Turbo BDM Light interface

(c) 2005, Daniel Malík
rev 1.5

1.0 Introduction to TBDML

1.1 Purpose of this document

This document describes the Turbo BDM Light (TBDML) interface and associated SW libraries and tools. TBDML is a hardware interface which connect between computer and BDM debugging port of Freescale microcontrollers. It enables debuggers and other SW tools to communicate with the microcontroller, download code into its on-chip flash, etc.

1.2 Aspirations and roots of TBDML

I have developed the TBDML hardware and software to satisfy the following requirements:

- low cost
- ease of assembly and prototyping (widely available through hole components only & simple programming interface for downloading firmware)
- open-end SW interface with documented API for easy integration into debuggers and new standalone tools
- easy SW migration under Linux
- support for at least one widely used debugger
- modern and widely available interface for communicating with the computer (USB)
- wide range of target MCU supply voltage (at least from 3.3V to 5V)

The SW APIs are based on the Turbo BDM interface I have developed previously. I have developed TBDM interface to achieve maximum accuracy and performance. It was capable of BDM speeds up to 3.75 Mbit/s and timing resolution of 16.7ns. At the same time it was very complicated, expensive, impossible to build without professionally made PCB and the components were hard to get.

I should also mention that the SW is open source. I am certainly not the best programmer and others should get a chance to make the SW better. I also appreciate that there are situations where somebody might need to do something special and modification of the source code would be needed to achieve it.

2.0 Description of TBDML

2.1 What you get

The TBDML package consists of

- complete HW description which enables you to build the interface
- binary of firmware for the interface, USB drivers and DLL interface library for Windows (I hope to add support for Linux at some point in time - anybody out there who would volunteer to help?)
- source code of the firmware and the DLL interface
- binary of GDI DLL library for Metrowerks Hi-wave debugger

2.2 Hardware

TBDML uses USB as the means of talking to the computer. Here is why:

- I like the concept of USB
- I think that USB is cool
- USB provides power to the interface; no bulky wall adapters and no inefficient regulators with hot heatsinks are needed.
- +5V on the BDM connector can be used to power the target board from the USB, so you can debug your code on the road without a bench power supply (also useful for university classes as you do not need to buy 20 power supplies for students).

The TBDML is based on MC68HC908JB8 MCU from Freescale. My reasons for selecting this MCU are:

- it is relatively speaking low-cost
- it has USB interface
- its I/Os operate from 3.3V rail and this enables simple interfacing to wide range of target MCU voltages
- it comes in dual-in-line package which is simple to handle on strip boards and wire-wrap boards.
- the development environment and the Hi-wave debugger looks the same for both HC08 and HC(S)12/S12X so I was able to develop the code fast without having to get used to another toolset.
- it can be programmed without specialist equipment and all the development tools are available free of charge

Disadvantages of the HC908JB8:

- programming requires RS232 interface and connector
- low bus speed of 3MHz (this limits the useable range of crystals connected to the target MCU)

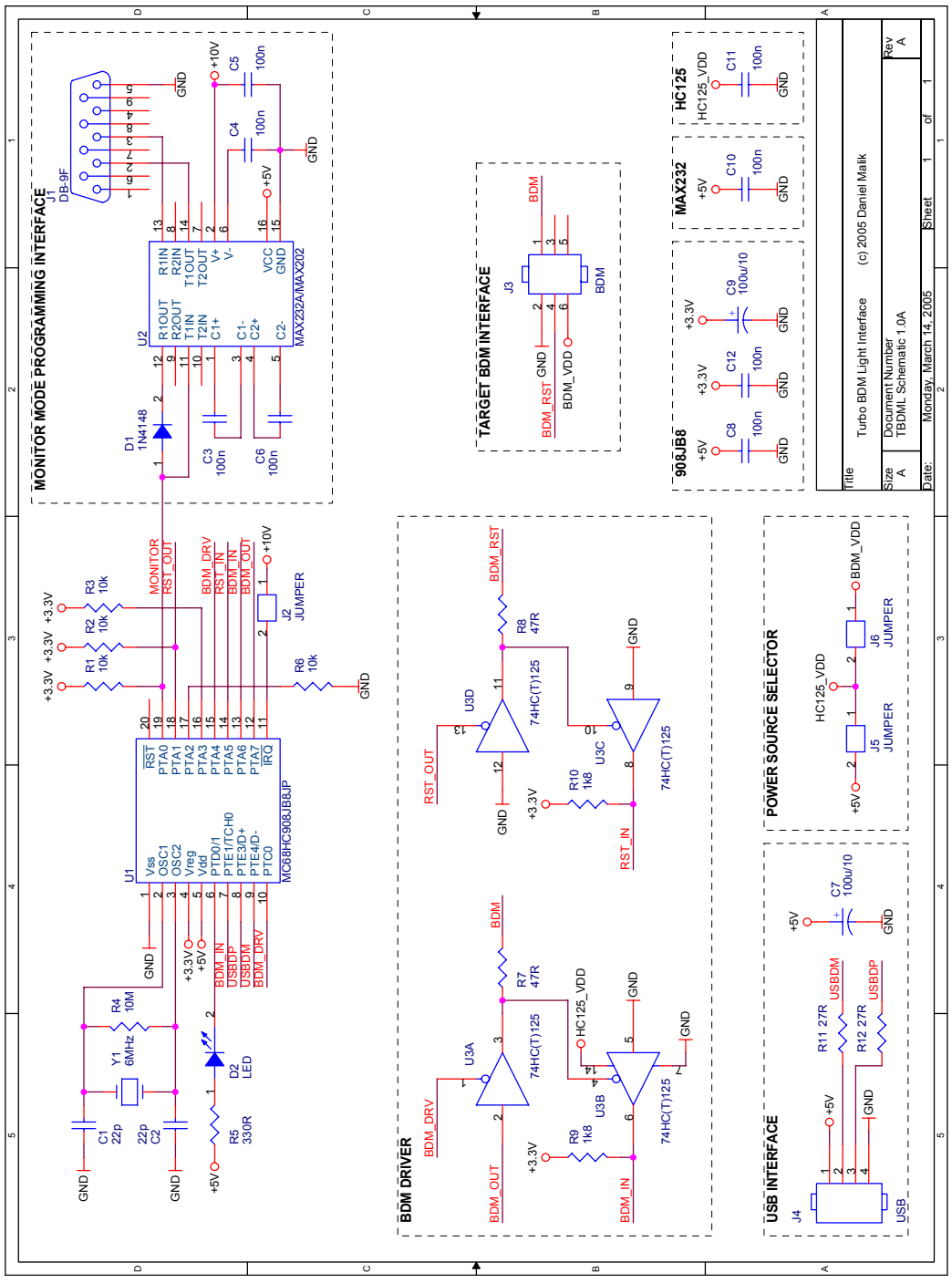

FIGURE 1. TBDML schematic

Schematic diagram of the TBDML interface is shown in figure 1. The interface has three main parts: the HC908JB8 MCU itself, BDM interface driver based on 74HC(T)125 buffer with tri-state outputs and RS-232 programming interface based on MAX232A or MAX202 driver. Please note that you do not need to populate the RS-232 interface in case you have some other means of programming the micro or a standalone RS-232 interface you can temporarily hook-up to this board.

2.2.1 Remarks on the BDM interface driver

I have used the 74HC125 to achieve low-cost translation of BDM signal with voltages anywhere between 3.3V and 5V to 3.3V logic of the MCU. When you look into the datasheet of HC125, you will notice that the logic high voltage levels coming out of the HC908 are outside of the guaranteed limits for the HC125 when powered by 5V supply from the target board. Typical performance of HC125 at room temperature is however far better than the guaranteed limit from the spec and the buffer will interpret anything even slightly above 2.5V as logic high (possibly with slight violation of the timing parameters). However if you do not feel that you want to risk this, you can exchange the buffer for 74HCT125. HCT125 has lower limit for minimum logic high voltage and is guaranteed to work. The price you will pay is that you cannot use HCT125 below 4.5V. The ideal solution is therefore to provide a socket for the buffer and to change it as needed. I have not done this on any of my own boards and I am relying on the fact that the HC125 works reliably even slightly outside of its specification. I expect that it will be fully reliable when used in lab/workshop temperatures.

2.2.2 Monitor mode programming interface

The monitor mode interface is a simple voltage-level converter between the single-wire MON08 interface and standard RS-232. When jumper J2 is closed the higher voltage from the charge pump is connected to the IRQ pin of the HC908JB8 and causes the part to enter the monitor mode. Further details can be found in section 3.1 on page 11.

2.2.3 Power selection

The TBDML has 3 possible set-ups for distributing power. These are detailed in the table below.

Setting	JP5	JP6	Description
1	closed	open	BDM driver circuit powered from +5V supplied by USB, no power is drawn from the target board. Make sure the power supply voltage of the target MCU is 5V.
2	open	closed	BDM driver powered from the target board. (default)
3	closed	closed	Both the BDM driver circuit and the target board are powered by +5V supplied by the USB. Make sure the target board requires less than 200mA (you can increase this limit by modifying the firmware source, but not all hubs support high currents) and that there is no conflict with power supply on the target board.

2.2.4 Printed Circuit Board

PCB I have designed for TBDML is shown in figure 2.

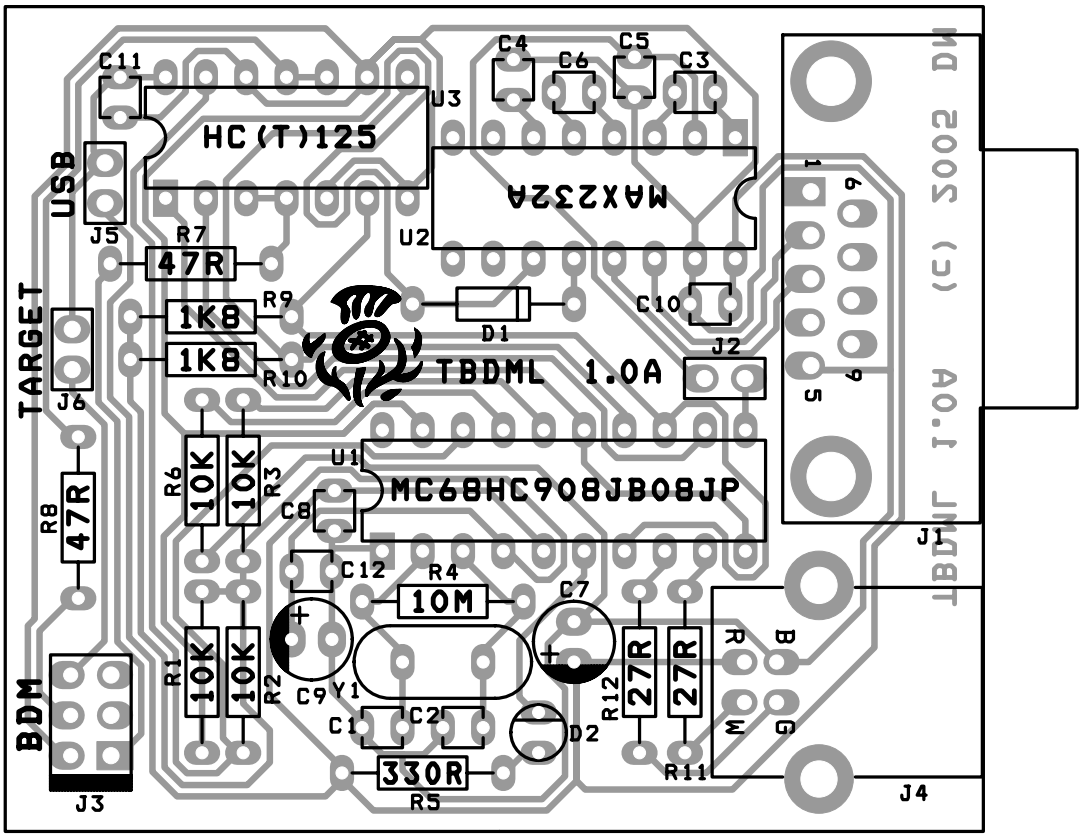

FIGURE 2. TBDML PCB

The PCB is designed to be single sided only (with no wire links!). The design rules have been set to 12 mil spacing and 15 mil minimum copper width. The PCB should be fairly cheap to produce (it is costing me around $3 a piece in

small quantity with both solder mask and silk screen). The real size of the PCB is roughly 52 x 62 mm.

You can download the standard gerber files and have the PCB made, make the PCB in your garage, design your own PCB or alternatively you can also populate the interface on a piece of prototyping board. I have wire-wrapped the first prototype and it did not give me any problems at all.

Note that the PCB has been provided with footprint for the USB B connector. This is actually violation of the USB specification as low-speed devices should have the cable hard-wired to them, but I have found a detachable cable very useful. If you do not feel like violating the specification, you can solder the cable straight into the PCB. The PCB is marked with wire colours and two extra holes are provided for strapping the cable to the PCB in case you wish to do this.

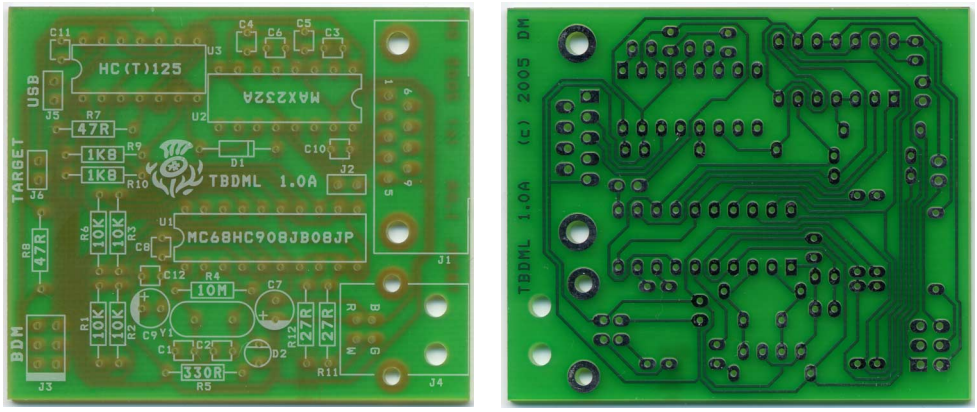

FIGURE 3. TBDML PCB

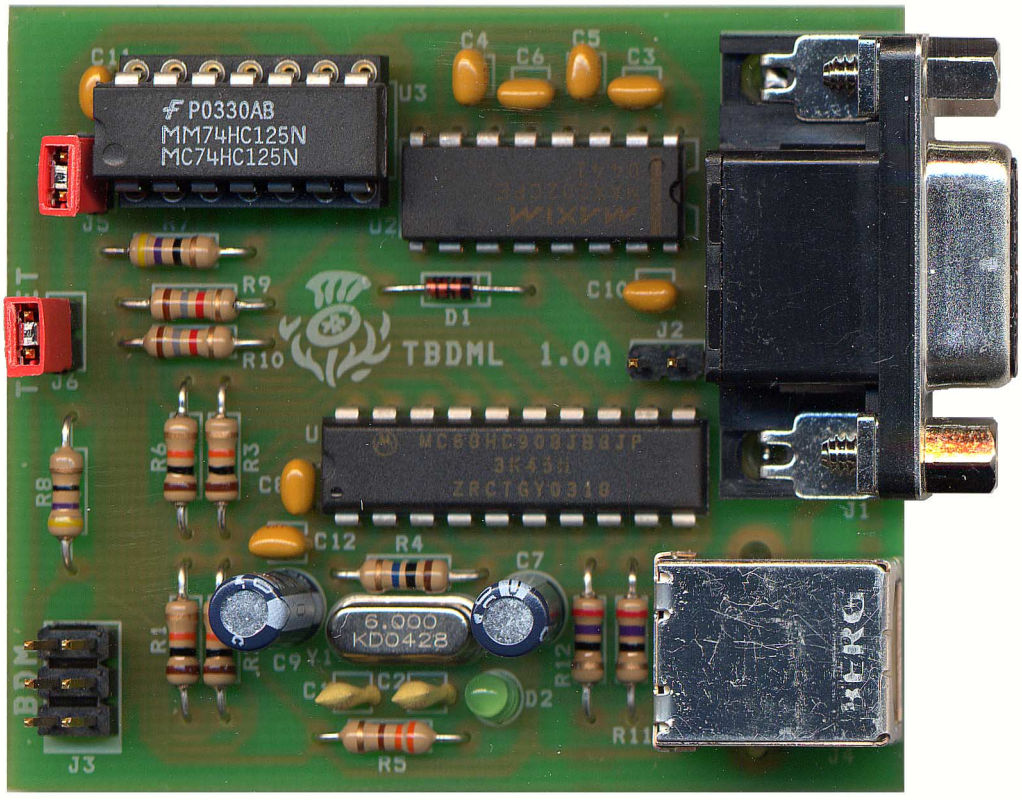

FIGURE 4. Populated TBDML PCB

2.2.5 Getting the components

To make life slightly easier for you I have listed order numbers of the components needed to build the interface in the table below. However please note that you will probably be able to get the componets at much lower price at your local high street shop. You might also want to try to get some of the components for free as samples (Maxim and Freescale offer free samples of the parts I am using in the design and this is how I am getting parts myself :-).

Item	Count	Reference	Value	Farnell	RS Comp	Digikey
1	2	C2,C1	22p	236-962	264-4668	495-1004-1-ND
2	8	C3,C4,C5,C6,C8,C10,C11,C12	100n	656-136	264-4933	399-2150-ND
3	2	C9,C7	100u/10	361-8390	205-1656	P5111-ND
4	1	D1	1N4148	368-118	446-8551	1N4148FS-ND
5	1	D2	LED	329-9480	228-5944	160-1080-ND
6	1	J1	DB-9F	410-6118	160-2742	182-709F-ND
7,8	1	J2,J5,J6,J3	JUMPER, BDM	412-9465	531-942	S2012-36-ND
9	1	J4 (option 1)	hardwired USB cable	395-0074 (cut end)	324-8362	AE1143-ND
9	1	J4 (option 2)	USB "B" receptacle	152-754	458-1648	WM17108-ND
10	4	R1,R2,R3,R6	10k	509-280	131-378	10KQBK-ND
11	1	R4	10M	509-644	135-667	10MQBK-ND
12	1	R5	330R	509-103	131-198	330QBK-ND
13	2	R7,R8	47R	509-000	131-097	47QBK-ND
14	2	R9,R10	1k8	509-190	131-283	1.8KQBK-ND
15	2	R11,R12	27R	508-974	131-069	27QBK-ND
16	1	U1	MC68HC908JB8JP	348-0252	445-6744	MC68HC908JB8JP-ND
17	1	U2	MAX232A	270-957	299-913	MAX202CPE-ND
18	1	U3	74HC(T)125	378-458 (381-998)	169-7403 (634-596)	296-12781-5-ND (296-8386-5-ND)
19	1	Y1	6MHz	221-582	226-1645	X413-ND

2.3 Software

The basic SW package for the TBDML interface consists of four different components:

- firmware in HC908JB8
- interface DLL (TBDML.DLL)
- USB driver (LIBUSB)
- GDI DLL plug-in for the Metrowerks Hi-wave debugger

All the components are intended to be used as binaries by majority of users. For those who would like to look deeper I am providing source code of the firmware and the interface DLL.

The LIBUSB is open source software available under combination of GNU general and lesser general public licenses.

The GDI DLL for the Metrowerks Hi-wave debugger was created based on information which is not available in the public domain. The license attached to these files is preventing me from disclosing them and the source code of this library.

2.3.1 TBDML DLL API

Debugging and other tools should primarily use the TBDML DLL to interface to the TBDML tool. This section describes the API the TBDML DLL v1.0 offers.

unsigned char tbdml_dll_version(void)

Returns version of the DLL in BCD format (major in upper nibble and minor in lower nibble).

unsigned char tbdml_init(void)

Initialises the USB interface and returns number of TBDML devices found attached to the computer. This function needs to be called before a device can be opened.

unsigned char tbdml_open(unsigned char device_no)

Opens communication with device number *device_no*. First device has number 0. Returns 0 on success and non-zero on failure. A device must be open before any communication with the device can take place.

void tbdml_close(void)

Closes communication with currently opened device.

unsigned int tbdml_get_version(void)

Returns version of HW (MSB) and SW (LSB) of the TBDML interface in BCD format.

unsigned char tbdml_get_last_sts(void)

Returns status of the last executed command: 0 on success and non-zero on failure.

unsigned char tbdml_set_target_type(target_type_e target_type)

This function sets target MCU type. *target_type* can be either *HC12* or *HCS08*. Returns 0 on success and non-zero on failure.

unsigned char tbdml_target_sync(void)

Measures BDM frequency of the target using the SYNC BDM feature and connects to the target. Returns 0 on success and non-zero on failure (no device connected or the SYNC feature not supported). If this function succeeds, there is no need to set the BDM communication speed as it is measured automatically.

unsigned char tbdml_target_reset(target_mode_e target_mode)

Resets the target MCU to normal or special mode. *target_mode* can be either *SPECIAL_MODE* or *NORMAL_MODE*. Returns 0 on success and non-zero on failure (reset pin stuck to ground, etc.).

unsigned char tbdml_bdm_sts(bdm_status_t *bdm_status)

bdm_status is a pointer to user allocated structure which the function fills with current state of BDM communication. Returns 0 on success and non-zero on failure.

The structure has the following format:

```
typedef struct {
        ackn_state_e ackn_state;
        reset_state_e reset_state;
        connection_state_e connection_state;
} bdm_status_t;
```

ackn_state can be either *ACKN* (target supports ACKN BDM feature) or *WAIT* (target does not support ACKN BDM feature).

reset_state can be either *RESET_INACTIVE* (no reset activity detected) or *RESET_DETECTED* (target was reset since the last call). *reset_state* defaults to *RESET_INACTIVE* after each call.

connection_state can be *NO_CONNECTION* (no target MCU detected), *SYNC* (target supports the SYNC BDM feature) or *MANUAL_SETUP* (BDM speed was set-up by calling tbdml_set_speed - see below).

unsigned char tbdml_read_bd(unsigned int address)

Reads one byte from the BDM memory area at the supplied address.

unsigned char tbdml_write_bd(unsigned int address, unsigned char data)

Writes one byte to the BDM memory area at the supplied address. Returns 0 on success and non-zero on failure.

unsigned char tbdml_target_go(void)

Starts target code execution from current PC address. Returns 0 on success and non-zero on failure.

unsigned char tbdml_target_step(void)

Steps over a single target instruction. Returns 0 on success and non-zero on failure.

unsigned char tbdml_target_halt(void)

Brings the target into active background mode (i.e. debug mode with user code execution halted). Returns 0 on success and non-zero on failure.

unsigned char tbdml_set_speed(float crystal_frequency)

Sets the BDM communication speed. *crystal_frequency* is crystal (or external source) frequency in MHz. Returns 0 on success and non-zero on failure. It is essential to provide frequency accurate at least to 2 decimal places (in MHz).

float tbdml_get_speed(void)

Returns crystal (or external source) frequency of the target in MHz.

unsigned char tbdml_read_byte(unsigned int address)

Reads one byte from memory at the supplied address.

void tbdml_write_byte(unsigned int address, unsigned char data)

Writes one byte to memory at the supplied address.

unsigned int tbdml_read_word(unsigned int address)

Reads one word from memory at the supplied address. The address must be aligned (even).

void tbdml_write_word(unsigned int address, unsigned int data)

Writes one word to memory at the supplied address. The address must be aligned (even).

void tbdml_read_block(unsigned int address, unsigned int count, unsigned char *data)

Reads *count* bytes from address *address*. The data is written to a user supplied buffer.

void tbdml_write_block(unsigned int address, unsigned int count, unsigned char *data)

Writes *count* bytes to address *address*. The data is take from a user supplied buffer.

unsigned char tbdml_read_regs(registers_t *registers)

Reads contents of target registers. Returns 0 on success and non-zero on failure. The register values are filed into user allocated structure of the following format:

```
typedef union {
        struct {
                unsigned int pc;
                unsigned int sp;
                unsigned int ix;
```

```
            unsigned int iy;
            unsigned int d;
        unsigned int ccr;
        } hc12;
        struct {
            unsigned int pc;
            unsigned int sp;
            unsigned int hx;
            unsigned int a;
            unsigned int ccr;
        } hcs08;
} registers_t;
```

void tbdml_write_reg_pc(unsigned int value)

Writes a new value into the PC target register.

void tbdml_write_reg_sp(unsigned int value)

Writes a new value into the SP target register.

void tbdml_write_reg_x(unsigned int value)

Writes a new value into the H:X (S08) or IX (HC(S)12/S12X) target register.

void tbdml_write_reg_y(unsigned int value)

Writes a new value into the IY target register (HC(S)12/S12X only).

void tbdml_write_reg_d(unsigned int value)

Writes a new value into the A (S08) or B:A (HC(S)12/S12X) target register.

void tbdml_write_reg_ccr(unsigned int value)

Writes a new value into the CCR target register.

3.0 Installation

3.1 Programming firmware into TBDML

Before your computer can recognize the TBDML interface as a valid USB peripheral, the firmware needs to be downloaded into the HC908JB8 micro-processor. The following description assumes that you are using the TBDML PCB described in section 2.2.4 on page 5.

There are many ways of programming binary image of the firmware into HC08 microcontroller. The procedure detailed here makes use of the PROG08 utility which is available from P&E Microcomputer Systems (www.pemicro.com) free of charge.

Step 1: Force the HC908JB8 into monitor mode

1. Close jumper J2. This will apply high voltage to the IRQ pin of the micro after the board is powered up.
2. Apply power to the board. This can be done for example by plugging the board into USB port of a computer. The computer will not recognize attachment of the USB device when jumper J2 is closed.

Step 2: Downloading the firmware in HC908JB08

1. Attach the RS-232 port of the TBDML board to the computer and start the PROG08 tool.
2. Configure PROG08 to use the correct COM port and 9600 baud. Select hardware class 3 (“Direct serial to target with MON08 serial circuitry”).
3. If the device already contains some code, check the “Ignore security failure” checkbox, otherwise select security key corresponding to a blank device.
4. Connect to the device. You will be requested to cycle power of the device (for example by disconnecting and reconnecting the USB cable).
5. After connecting to the device the PROG08 tool will display a prompt asking for programming algorithm file. Select the 908_JB8 programming algorithm.
6. Erase the device (if not blank).
7. Specify S-record file (bdm_light.sx).
8. Program the module.

Step 3: Finish

1. Close the PROG08 tool.
2. Power down the TBDML board and disconnect the RS-232 cable.
3. Open jumper J2

Now the TBDML board should be fully functional and ready to use.

3.2 Installing Windows drivers

The following procedure details standard steps when installing new hardware drivers under the Windows operating system. I assume that you have downloaded the TBDML windows driver package and unpacked it to a suitable directory on your computer.

1. Make sure jumper J2 is open and attach the TBDML board to USB port of the computer.

2. Windows will detect attachment of a new hardware device and will start the driver installation procedure (see figure 5).

FIGURE 5. "Found new hardware" window

3. Select option "Install from a specific location" and click next. Then specify location of the drivers (see figure 6).

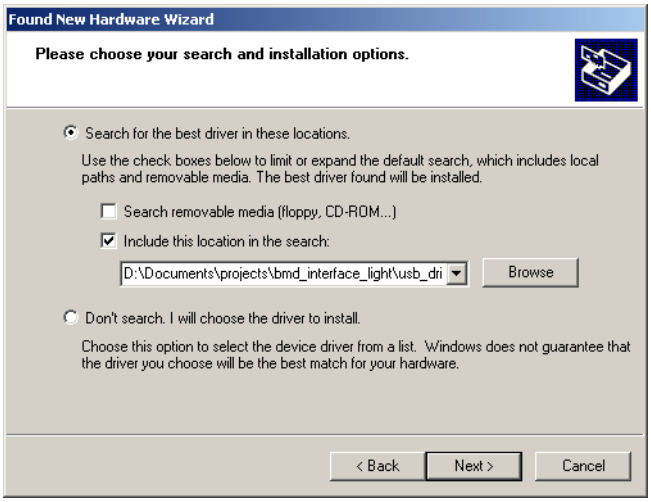

FIGURE 6. Specifying location of drivers

4. Windows will then install the required driver and DLL files (see figure 7).

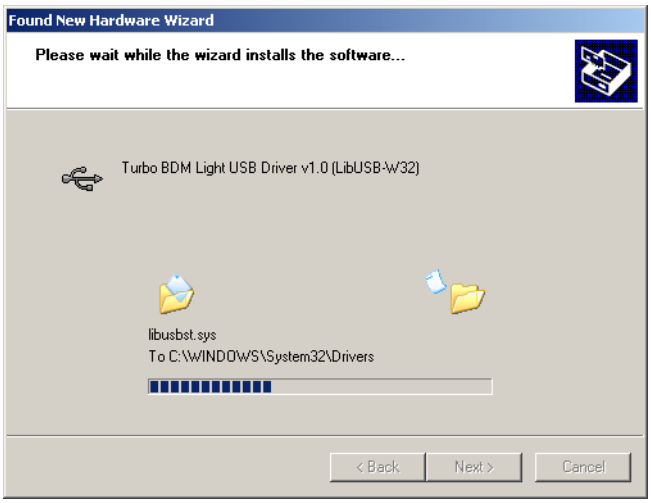

FIGURE 7. Driver installation in progress

5. Once the installation procedure is finished the device will be ready to use (see figure 8). Restart of Windows should not be needed.

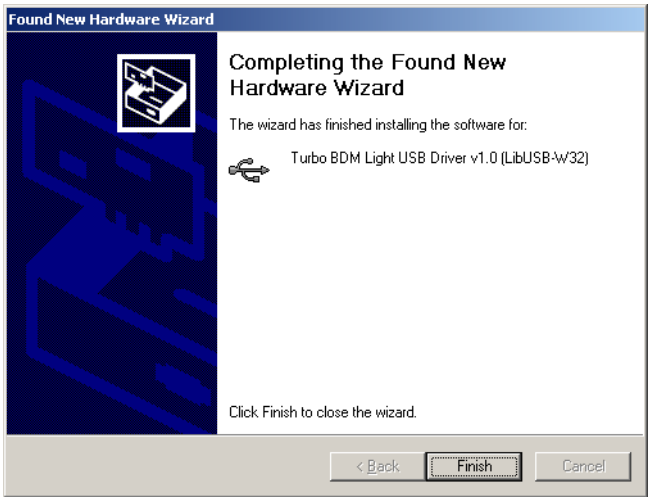

FIGURE 8. Finishing the installation

3.3 Using the DGI DLL under the Hi-wave Debugger

The procedure detailed in this section shows how to configure the Hi-wave HC(S)12/S12X debugger from Metrowerks to work with the TBDML interface. Please make sure you download the latest version of the tools from Metrowerks as the debugger interface of the older version does not necessarily support the required features. I am using CodeWaririor version 4.5.

1. Open the debugger and type command “set gdi” in the command window. If the command window is not open, you can open it through the Component->Open menu.
2. Then enter the correct path to the GDI DLL which is included in the driver package (tbdml_gdi12.dll). The common place for all the GDI interface libraries is in the “prog” directory within the Metrowerks tools tree, but you are free to place it somewhere else as well.

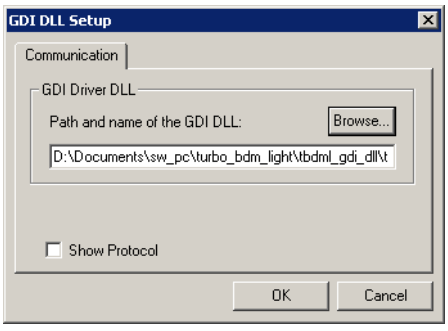

FIGURE 9. “GDI DLL Setup” window

3. The “TBDML HCS12” menu should now appear in the debugger’s pull-down bar and the debugger is ready to use.
4. Press the Save button to save the debugger configuration.

4.0 Performance

4.1 Limits of target MCU crystal frequency

The TBDML interface uses relatively very slow HC908JB8 MCU which runs at 3MHz bus frequency. This has impact on the maximum crystal clock frequency of the target MCU. My calculations show that the TBDML interface with the JB8 MCU is guaranteed to connect to devices with crystal frequencies up to 16.5 MHz. Practical experiments have shown that my calculations are probably a bit pessimistic and I was successfully able to connect to MCUs with crystal clock frequency up to 19.5MHz.

For practical reasons the TBDML also limits the minimum target crystal frequency to 0.9MHz. My experiments indicate that the real minimum frequency is slightly below 0.8MHz.

4.2 Response time and transfer rate

By the nature of the USB protocol the response time for low and full speed devices cannot be below 1ms. I have tried to optimize the communication protocol on the USB to achieve maximum throughput. Practical limitations (caused by the Windows operating system) cause additional delays however. Average (since under Windows nothing is certain) execution times for different kinds of commands are detailed in the following table.

Command type	Description	Average execution speed
Short	Commands which transfer up to 5 bytes of data into TBDML and require no return values.	3ms
Normal	Commands which transfer up to 5 bytes of data into TBDML and request up to 8 bytes of return values.	4ms
Data transfer	Commands which transfer large blocks of data.	6.7 kB/s

When programming the flash of the target MCU there is additional overhead created by the flash programming routines. The speed is also dependant on crystal frequency (the higher, the better). The Metrowerks Hi-wave debugger with TBDML interface connected to HCS12 target with 4MHz crystal typically programs the flash at 2.7kB/s rate.

5.0 Ideas for further work

1. When the target MCU needs to operate with crystal faster than 16MHz the HC908JB8 is not fast enough. It should be relatively simple to port the SW onto HC908JB16 which runs twice as fast and would therefore be good up to 33MHz of target crystal frequency. The downside is that HC908JB16 only comes in SMD package and would therefore be hard to work with on prototyping boards. Other subtle differences between the two parts would

require additional changes to the hardware of the interface. However at a first glance it seems that the range of possible supply voltages for the target MCU would benefit from these changes and cover also the very low voltage parts (2.0-5.5V).

2. It would be nice to port the TBDML interface library under linux. The LIBUSB package it is based on was primarily developed for Linux, BSD and MacOS rather than Windows. It should therefore be theoretically relatively simple to create support on these platforms. Providing one has the experience and time... Any volunteers?
3. At the moment the only debugger supported is the Hi-wave from Metrowerks. Since it only runs under Windows it does not make much sense to create support for TBDML on other platforms until other debuggers (like GDB) are supported. Would anyone be willing to help with this task?

6.0 Support

I am developing TBDML in my free time. It comes for free, so do not expect much in terms of support. If you discover any bugs or have difficulties with the interface I might have a look at it, but I cannot guarantee it.

7.0 License

I am making all the work (with the exception of the GDI DLL) available for everyone under the GNU general public license version 2. I do not want to restrict support for the interface in commercial products however and I can provide you with a copy of the work under GNU lesser general public license which enables use of the work in commercial applications - ask for it if you need it.

8.0 References

[1] LIBUSB documentation, http://libusb.sourceforge.net/

[2] BDM documentation, S12XBDMV2.PDF available from Freescale

[3] Datasheet to HC908JB8, MC68HC908JB8.PDF available from Freescale

[4] Datasheet to HC908JB16, MC68HC908JB16.PDF available from Freescale

[5] Datasheets to MAX202 and MAX232 available from Maxim Integrated Products

[6] Documentation to Generic Debugging Interface, available from Tasking