20 November 2009

AVT-84x Auto Block Transmit (ABX)

The so-called "Auto Block Transmit" function (noted as "ABX") became available in firmware version 3.0 (and later).
This document provides a brief description of that function and the related commands.

Operational Description

The ABX function gives an AVT-84x unit the capability of transmitting a sequence of CAN frames containing a total of up to 32 KBytes (32 768 bytes) of data - without host computer intervention.

Note: This function is only available in CAN mode. However, it is available to either CAN0 or CAN4 channels (separately, simultaneous operations are not permitted).

The operator (with a host computer) sets up and stores CAN data and CAN transmit parameters into non-volatile memory of the AVT-84x unit. This is done using the commands described below.

Then, using one simple command (issued by the host computer or stored as an auto start command) the AVT-84x will begin transmitting the stored CAN data. The data is transmitted according to the stored CAN parameters. The AVT-84x unit composes CAN frames and transmits them, in sequence, until all of the data is transmitted (or the operation is terminated by the host computer).

Document Outline

Presented below are:

All commands involving non-volatile parameters are described first.

The control command (does not involve non-volatile parameters) is then described.

Lastly, a brief example is described.

Technical Support

Questions or Problems ??   Write or call for prompt support.

support@AVT-HQ.com   410-798-4038

Command Descriptions (non-volatile parameters)

The following commands involve querying for or storing various Auto Block Transmit (ABX) parameters. All of these parameters are stored in non-volatile memory of the AVT-84x.

All of these parameters should be initialized by the user prior to invoking the ABX function.

ABX operational parameters are stored in EEPROM space.

ABX data is stored in FLASH space.

ABX Separation Count 7x 36

The 7x 36 command queries for and sets the CAN frame 'separation count'. This is the time between CAN frames that the AVT-84x unit is transmitting.

This parameter sets a count value. The AVT-84x unit will wait by 'counting' the value specified by this command before transmitting the next CAN frame.

There are two 'sources' for this count: milliseconds or 'loops'.

Setting the separation count to be milliseconds results in the AVT-84x 'counting' the specified number of milliseconds between transmitted CAN frames.

A 'loop' is the time it takes the AVT-84x firmware to make a complete loop in the firmware, in CAN mode. This time is variable, but empirical measurements in a lightly loaded environment reveal the 'loop' time to be approximately 45 microseconds. The user should keep in mind that a full size 11-bit CAN frame at 500 kbaud occupies approximately 186 microseconds on the CAN bus.

71 36:	query for setting
74 36 0r xx yy:	command 0r = 01 = millisecond count 0r = 02 = 'loop' count xx yy = count value default: 84 36 02 00 0A ('loop' with count of 10 decimal ~ 450 microseconds between CAN frames)

ABX Message ID 7x 37

This command sets the transmit ID and related parameters for the transmitted CAN frame.

71 37:	query for setting
74 37 mm rr ss:	command mm: b7 = IDE = 0 = 11-bit ID b6 = RTR = should be set to 0, but user can define all other bits (b5 to b0) are 0 rr ss: 11-bit ID, right justified
76 37 mm rr ss tt vv:	command mm: b7 = IDE = 1 = 29-bit ID b6 = RTR = should be set to 0, but user can define all other bits (b5 to b0) are 0 rr ss tt vv: 29-bit ID, right justified

ABX Data 76 38

This command queries for or stores the CAN frame data. All of this data is stored in the AVT-84x unit in non-volatile FLASH space.

The address range is $0000 thru $7FFF (16 KBytes = 32 768 bytes).

When transmitting, all data is read and transmitted starting at address $0000.
The total number of data bytes transmitted is set by the 7x 39 command, described below.

Reading stored data is easy and very flexible.

- Data can be read starting at any address in the range and for any number of bytes specified (so long as the resulting address does not exceed the address range).

Storing or writing data has special rules.

- Data is stored in 'sectors' where a 'sector' is a maximum of 512 bytes.
- Data must be stored starting at a sector start address.
- A sector start address is on an even 512 byte boundary. In a 16-bit address, a sector start address has bits 8:0 all set to zero. In binary, a valid sector start address will be of the form:
    xxxx xxx0 0000 0000
  In hex, some example valid sector start addresses are (where 'x' is any value):
    $ x000
    $ x200
    $ x400
    $ x600
    $ x800
    $ xA00
    $ xC00
    $ xE00
- It is not necessary to completely fill a sector. However, it is not allowed to specify a start address in a sector that is not the sector start address.
- Any data not specified when storing a sector is automatically filled with $FF bytes.
- To completely fill the available 32 KBytes of ABX data would require the host computer to send 64 (decimal) commands to the AVT-84x unit.

Note that both the response to a query and the command have the unique format where the data does not contain a "header" byte. The data immediately follows the command (when storing) or the response (when making a query).

General format of the query and command.

```
76 38 0r ss tt kk ll
        0r = 01 = store data
        ss tt = sector start address
        kk ll = count of bytes to immediately follow (if a command)
                or count of bytes requested (if a query)
```

Examples

query for stored data; send this query:

76 38 02 ss tt kk ll

request to read stored data
start reading at address $ ss tt
read and send back to the host $ kk ll number of bytes

the AVT-84x will respond with:

86 38 02 ss tt kk ll   ....   the requested number of bytes will immediately follow.

store 512 bytes of data, or fewer; send this command:

76 38 01 ss 00 kk ll   ......   the data must follow immediately

command to store data
start storing the data at address $ ss 00
(which must be a valid sector start address)
$ kk ll specifies how many bytes the host computer
will send immediately following the command
($ kk ll   valid range is $0001 to $0200)

After the AVT-84x receives the command and all expected data bytes, the AVT-84x will respond with:

86 38 01 ss tt kk ll

ABX Byte Count   7x 39

This parameter is the count of bytes the ABX function is going to read from non-volatile memory, fill CAN frames, and transmit.

Do not confuse this with any other 'count' in any other command.

This is the count of the total number of data bytes that the ABX function will transmit in sequential CAN frames.

71 39:	query for stored value
73 39 xx yy:	store the data count xx yy = number of bytes to transmit, total

Command Description (control)

There is only one command needed to commence an Auto Block Transmit operation. Described below. (Once started, the function will run to completion unless the user issues a disable command.)

Remember that all parameters and data are stored in non-volatile memory on the AVT-84x unit. Thus, once an AVT-84x unit has been initialized, the user can start an Auto Block Transmit operation at any time. There is no need to re-initialize the ABX parameters just because the AVT-84x unit has been reset or power cycled. Note that all of this can be automated using stored "Auto Start Commands".

ABX Control 7x 3A

Prior to issuing this command the user must have initialized all ABX operation parameters using the previously described commands.

(Note that the function can be invoked without initializing the stored parameters - it would work and likely be rather ugly.)

71 3A:	query for operational status, both CAN channels
72 3A 0x:	query for operational status for CAN channel ‘x’
73 3A 0x 0y:	set CAN channel ‘x’ operation to value ‘y’

Examples

query for status of both CAN channels; send this query:

```
71 3A
```

receive these responses

```
83 3A 00 00        CAN0 is disabled
83 3A 04 00        CAN4 is disabled
```

query for status of CAN4; send this query:

```
72 3A 04
```

receive this response:

```
83 3A 04 00        CAN4 is disabled
```

enable the ABX function for channel CAN0; send this command:

```
73 3A 00 01
```

receive this response:

```
83 3A 00 01        CAN0 is enabled
```

Note: The ABX function using CAN0 will begin immediately.

when the transfer completes, receive this response:

```
83 3A 00 00        CAN0 is disabled
```

to disable an ABX operation that is in progress on CAN0; issue this command:

```
73 3A 00 00
```

Note: The ABX function will be halted immediately.
It can be restarted, but it can not be resumed.

Operational Example

I want to set the following ABX parameters:

set CAN ID = 03 57   (11-bit, no RTR)
separation time to 20 milliseconds
store some data
total data count to 100 bytes

Then I will set the AVT-84x unit, as noted here, and transmit the block.

CAN0 at 500 kbaud
do not receive any CAN frames
enable CAN0 for operations.

Communications from host computer to AVT-84x unit

```
;  reset the unit
F1 A5

;  enter CAN mode
E1 99

;  set ABX CAN ID = 03 57,  11-bit,  RTR = 0
74 37 00 03 57

;  set ABX separation to milliseconds and a count of 20 (decimal)
74 36 01 00 14

;  store some ABX data, start address = $0000,  byte count to store = $0020
76 38 01 00 00 00 20
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F

;  note that the rest of the data, by default, = $FF

;  set ABX transmit count to 100 bytes
73 39 00 64

;  at this point all ABX parameters are defined and stored


;  set CAN0 baud rate to 500 kbaud
73 0A 00 02

;  will not define any acceptance ID masks

;  will not define any acceptance IDs

;  enable CAN0 for operations
73 11 00 01


;  invoke the ABX function,  using channel CAN0
73 3A 00 01
```