<dalf1_api.doc>

DALF – 1; Rev F Motor Control Board

API Interface

Revision 0.13
June 28, 2007

Table of Contents

Warranty

The software libraries and tools are provided "as is" without warranty. The entire risk for the results and performance of these libraries and tools is assumed by the purchaser. Embedded Electronics LLC does not warrant, guarantee or make any representation regarding the use of this product. No other warranties are made, expressly or implied, including, but not limited to, the implied warranties of merchantability and suitability of products for a particular purpose. In no event will Embedded Electronics be held liable for additional damages, including lost profits, lost savings or other incidental or consequential damages arising from the use or inability to use Embedded Electronics LLC products.

Disclaimers

Embedded Electronics LLC reserves the right to make changes without notice to this product. Changes made to improve reliability, performance, capabilities, design or ease of use, or to reduce size or cost could effect documentation, hardware, and firmware. Any Embedded Electronics LLC product may not be used as components in life support devices of any description.

Copyright

Software License

The <main.c> and <dalf.lib> files provided on the CD for the purpose of encouraging additional software development are subject to a license agreement explained in the End User License Agreement (EULA). The EULA file is included on the CD that ships with the product and may also be read on the EE Website **http://www.embeddedelectronics.net/**

INTRODUCTION

This document describes the RS232 serial communication interface between the Dalf1 Motor Control Board(s) and a "smart" PC Application. The PC Application issues commands to the board(s) and receives data and status from the board(s) in "Message Packets".

Updated documentation is maintained at **http://www.embeddedelectronics.net/**

The Dalf-1F board hardware does not support networking of multiple boards on the RS232 Interface, but later board revisions may. This API design accommodates that possibility by including a Network ID field (NID) in message packets to identify the target device.

SETTING THE SERIAL PORT OPERATING MODE:

The Dalf Board firmware supports two serial port interfaces. One of these is a **Terminal Emulator Command Interface (TE)** for use with common terminal emulator applications such as Hyperterm or TeraTerm. This document describes the **Dalf Applications Programming Interface (API)** which has similar capabilities to TE. The API uses the same hardware resources as the TE Interface, but is better suited for communication with a smart PC Application like a Windows GUI. The command sets are similar, but the communication protocol is quite different. **The board will always power up in TE mode.** In order to switch from the TE Interface to the API it is necessary for the application to send a special 2-character sequence over the RS232 link to the Dalf Board:

0x1B 0x31 (ESC "1") - Sets interface to TE Mode
0x1B 0x32 (ESC "2") - Sets interface to API Mode

On receipt of the request, the board will quietly change to the specified serial port operating mode. There is no acknowledgement. In networking applications, all networked boards will change operating mode.

Serial Networking

To accommodate a networked configuration, each board maintains a single, unique, Network ID (NID) byte to distinguish it in the network. The NID is transmitted in each message packet to indicate the target for the message packet and only the addressed board responds. The factory default value for each board is NID=1 (must be changed for network operation - see owners manual).

NID	Packed Destination
0	PC Application
1-254	Individual Dalf1 Boards
255	Network Broadcast (eg; RESET)

PHYSICAL LAYER

- **Connections:** Primary serial port connector on the Dalf1 board, standard serial cable, COM port on the PC.
- **Protocol:** RS232 (level shifted), asynchronous, full duplex, 8N1, Baud Rate 19,200.
- **Tristate:** In a network configuration, each board must keep the serial port transmit line tri-stated when not in use. This avoids bus conflicts with multiple boards driving this line.

PACKET LAYER

The PC Host issues command messages and receives response messages from a particular board using the message packet format described below. A Packet consists of (N+6) bytes where N is the number of bytes in the DATA Field. The NID field indicates the target for the message. The DATA field is used to supply necessary command arguments and return data and status to the PC Host. The content and length of the DATA field is command dependent. Some commands support default arguments, hence variable values for N. During command dispatching, the command is identified by the CMD and N bytes.

Commands to a board which result in the board returning data or status in a response packet will have the PC Host identified (NID=0) as the message target. Some commands will have not data (eg; RESET) and will have N=0. The response packet to a command that reads and returns a block of memory could have N as large as 128, so the maximum packet size is 134 (N+6).

Message Packet Format

FIELD	BYTES	FORMAT	DESCRIPTION	
STX	1	HEX	**STX=0x02**;	Start of packet
NID	1	HEX	Network ID;	0-255; PC=0, Network=255
CMD	1	HEX	Command Char;	"A"-"Z"
N	1	HEX	Msg body length;	0<=N<=128
DATA	N	HEX	Msg body;	Command Arguments and Response data
CHKSUM	1	HEX	Check Sum;	Zero sum on N+6 byte packet
ETX	1	HEX	**ETX=0x03**;	End of packet

The Dalf Board responds to each **command packet** with an **ACK (0xAA)** or an **ErrCode** byte to inform the host of success (ACK) or failure of the command. Protocol violations and other errors (See ERROR LIST) will result in an error code being transmitted by the board and the command will not have been executed.

When the PC Host receives a **response packet** from the board, no acknowledgement is expected by the board. Normally, any characters transmitted outside the scope of the [STX … ETX] packets to the board are treated as errors. The one important exception to this rule is the serial port mode change discussed previously.

TRANSACTION LAYER

ERRORS DETECTED BY THE BOARD
The board firmware can detect many types of errors in a message received from the PC host - see the ERROR LIST. Any error causes the board to record the error code, wait for a period of bus idle, flush the receive buffer (current message), and transmit the ErrCode byte. In addition, it is possible to read the last ErrCode byte unless it has been overwritten by a subsequent error.

After a command packet is transmitted by the PC Application, there are 3 possible events and it is important for the Application to wait for one of these before issuing the next command.

1. Receipt of ACK and any data or status response (NO ERROR).
2. Receipt of ErrCode (ERROR).
3. Timeout while waiting for an expected response (ERROR).

[1] BOARD SENDS ACK (NO COMMAND ERROR)
==========
When the addressed board has found no errors in a command packet and has successfully commenced command execution, the board will respond with an **ACK** chr. If the response to a particular command calls for the board to send data or status to the host, the PC application must wait for the data response packet (or timeout) before sending the next command. This avoids potential bus conflicts in a networked configuration where more than one board could be trying to send responses to the host simultaneously.

HOST RECEIVES ACK
==========
When the host PC receives an ACK from a board it indicates that the board has successfully received the packet, verified the data, and started command execution. **Depending on the command it does not necessarily signal command completion.** If the command response calls for the board to send data or status to the host, the host must wait for delivery (or timeout) before issuing the next command.

[2] BOARD SENDS ERRCODE (COMMAND ERROR)
==========
When the addressed board encounters an error during receipt or processing of a command packet the board will respond as follows:

- The error is recorded in **ErrCode**.
- Serial interrupts are disabled (no receive communication).
- The code waits for bus idle + 5 msec.
- The input (receive) buffer is flushed.
- Serial interrupts are enabled.
- The ErrCode byte is transmitted to the host.

HOST RECEIVES ERRCODE
==========
When the host PC receives an error code (not the expected ACK) from a board it indicates that the board has detected an error condition. The CMD associated with the message will not have been executed. The host should:

- Wait for a brief period with no transmissions (minimum 5 msec)
- Take any remedial action (eg; correction, retry).

[3] PC APPLICATION TIMES OUT

The timing value for the **PC Application timeout will be determined by the application**. As a starting point, 200 msec should be more than adequate (None of the commands in the Dalf Command Set take significant time to execute). Waiting at least the length of time that is used for the Dalf Board Timeout (see RX1TO) should also guarantee that the Dalf Interface will be functional.

Other Details

If transmission from the board to the PC was unsuccessful, it is up to the PC Host to take whatever corrective action (eg; retry) is necessary.

When the board does not get timely completion of a message packet (ETX), it is a timeout error condition that is detected by the board. If the timeout is detected by the board before it can determine the message destination (NID hasn't arrived), no error condition will be signaled to the PC Host. The PC can detect this condition by its own timeout while waiting for message acknowledgement. If the board times out after it has determined that it is the target for the message (NID has arrived) it is treated just like all other errors and an error code will be transmitted by the board.

If a message from the PC Host is a network broadcast (NID=255), no response will be sent from the board.

The board timeout period is controlled by the value **RX1TO** in the Parameter Block. The default is 200 msec which should generally be plenty, but this can be customized.

Board Responder Rules:

- A board **will not** initiate message transactions (either with the PC Host or another network board).
- A board **will not** respond unless it can successfully determine that it is the target for a message from the host. If an error occurs that makes it impossible for the board to determine the NID of the packet, no response (including ACK or ErrCode) will be issued. The PC Host can detect this condition by timeout.
- An addressed board **will** respond with an ACK or ErrCode within the specified timeout period after successful command message receipt.

PC Host Rules:

- Host **should** wait at least RX1TO msec before repeating a message which has timed out.
- Host **should** wait a reasonable period between sending successive commands. While not a **must**, this allows sufficient execution time for other board processes to proceed without frequent serial port interrupts. The determination of "reasonable" depends on board setup (eg; baud rate) and activity, but as a general guideline, perhaps a minimum of 25 msec will be appropriate.
- Host **must** wait for a message transaction to be completed (including any data or status response) with a successful response or an error indication, before issuing a subsequent message to any board.

TRANSPORT LAYER

Command Definitions (1<=NID<=255; from PC Host):

CMD	N	DATA	DESCRIPTION
A	1	fPWM (0x00 - 0x18)	Set PWM Frequency by table index
B	2	Fan#(1,2), ON/OFF#(0,1)	Fan On/Off control
C	1	Adc Channel (0-6)	Get A/D Reading
C	0		Get A/D Reading (all)
D	3	HH(0-23), MM(0-60), SS(0-60)	Set RTC
D	0		Get RTC
E	1	Mtr#	Get Mtr Position
E	0		Get Mtr Position (both)
F	4	Mtr#, Encoder[0 1 2]	Set Encoder
F	1	Mtr#	Set Encoder to Zero
I	0		Reset
J	3	IOEXP#(1,2), Reg#, byte	IOEXP Write byte
K	2	IOEXP#(1,2), Reg#	IOEXP Read byte
L	4	MemType, AdrsLo, AdrsHi, BlkLen	Memory Block Read (3 mem types)
M	3	PotDevice#(1,2), Reg#, byte	Digital Pot - Write Register
N	1	Channel# (0x01 - 0x03)	Get specified R/C pulse width
N	0		Get All R/C pulse widths
O	1	Mtr#	Stop specified motor
O	0		Stop both motors
P	7	Mtr#, Kp[0 1], Ki[0 1], Kd[0 1]	Set PID parms Kp, Ki, Kd
P	1	Mtr#	Get PID settings
Q	6	Mtr#, Tgt[0 1 2], Limit[0 1]	PID Step Response
Q	4	Mtr#, Tgt[0 1 2]	PID .. .. ; Default Limit
R	3	MemType, AdrsLo, AdrsHi	Read Memory Byte (3 memory types)
S	6	Mtr#, Dir, Vm[0 1], Acc[0 1]	Move; ConstantV, Closed Loop
S	4	Mtr#, Dir, Vm[0 1]	Move; .. .. Default Acc
S	2	Mtr#, Dir	Move; .. .. Default Vm, Acc
T	1	Mtr#	Trigger move (closed loop)
T	0		Trigger move (closed loop; both)
U	1	Mtr#	Get mtr status
U	0		Get mtr status (both)
V	1	Mtr#	Get mtr velocity
V	0		Get mtr velocity (both)
W	4	MemType, AdrsLo, AdrsHi, byte	Write Memory Byte (3 memory types)
X	4	Mtr#, DIR, Speed, tSlew	Move (Open Loop)
X	3	Mtr#, DIR, Speed	Move .. .. Default tSlew
Y	8	Mtr#, Tgt[0 1 2], Vm[0 1], Acc[0 1]	Move (Closed Loop)
Y	6	Mtr#, Tgt[0 1 2], Vm[0 1]	Move .. .. Default Acc
Y	4	Mtr#, Tgt[0 1 2]	Move .. .. Default Vm, Acc
Z	0		Upload parms to EEPROM

Notes:
1) Mtr#: (1,2)
2) PotDevice#: (1,2); Note 4 pots, 2 per device.
3) Tgt: 24-bit, signed, little endian.

4) MemType: 1=RAM, 2=Ext EEPROM (24LC512), 3=Int EEPROM
5) BlkLen: Block Size: 1 <= BlkLen <= 128.
6) Kp,Ki,Kd: 16-bit, unsigned, little endian.
7) Dir: 0=Fwd, 1=Rev
8) Speed: [0 .. 100] Duty Cycle Percentage
9) tSlew (msec): Ramp rate: 1% Duty Cycle every tSlew msec
8) Vm: (ticks/Vsp): Midcourse velocity*256; 16-bit unsigned.
9) Acc: (ticks/Vsp^2: Acceleration*256; 16-bit unsigned.

Response Definitions (NID=0; from Dalf1 Board):

CMD	N	DATA	DESCRIPTION
C	1	ADC0[i]	Specified A/D Reading
C	7	ADC0[0]..ADC0[6]	All 7 A/D Readings
D	6	HOURS[0 1], MINS[0 1], SECS[0 1]	RTC (Hex)
E	6	E1[0 1 2], E2[0 1 2]	Encoder positions
E	3	Ex[0 1 2]	Encoder position; x=1,2
K	1	Byte	Byte read from IOEXP
L	n	Byte[0 1 .. n-1]; n=BlkLen<=128	Memory block bytes
N	2	PulseWidth[0 1]	R/C pulse width (uS)
N	6	PW1[0 1], PW2[0 1], PW3[0 1]	R/C pulse widths (uS)
P	13	KP[0 1], KI[0 1], KD[0 1], VSP, VMIN, VMAX, MAXERR[0 1], MAXSUM[0 1]	PID settings et. al
Q	24	PID Err[0 1 2] (times 8)	PID Step Response; 8 values of Err
R	1	Byte	Byte read from memory
U	6	MtrStatusx	MtrStatusx[0..5]; x=1,2
U	12	MtrStatus1[0..5], MtrStatus2[0..5]	Both mtr status
V	3	Vx[0 1 2]	Mtr velocity (ticks/VSP)
V	6	V1[0 1 2], V2[0 1 2]	Mtr velocities (ticks/VSP)

Notes:
1) All multi-byte arguments and returned data are in little endian (low order byte first) format. This is an API vs. TE difference.

2) MtrStatusx: MTRx_MODE1, MTRx_MODE2, MTRx_MODE3, Powerx, Mtrx_Flags1, Mtrx_Flags2; x=1,2. (See owner's manual for a description of these bytes).

3) **Cmd Q** is special. Unlike other commands, Cmd Q can elicit a multi-packet response from the board. Beginning with firmware version 1.60, 8 PID Err values (3 byte, 2's complement integer format for each value) are returned in each packet for communication efficiency. The response packets are spaced sequentially in time with an inter-packet gap of 8*VSP msec (see Dalf Owner's Manual). All data should be received before issuing another command. The number of packets returned is **Limit/8** (rounded up to the nearest integer) where "**Limit**" is the argument to Cmd Q. In the event that the last packet is a "partial" packet, the remaining Err values in that packet will be zero.

CMD INTERFACE TIMEOUT FEATURE

Beginning with Firmware Version 1.62 there is support for a timeout feature that detects and responds to a lack of command activity on any of the 3 serial command interfaces (TE, API, I2C2). The feature is primarily intended for use with a programmable interface so it is more likely to be of interest for users of the API or I2C2 interfaces. When enabled, the feature may be used to automatically shutdown motors when a valid command has not been received on the command interface for a duration of time that is adjustable.

One application is its use as a safety feature analogous to the "signal loss" detection and response in radio controlled systems. As long as you keep periodically sending commands at a repeat rate faster than the timeout period, the motors will continue whatever action they have been commanded to perform. If, for some reason, the communication channel is broken, the motors will timeout and stop themselves. Notice that any valid command resets the timeout, not just motor movement commands.

The timeout period in milliseconds is the product of two byte variables **CMDSP** and **CMDTIME** stored in the Parameter Block. Change CMDSP and CMDTIME to set your desired timeout. The units for CMDSP are msec, and the units for CMDTIME are CMDSP msec. For example, if CMDSP = 20 and CMDTIME = 255, then the timeout period is 5,100 msec or just a bit over 5 seconds. The maximum timeout period is 255*255 = 65,025 msec or about 65 seconds. To enable the feature there is a bit "**cmdto**" in the **SYSMODE** variable in the Parameter Block. By default, the feature is not enabled (cmdto='0'). To enable the feature, simply set the bit. You may make changes to the timeout or the enable status of the feature at runtime by altering the ERAM versions of these variables. See the Owner's Manual for details about the location of these variables in the Parameter Block and ERAM.

Under the hood:
A countdown timer ***Cmdcount*** is decremented by a 1 msec interrupt service. When Cmdcount becomes zero, it is re-initialized with the CMDSP value and a service request is generated to be handled within the main processing loop. The main processing loop responds by decrementing a byte counter ***CmdTicks***. Assuming the timeout feature has been enabled ("cmdto"=1) and *CmdTicks* has become zero, it is re-initialized with the CMDTIME value and one or more of the motors may be shutdown. If instead, during this timing process a valid command has been received, the value of *CmdTicks* will have been re-initialized with the CMDTIME value, thus avoiding timeout.

There are a couple of things to note here:

- CMDSP controls the timing resolution. For example; if CMDSP = 0x14 = 20 msec, then the main loop will check for timeout every 20 msec. Your actual timeout response could vary by as much as 20 msec from what you have set up with your choice of CMDSP and CMDTIME.

- Making CMDSP small generates additional overhead in the main loop. If CMDSP = 0x01 for example then the main loop gets a service request every 1 msec - probably undesirable.

- If a motor is being controlled thru one of the Pot or R/C interfaces it will be unaffected by this serial timeout feature - even if the feature is enabled.

ERROR LIST

The table below shows the list of potential errors associated with the API. The host application can query and reset the ErrCode variable using the Read and Write Memory commands. See the "Fixed Address RAM Values" section in the Dalf Owner's Manual for the memory location of the ErrCode variable.

ERROR LIST

Err Code	Error Description ---------- Example
0x00	No Error
0x01	Parse ---------- Unexpected chr in command packet
0x02	Number of arguments----- [CMD,N] not found for command packet
0x03	Parameter (bad value)----- Mtr#=5
0x04	Mode ---------- Serial port move cmd, but R/C mode
0x05	Rx1 Framing (hdwr)------- Hardware framing error (baud?)
0x06	Rx1 Overrun (hdwr)------- Hardware overrun
0x07	Buffer Overrun (sftwr)----- Software receive buffer overrun
0x08	Protocol ---------- Expected ETX not received.
0x09	ChkSum ---------- Mismatch on computed packet checksum
0x0A	Timeout ---------- Timeout waiting on expected packet completion
0x0B	Disabled ---------- Mtr control interface disabled (eg; over current)
0x0C	
0xAA	**ACK (no error)** ---------- Successful command receipt and start of processing

In some cases, multiple error conditions map to the same error codes. For example, there are many potential protocol violations.