CMD_OVERRIDE_IMAGE

MessageType: CUP_MT_OVERRIDE_IMAGE

This command is used to send the UT32M0R500 BootROM an "image override" value to be saved to NVM.

Field: Query

A Boolean field that instructs the BootROM to query – or not – the current override image number. '1' indicates that a query is to be performed and that the included override value is NOT to be saved. '0' indicates that the query is NOT to be performed and that the included override value is to be saved.

Field: **OverrideImage**

The image number to be assigned to the override value in NVM.

Note: this is a **signed** value. A value of '-1' *clears* the override and the normal prioritized image selection algorithm will be exercised at the next boot-up

The Status field of the reply will contain valid data.

If Query is '1', the ReplyDataLSB field of the reply will contain the current override value. If Query is '0', both ReplyDataMSB/LSB fields are to be ignored by the host.

4.8 CUP_CMD_RESET_SEQUENCE

Message: CUP_CMD_RESET_SEQUENCE

MessageType: CUP_MT_RESET_SEQUENCE

This command is used to instruct the UT32M0R500 BootROM to reset its internal sequence number to the included value.

Field: **NewSequenceNumber**

A value to which the host instructs the BootROM to reset its internal sequence number, typically '0'.

Note: the next command from the host must use this new value for the header's sequence number or else the BootROM will not complete the re-sync.

Field: HostReplyID_Valid

A Boolean by which the host instructs the BootROM to accept – or ignore – the accompanying HostReplyID_MSB/LSB values. If '0' (false), the BootROM must ignore HostReplyID_MSB/LSB, regardless of their values. If '1' (true), the BootROM must accept the HostReplyID_MSB/LSB values and start using them in replies ***immediately,*** including the reply to this command.

Fields: HostReplyID_MSB and HostReplyID_LSB

The CAN ID to which the BootROM is to send all successive replies. Bits 2..0 of HostReplyID_MSB become bits 10..8 of the reply ID. Bits 7..0 in HostReplyID_LSB become bits 7..0 of the reply ID. Bits 7..3 of HostReplyID_MSB are ignored.

Only the **Status** field of the reply will contain valid data.

5.0 Message Details – REPLY

5.1 CUP_CMD_REPLY

Message: CUP_CMD_REPLY

MessageType: CUP_MT_xyz

This is the reply structure used to reply to any of the above commands. In the MsgHeader, MessageType and SequenceNumber will mirror the values from the command that initiated the reply.

Field: **Status**

Any one of the **CUP_STAT_xyz** values:

CUP_STAT_ACK:	everything went 'OK' with the last message
CUP_STAT_MSG_TYPE_ERROR:	received message was NOT of type CUP_MT_xyz
CUP_STAT_SEQUENCE_ERROR:	got message 'N', expected 'M'
CUP_STAT_IMAGE_NUM_ERROR:	the image number is outside the range of 0..3
CUP_STAT_MSG_SIZE_ERROR:	the message was the wrong size
CUP_STAT_DATA_ERROR:	some value in the message was invalid
CUP_STAT_ACTION_ERROR:	the requested action resulted in some form of error

6.0 Protocol Notes

Recommended Sequence of Commands

While the BootROM does not expect/enforce a particular sequence of commands, the following sequence is recommended for normal update operations:

CUP_MT_DEVICE_TO_PROCESS
CUP_MT_IMAGE_TO_PROCESS
CUP_MT_ERASE_IMAGE CUP_MT_BEGIN_ACCEPTING_IMAGE
CUP_MT_ASCII_HEX_RECORD_COMPONENT *// repeat hundreds (thousands?) of times!*
CUP_MT_CRC_STAMP_IMAGE

Repeat the above sequence as many times as needed to upload the image(s). After which, if desired, an image override value may be assigned with:

CUP_MT_OVERRIDE_IMAGE

7.0 Appendix

"can_protocol_update.h" (reference only)

```
#ifndef uint8_t
#define uint8_t unsigned char
#endif

#ifndef int8_t
#define int8_t signed char
#endif

// MessageType(s), sorta' matches the CONSOLE_COMMAND enums

#define CUP_MT_DEVICE_TO_PROCESS          3
#define CUP_MT_IMAGE_TO_PROCESS           4
#define CUP_MT_ERASE_IMAGE 5
#define CUP_MT_BEGIN_ACCEPTING_IMAGE      6  // not 1-for-1 with CONSOLE_COMMAND
enum #define
CUP_MT_ASCII_HEX_RECORD_COMPONENT   7  // not 1-for-1 with CONSOLE_COMMAND enum
#define
CUP_MT_CRC_STAMP_IMAGE                    8
#define CUP_MT_OVERRIDE_IMAGE             9

#define CUP_MT_RESET_SEQUENCE  12

// Destination, matches the UPDATE_TO enums

#define CUP_DEST_NOR_FLASH                1

// Action, matches the UPDATE_ACTION enums
#define CUP_ACTION_WRITE                  1
#define CUP_ACTION_VERIFY                 2
// FileFormat, matches the FILE_FORMAT enums

#define CUP_FORMAT_INTEL_HEX              0

#define CUP_STAT_ACK                      0 // everything went 'OK' with the last
message #define
CUP_STAT_MSG_TYPE_ERROR                   1 // received message was NOT of type
CUP_MT_xyz
#define CUP_STAT_SEQUENCE_ERROR           2 // got message 'N', expected 'N+1', got
something else
#define CUP_STAT_IMAGE_NUM_ERROR          3 // the image number is outside the
range of valid values
#define CUP_STAT_MSG_SIZE_ERROR           4 // the message was the wrong size
#define
```

```
CUP_STAT_DATA_ERROR                  5 // some value in the message is invalid
#define CUP_STAT_ACTION_ERROR             6 // the requested action resulted in
some form of error
#define CUP_STAT_7                        7 //
#define CUP_STAT_8                        8 //
#define CUP_STAT_9                        9 //
#define CUP_STAT_10                       10  //
#define CUP_STAT_11                       11 //
#define CUP_STAT_12                       12  //

#define NUM_CAN_HEX_BYTES (CAN_BUF_SIZE - sizeof (CAN_MSG_HEADER))

#pragma push
#pragma pack (1)

typedef struct
{
uint8_t MessageType;                      // one of the CUP_MT_xyz defines above
uint8_t SequenceNumber;                   // rolling count 0..255, then start over
} CAN_MSG_HEADER;
// commands typedef struct
{
CAN_MSG_HEADER MsgHeader;
uint8_t Destination;                      // one of the CUP_DEST_xyz defines above
uint8_t DisplayProgressToUART;            // boolean, '1' = yes, '0' = no
} CUP_CMD_DEVICE_TO_PROCESS;

typedef struct
{
CAN_MSG_HEADER MsgHeader;
uint8_t Image;
} CUP_CMD_IMAGE_TO_PROCESS;

typedef struct
{
CAN_MSG_HEADER MsgHeader;
uint8_t Query;                            // '1' = query image for validity, '0' = erase
} CUP_CMD_ERASE_IMAGE;

typedef struct
{
CAN_MSG_HEADER MsgHeader;
uint8_t Action;                           // one of the CUP_ACTION_xyz defines above
uint8_t FileFormat;                       // one of the CUP_FORMAT_xyz defines above
} CUP_CMD_BEGIN_ACCEPTING_IMAGE;

typedef struct
{
CAN_MSG_HEADER MsgHeader; char Data[NUM_CAN_HEX_BYTES]; } CUP_CMD_ASCII_HEX_RECORD_COMPONENT;
 typedef struct
{
CAN_MSG_HEADER MsgHeader;
```

```
uint8_t Query;                          // '1' = query image's CRC, '0' = set embedded
CRC
uint8_t Calculate;                          // if Query == '1', then '1' ==
calculate image's CRC, '0' ==
                                    // use embedded CRC; ignored if Query == '0'
uint8_t CRC16_CCITT_MSB;                // ignored for query
uint8_t CRC16_CCITT_LSB;
} CUP_CMD_CRC_STAMP_IMAGE;

typedef struct
{
CAN_MSG_HEADER MsgHeader;
uint8_t Query;                          // '1' = query, '0' = set
uint8_t OverrideImage;                  // '-1' means erase/delete/cancel override;
ignored if Query
                                    // == '1'
} CUP_CMD_OVERRIDE_IMAGE;

typedef struct
{
CAN_MSG_HEADER MsgHeader;
} CUP_CMD_FORCE_LOAD_IMAGE;

typedef struct
{
CAN_MSG_HEADER MsgHeader;
} CUP_CMD_JUMP_TO_IMAGE;

typedef struct
{
CAN_MSG_HEADER MsgHeader;
uint8_t NewSequenceNumber;
uint8_t HostReplyID_Valid;                  // Boolean -- indicates if
HostReplyID_MSB/LSB contain valid
                                    // data
uint8_t HostReplyID_MSB;                // bits 10..8 of the host ID (bits 15..11 are
ignored)
uint8_t HostReplyID_LSB;                // bits 7..0 of the host ID
} CUP_CMD_RESET_SEQUENCE;

// replies typedef struct
{
CAN_MSG_HEADER MsgHeader;
uint8_t Status;                         // one of the CUP_STAT_xyz defines above
uint8_t ReplyDataMSB;                   // depends on command
uint8_t ReplyDataLSB;                   // single-byte replies have the result HERE!!
} CUP_REPLY;

#pragma pop
```

8.0 Revision History

Date	Revision #	Author	Change Description	Page #
8/16	Draft	SW	Initial Release	
8/16	1.1	SW	Added changes to SeqNumReset to support “Host ID Reply”	
2/17	1.2	SW	Corrected message count in CUP_CMD_ASCII_HEX_RECORD_COMPONENT	
12/17	1.3.0	AW	Reformat and added header sections	