DALF – 1; Rev F   Motor Control Board

I2C2 Interface

Revision 0.05
June 28, 2007

Table of Contents

WARRANTY 2
INTRODUCTION 3
PHYSICAL LAYER 3
NETWORKING 4
PACKET LAYER 4
TRANSACTION LAYER 5
ERROR HANDLING 6
TRANSPORT LAYER 7
TIPS FOR DEVELOPING AN I2C2 MASTER 9
CMD INTERFACE TIMEOUT FEATURE 10
ERROR LIST 11

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

Copyright 2006 Embedded Electronics, LLC.

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 Command/Monitor MASTER/SLAVE I2C Communication Interface using the Dalf1 **Secondary I2C Bus**.

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

The I2C bus is a standard designed by Phillips and consists of two signals {SCL, SDA} plus GND. The Dalf Board brings out the secondary I2C bus signals to screw terminals labeled {SC2, SD2}. In addition, convenient pin-outs for a 4 pin header {SC2, SD2, VDD, GND} located near the screw terminals facilitates daisy chaining the bus for networking - see schematic. The firmware configures the Secondary I2C Bus (**I2C2**) as an I2C SLAVE in order to accept commands and return data to an off board MASTER that hosts this bus.

The Dalf Board firmware supports two separate I2C Bus Modules. The **Primary I2C Bus** is configured as a MASTER to allow the board firmware to access the on-board I2C Peripherals. The primary I2C Bus signals are also routed to screw terminals {SC1, SD1}. Do not confuse this connector with the one mentioned above.

The Dalf Board firmware also supports two other serial Command/Monitor Interfaces which utilize the RS232 bus and the DSUB9 connector. The Terminal Emulator **(TE)** Interface is described in the Dalf Owner's Manual. The Application Programming Interface **(API)** is described in the Dalf API Interface Document. The TE, API, and I2C2 Interfaces are quite similar in terms of functionality, but have different communication protocols. This document contains a very terse tabular format description of the commands and expected responses.

This document is dedicated to describing the communication protocol for the I2C2 Interface only. For command content and function, refer to the description of the commands in the TE section of the Dalf Owner's Manual.

PHYSICAL LAYER

- **Connections:** Two wire plus GND: {SC2, SD2, GND} screw terminal connections on the Dalf1 Board. A four terminal header (SC2, GND, VDD, SD2) can be mounted on the provided pads for easy off board daisy-chaining of the bus.
- **Configuration:** The Dalf Board is configured as an I2C SLAVE on the I2C2 Bus.
- **Addressing:** The address of the Dalf Board is configurable **(default: 0x60)**. The address is maintained in the **Dev_DALF** parameter in the non-volatile Parameter Block section of the EEPROM - see Owner's Manual.
- **Hardware Protocol:** I2C @ 400 KHz.
- **PullUps:** The Dalf Board supplies 2.7K pull-up resistors on the SC2 and SD2 lines to Vdd. If the MASTER also supplies pull-ups, one set should be removed.

NETWORKING

The I2C2 Interface supports networking of multiple Dalf1 boards. This feature arises naturally as part of the nature of the I2C Bus Specification in which device access is specified in an 8-bit "address". Actually only the upper 7 bits of the "address" constitute the device address, while Bit0, the R/W# bit, functions as a control bit. The Dalf Board powers up and configures itself as an I2C SLAVE Device on the secondary I2C Bus with address = Dev_DALF.

To configure multiple Dalf Boards as SLAVES:

1) **Pull-Ups:** There should be exactly one set of pull-up resistors on the Secondary I2C Bus. It will be necessary to remove all but one set of these pull-ups on networked Dalf boards (don't forget to check the MASTER).

2) **Addressing**: Use the TE interface (for example) to record unique values for the Dev_DALF parameter for each Dalf Board (eg; 0x60, 0x62, 0x64, ..) in the Parameter Block.

3) **Connections**: Near the screw terminals for the I2C2 connections is a spot for a header that can be used to easily daisy chain the bus.

PACKET LAYER

The MASTER Device issues Command Packets and receives board Response Packets from the SLAVE Dalf Board(s) using the message packet format described below. A Packet consists of (N+3) bytes where N is the number of bytes in the DATA Field. All multi-byte values are transmitted in "little endian" format in which the low order byte appears first. The maximum packet size occurs as the response to the L Command which can deliver up to 128 bytes of data, so the maximum packet size is N+3=131.

For **Command Packets** from the MASTER, the DATA field is used to supply command arguments. The content and length of the DATA field is command dependent. Some commands require no arguments hence have N=0 (Pkt size 3).

Every command from the MASTER elicits a response from the Dalf SLAVE board which consists of either a single byte "Success/Fail" code, or in the case of successful execution of a command which returns data, a **Response Packet** in the same format as that of the command. Response Packets use the DATA field to supply command results and status according to the specific command (See the TE section in the Owner's Manual). As with Command Packets, the value of N (hence the size of the packet) is command dependent. The CMD field echoes the CMD chr transmitted in the Command Packet unless there has been an error (in which case the first and only char will be the Success/Fail code).

A complete transaction between Master and Slave consists of the Master writing a command packet to the Slave followed immediately by a Master Read from the Slave (packet or Success/Fail code).

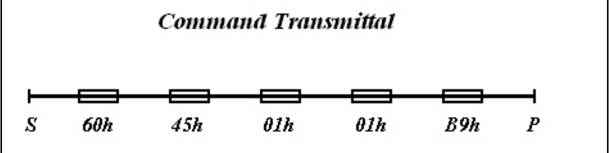
Message Packet Format

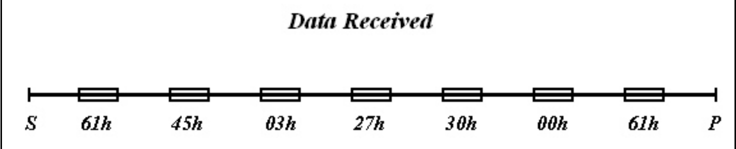
FIELD	BYTES	FORMAT	DESCRIPTION
CMD	1	HEX	Command Char; "A"-"Z"
N	1	HEX	Msg body length; 0<=N<=128
DATA	N	HEX	Msg body; Command Arguments and Response data
CHKSUM	1	HEX	Check Sum; Zero sum on N+3 byte packet

TRANSACTION LAYER

This document is not the place to discuss I2C protocol or hardware standards. Refer to the I2C Specification from Phillips if you are unfamiliar with the I2C Bus Standard. Here is an abbreviated example of a typical Dalf Board transaction using the I2C2 Interface. It does not show all of the '1's and '0's and does not show the clock pulses, but should provide enough detail to be useful:

Example: I2C2 Bus Master Transaction to obtain Motor #1 Position

 Command Transmittal S 60h 45h 01h 01h B9h P	Here the MASTER has transmits the START bit (S), the Device Address with **WRITE** control (60h), the CMD ("E"=45h), the data length (N=01h), the data (Mtr# =01h), the CKSUM byte (B9h) and the STOP bit (P). The SLAVE will have acknowledged each byte on the SCLK pulse following the byte.

 Data Received S 61h 45h 03h 27h 30h 00h 61h P	MASTER transmits START, Device Address with **READ** control (61h), and receives from the SLAVE: CMD=45h, N=03h, Mtr Position=003027h, and CKSUM=61h. The MASTER asserts STOP. For each byte received (except the last), the MASTER acknowledges receipt on the next SCLK pulse. **The last byte (CKSUM) must be "negatively acknowledged"** to let the SLAVE know that transmission is ending.

The Dalf SLAVE employs an interrupt driven receive interface using the PIC MSSP2 Module. If the MASTER erroneously requests data (MASTER READ) that the SLAVE doesn't have (eg; a Master Read Operation without a preceding Master Write Operation), the result will be that the slave holds the SCLK line low (clock stretching) for a period of time until timeout. After timeout, the SLAVE will load the output buffer SSP2BUF with the timeout error code and release the SCLK line, delivering the error byte to the MASTER. In this way the bus remains functional, and the MASTER can detect the timeout condition (bogus CMD byte). The period for the timeout is configurable with the RX1TO parameter in the Parameter Block - See the Dalf Owner's Manual.

ERROR HANDLING

If the first byte of the Slave Response is not a CMD character 'A'-'Z', it is the success/fail code and this is the only byte in the response. In this case (as in general for the last byte received by the Master), it must be negatively acknowledged. If the byte has a value equal to **chr_OK**, then it simply indicates that the command was successful and there is no data to return. Anything else is an error code (see the Error List).

I2C MASTER (Host) RULES:

1. The host **must** initiate a READ Operation immediately following every WRITE.

2. The last byte of the response packet must be **"negatively acknowledged**" by the MASTER as defined by the I2C Bus Standard. In the special case where an error has occurred during command execution, or the particular command doesn't return data, the last byte will be the only byte. In the cases where data is returned, the last byte will be the CKSUM byte in the N+3 position of the packet. In this case, there are 2 approaches to determine which byte is the last (CKSUM byte): The first approach would use a "lookup table" to let the MASTER know the value of "N" for every command response. The second approach determines "N" by reading the value of the 2nd byte in the Response Packet (see the sample code).

3. The Dalf Board firmware employs **clock stretching**. This feature of the I2C Specification permits a transmitting device to hold the SCL line low until the data is ready for transmission. This has implications for the Master if a "bit-bang" interface is used (See the Tips Section of this document).

4. The 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 I2C interrupts. The determination of "reasonable" depends on board activity.

5. The host **must** wait for READ and WRITE transactions to complete (or timeout) before initiating a new transaction.

TRANSPORT LAYER

Command Definitions (from 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 (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	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 I2C2 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 bytes; 2's complement format) 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). The I2C Master must wait until all data has been 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.

TIPS FOR DEVELOPING AN I2C2 MASTER

If you are developing the code for an I2C Master to control a Dalf Board over the I2C2 Interface, I have some recommendations.

First: Look at the sample code provided on the CD {<test2.asm, test1.c>:

void MasterI2C2(void) - Command dispatcher.
WriteI2C2_Pkt - Command packet transmittal to the Dalf Board .
ReadI2C2_Pkt - Request a response packet transmittal from the Board.

These routines are provided as guidelines only. They have been used successfully with Dalf-to-Dalf communication (one board configured as a MASTER and the other in the usual configuration as an I2C SLAVE). The Write and Read routines are written in PIC Assembler. The MasterI2C2() is a C function "transaction handler" that dispatches based on the particular command. Even if you are unfamiliar with PIC Assembler, the comments should provide some useful guidance for your own code development.

Second: Consider designing your I2C MASTER as an interrupt driven system. The sample code mentioned above is -NOT- interrupt driven which is perfectly ok for normal operation since the Dalf Board is very responsive. There is one special case that argues for an interrupt driven system. This involves the response from Cmd_Q - the PID Tuning Command. Unlike other commands, Cmd_Q responds with multiple packets spaced evenly in time at timing intervals of 8*VSP msec (see Dalf Owner's Manual). While the code in ReadI2C2_Pkt can be employed in a loop to successfully read all of these packets, most of the time spent in the loop will be wasted while waiting for the next packet to arrive. An interrupt driven I2C Interface for the MASTER would not have this issue.

Third: If you are developing a "bit-bang" interface, be aware that the Dalf SLAVE device employs clock stretching. The implication for the MASTER code is that it will be necessary to test the SCL line before transmit of address or data and before clocking in receive data.

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 I2C2 Interface. The Host 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. See the TE Interface section of the Dalf Owner's Manual for details of the Read Memory and Write Memory commands.

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 motor move cmd received, 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 chr 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	**Chr_OK**

In some cases, multiple error conditions map to the same error codes. For example, there are many potential protocol violations. This is the same error list used by the API. Some errors may not apply to the I2C2 Interface.