Bootloader UART Protocol

BL6xx

Application Note *v1.3*

1 Module Bootloader Protocol

This document describes the protocol used to interact with the bootloader in a Laid BL6xx series module.

The flash-based module is delivered by Laird with bootloader firmware able to upload program application software into its flash memory.

Note: This protocol is derived from a bootloader protocol in an Atmel AT91 microcontroller. The content of this document was copied and modified from an Atmel document – *Flash Memory Upload Protocol* (July 2000).

2 Protocol Overview

This protocol is used for uploading content to the flash memory inside a module.

All multi-byte fields are little-endien.

3 Protocol Requirements

Full duplex communication between transmitter and receiver.

Asynchronous bi-directional full duplex serial link, 8 data bits, no parity, 1 stop bit at 115200 baudrate

Each transmitted data block contains a checksum, which enables the receiver to verify the integrity of the transmitted data.

- The transmitter calculates the checksum and inserts this at the end of the block,
- The receiver calculates the checksum of the received data and compares it to the received checksum.

Specific characters are defined by the protocol. This protocol is implemented by the following functions:

Character	Code	Signification	Source
<SYNC>	0x80	Synchronization	Host
<ATS>	‘b’	Acknowledge to synchronization	Module
<ACK>	‘a’	Acknowledge	Module /host
<NACK>	‘n’	Non acknowledge	Module /host
<ERASE>	‘e’	Erase	Host
<WRITE>	‘w’	Write	Host
<READ>	‘r’	Read	Host
<DATA>	‘d’	Data	Module /host
<VERIFY>	‘v’	Verify	Host
<ERROR>	‘f’	Error	Module
<PLATFORM>	‘p’	Platform	Host
<VERSION>	‘V’	Version	Host

Note: The code for the specific characters in the second column are the ASCII values for character so for example ‘b’ is 0x62 and ‘p’ is 0x70 etc.

4 Protocol Procedures

All procedures described in this section are Laird defined.

4.1 Synchronization and Platform Check Procedure

This procedure allows the bootloader to synchronize with the host and shall be the first packets a host sends to the bootloader in the sequence shown in the messages sequence diagram below.

The host system sends a synchronization command and the boot software shall send an <ATS> packet, following which <ACL>s are exchanged and then finally the <PLATFORM> packet is sent by the host and only an <ACK> response from the bootloader will allow further communication. If an <ERROR> is received from the module then it implies that the platform check failed and the firmware upgrade cannot proceed.

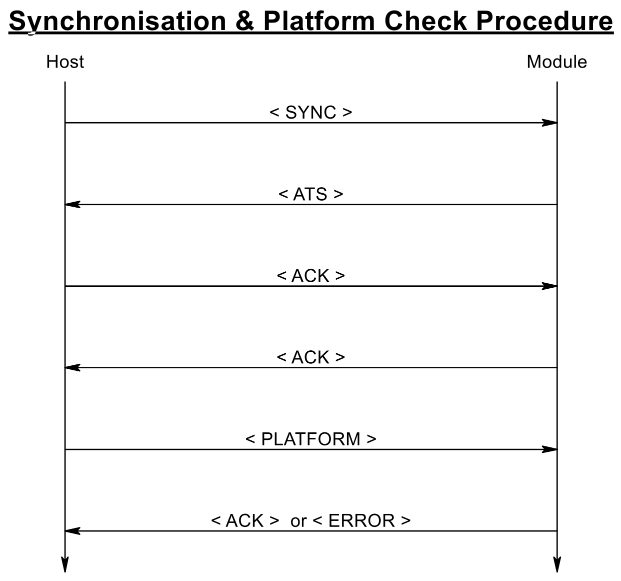

Figure 1: Synchronization and platform check procedure

4.2 Data programming Procedure

When the interface is synchronized the data programming can begin. This procedure requires several steps as the following message sequence diagram.

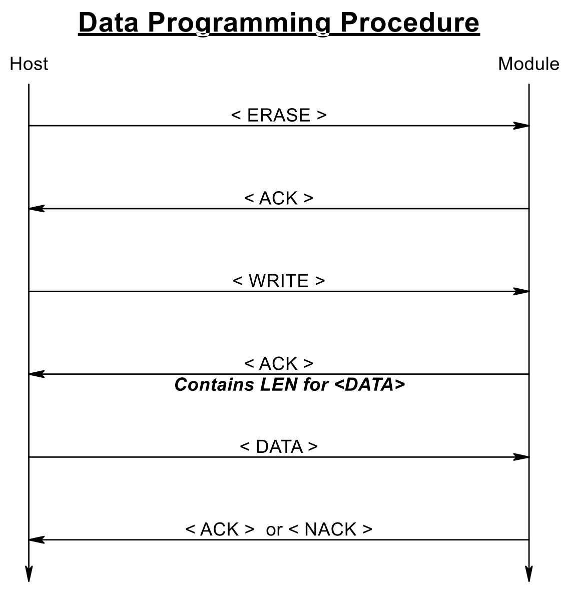

Figure 2: Data programming procedure

4.3 Verification Procedures

There are two ways for checking the Flash: the first is the use of the <VERIFY> command and the second is the use of the <READ> command.

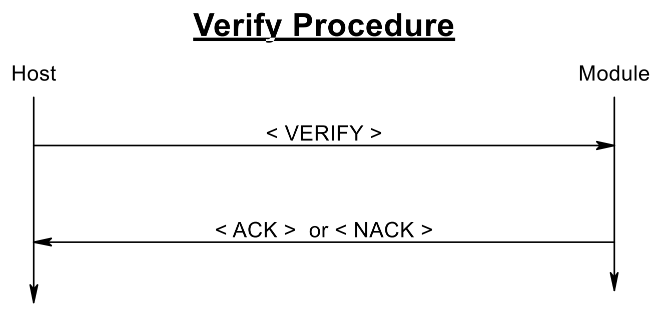

Figure 3: Verify procedure

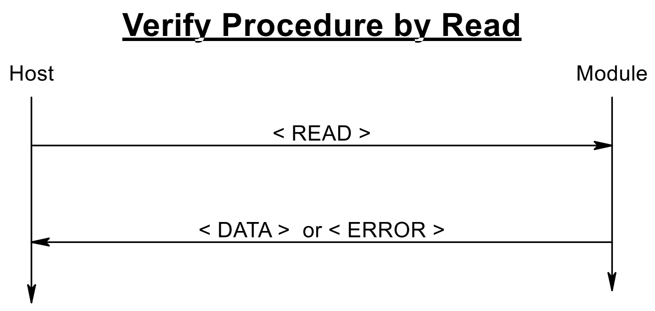

Figure 4: Verify procedure by read

5 List of Commands

The bootloader uses a binary protocol and each packet has a predetermined fixed length apart from the data packet identified by a ‘d’ character whose length will have been specified by a preceding ‘w’ or ‘r’ command packet in their one-byte LEN field.

5.1 Command Synchronization: SYNC

Sent by the host after the power up to synchronize the interface at the right baud rate.

Offset	Size (bytes)	Name	Value	Content
00h	1	<SYNC>	‘s’	Command code

5.2 Command Answer to Synchronization: ATS

Sent by the module after the synchronization procedure. The ATS frame consists of global boot software information. Only the Version field is relevant the rest shall be ignored as they are don’t care.

Offset	Size (bytes)	Name	Value	Content
00	1	<ATS>	‘b’	Command code
01	1	Chip	0x01	Ignore
02	2	Manufacturer	0x001F	Ignore
04	2	Flash	0x00C0	Ignore
06	1	Major Version	0xHH	Bootloader Major version
07	1	Minor Version	0xHH	Bootlaoder Minot version
08	2	CD	0x00	Ignore
10	4	ADDR	0x00001000	Ignore

5.3 Command Acknowledgement: ACK and NAK

Command acknowledgement consists of transmitting the <ACK> command code when the command has been executed.

For the <DATA> command, the acknowledgement is transmitted if the integrity of receive buffer is verified. This integrity is checked by comparing the calculated checksum to the receive checksum. Otherwise, the <NAK> command is transmitted.

Offset	Size (bytes)	Name	Value	Content
00	1	<ACK>	‘a’	Command code

Offset	Size (bytes)	Name	Value	Content
00	1	<NACK>	‘n’	Command code

5.4 Platform Check: PLATFORM

Sector erasing command consists of erasing a sector of the flash at the specific address.

Offset	Size (bytes)	Name	Value	Content
00	1	<PLATFORM>	‘p’	Command code
01	4	Platform ID		Unique ID

5.5 Version Info: VERSION

This will result in a six-character response consisting of “Vn.nnn” where n is a digit from 0 to 9

Offset	Size (bytes)	Name	Value	Content
00	1	<VERSION>	‘V’	Command code

5.6 Sector Erasing: ERASE

Sector erasing command consists of erasing a sector of the flash at the specific address.

Offset	Size (bytes)	Name	Value	Content
00	1	<ERASE>	‘e’	Command code
01	4	SECT		Address Sector erase

5.7 Data Writing: WRITE

This command consists of sending the write address to the boot loader.

Offset	Size (bytes)	Name	Value	Content
00	1	<WRITE>	‘w’	Command code
01	4	ADDR		Start address to write
05	1	LEN		Next Blocks Data length

5.8 Data Reading: READ

This command consists of sending the read address and the block length to the boot loader.

Offset	Size (bytes)	Name	Value	Content
00	1	<READ>	‘r’	Command code
01	4	ADDR		Start address to read
05	1	LEN		Next Blocks Data length

5.9 Data Block: DATA

This block is used to transfer data from host to MODULE and from MODULE to the host and immediately follows a WRITE or READ packet and as such the number of bytes in this packet is as per the LEN field in the preceding WRITE or READ packet.

Offset	Size (bytes)	Name	Value	Content
00	1	<DATA>	‘d’	Command code
01	LEN value	D0..Dn		Data LEN defined in R/W
02+LEN	1	1-byte checksum		This is the least significant byte of the same checksum calculated in the Verify packet described in section 5.9

5.10 Memory Verification: VERIFY

Memory verification consists of receiving a start address, a size and a checksum and comparing it to the checksum calculated in memory. The checksum is a 32-bit sum, which is calculated by adding all bytes from BASE to BASE+SIZE-1 of the memory space and is used as a check. Fail checksum verification causes <NACK>.

Offset	Size (bytes)	Name	Value	Content
00	1	<VERIFY>	'v'	Command code
01	4	ADDR		Start address to verify
05	4	LNG		Size of code to verify
09	4	CHK		Checksum See Section 6 for source code for the checksum calculation

5.11 Error notification: Error

Sent by the MODULE to notify an error.

Offset	Size (bytes)	Name	Value	Content
00	1	<ERROR>	'f'	Command code
01	1	Error		Error code

Error Code

Hex value	Content
1	Flash write error
2	Flash read error
3	Flash erase sector error
4	Unrecognized command

6 Checksum Algoritm

The Data Block and Memory Verification packets have a checksum component and the source code for that calculation is as follows:

```
unsigned long CalcChecksum(
    unsigned char  *pData,
    unsigned int  nBlockLenBytes
    )
{

    unsigned long nCheckSum=0;
    while(nBlockLenBytes--)
    {
        nCheckSum += *pData++;
    }
    return nCheckSum;
}
```

7 Revision History

Version	Date	Notes	Contributor	Approver
1.0	12 June 2018	Initial Release	Mahendra Tailor	Chris Cole
1.3	24 Mar 2020	▪ Added comment that multi-byte fields are little-endien. ▪ Corrected Typo in section 5.9 where the LEN field was incorrectly stated as starting on offset 2. It is at offset 1	Mahendra Tailor	Jonathan Kaye