BDM Interface for Motorola 683xx MCU
Usage with GDB Debugger

Pavel Pisa (pisa@cmp.felk.cvut.cz)

2000.7.30.

Abstract

The BDM Interface can supply more expensive ICEs (In Circuit Emulator) for Motorola 683xx family of processors based on the CPU32 core (68332, 68333, 68334, 68336, 68376 and 68340). This document tries to describe the CPU32 BDM interface and its usage with GNU debugger under the Linux operating system. Some newer members of Motorola MCUs use similar, but not compatible BDM interfaces, as well. Last section tries to summarize information about these interfaces.

Contents

1 BDM Overview

BDM mode of the CPU32 halts execution of a normal machine code fetched from the memory and starts the internal MCU microcode to process commands received from a dedicated serial debug interface. These commands can be used to view and modify all CPU32 registers and to access into on-chip and external memory locations.

The CPU32 processor must be started in a special mode to enable BDM interface. This is achieved by holding the BKPT pin low during the reset time.

Switch to the BDM mode can be enforced by the following tree ways:

- Driving the BKPT pin low when a fetch of an instruction occurs - the BDM mode is entered after processing this instruction

- Inserting BGND instruction (4AFAh) into the program memory
- Double bus fault, which in a normal case leads to CPU halt

Return from the BDM mode is initialized by the BDM command GO or CALL.

2 Hardware and BDM Protocol

Motorola has defined a standard pinout for the debug connector, which is compatible with most of the development tools. Older versions have only eight pins and the newer ones add two additional pins for enforcing bus error and memory interface monitoring. Pins 2 to 10 of the new connector version are equivalent to the pins 1 to 8 of the older one. Table 1 describes the function of these

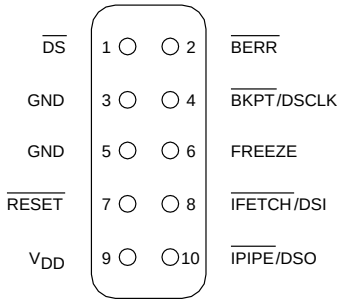


Figure 1: Standard Ten Pin BDM Connector

pins. Hardware dimensions of the connector are equivalent to the jumper array, which has 100 mils (2.54 mm) spacing.

BDM uses 17 bit serial synchronous communication with the CPU32 processor. All data and command transfers are performed in an MSB first format. An internal CPU32 receiver is implemented by shift and latch registers. The CPU32 latches every input bit value on the DSI line at the time of rising edge detection of the DSCLK signal. Because of the DSCLK edge detection is performed synchronously with the system clock, the maximum DSCLK frequency is equal to one half of a system clock frequency. A 17 bit input word is latched after 17 rising edges on the DSCLK line then the CPU32 microcode sequencer is started to perform instruction or process extension words. Transmit latch register is updated by the CPU32 continuously. The transmit shift register and DSO pin reflect changes of the latch register until the first low level of 17 bit protocol is detected on the DSCLK line. Then the state of the DSO line can be read as a MSB received bit. Then the transmit shift register is not updated by the transmit latch register until all 17 bits are read. The DSO line is changed only after rising edges of the DSCLK line during the rest of transfer. Command and data transfers initiated by the development system should clear

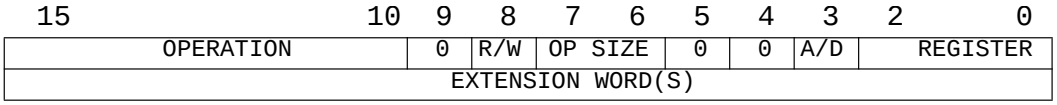


Figure 2: General CPU32 BDM Command Format

bit 16. The current implementation ignores this bit; however, Motorola reserves the right to use this bit for future enhancements. The CPU32 returns 17 bit status or value every time 17 bits are send to it. The meaning of 17 bit status is described in Table 2. Some commands except the first command word need an additional address and data words. Figure 2 shows the general BDM instruction format without 16-th bit.

Table 3 contains possible BDM commands for the CPU32 processors family. I have noticed, that the new ColdFire family processors use same basic command set with additional real-time

Pin Number	Pin Name	Description
1	DS	Data strobe from target MCU.Not used in current interface circuitry
2	BERR	Bus error input to target. Allows development system to force bus error when target MCU accesses invalid memory
3	VSS	Ground reference from target
4	BKPT/DSCLK	Breakpoint input to target in normal mode;development serial clock in BDM.Must be held low on rising edge of reset to enable BDM
5	VSS	Ground reference from target
6	FREEZE	Freeze signal from target.High level indicates that target is in BDM
7	RESET	Reset signal to/from target.Must be held low to force hardware reset
8	IFETCH/DSI	Used to track instruction pipe in normal mode.Serial data input to target MCU in BDM
9	VCC	+5V supply from target.BDM interface circuit draws power from this supply and also monitors 'target powered/not powered' status
10	IPIPE/DSO	Tracks instruction pipe in normal mode.Serial data output from target MCU in BDM

Table 1: 10 Pin BDM Connector Description

Bit 16	Data	Message Type
0	xxxx	Valid Data Transfer
0	FFFF	Command Complete; Status OK
1	0000	Not Ready with Response; Come Again
1	0001	BERR Terminated Bus Cycle; Data Invalid
1	FFFF	Illegal Command

Table 2: BDM Status or Values Returned by CPU32

commands. Changed RSREG and WSREG commands need to address more registers and that is why an additional register address word is necessary for these instructions. Another big change can be seen with ColdFire BDM cable, because of the ColdFire has single-stepping flip-flop built inside.

Table 4 describes the meanings of the last variable nibble or byte values in command codes.

3 Cable Wiring and Logic

There exist two standard wirings of a cable between the CPU32 BDM interface and standard PC printer port. The first is public domain interface (PD_BDM). It was published by Motorola (its support BBS) and can be used with free BD32 (bd32v122.zip) debugger and BDM library example implementation (bdm-v090.zip). Both are writted by Scott Howard. The second cable is provided with Motorola commercial systems and is known as ICD_BDM cable. This cable can be used with both above mentioned programs and with the free TPU debugger and downloader (tpubug.zip).

Command	Mnemonic	Code	Additional Words and Notices
Read D/A Register	RREG	`218r`	receive two words with value from CPU32
Write D/A Register	WREG	`208r`	send two words with value to CPU32
Read System Register	RSREG	`258s`	receive two words with value from CPU32
Write System Register	WSREG	`248s`	send two words with value to CPU32
Read Memory Location	READ	`19tt`	send 2 word address and receive 1 or 2 words value
Write Memory Location	WRITE	`18tt`	send 2 word address and 1 or 2 words value
Dump Memory Block	DUMP	`1Dtt`	receive 1 or 2 words value from next memory locationto location selected by previous READ command
Fill Memory Block	FILL	`1Ctt`	send 1 or 2 words value for next memory locationto location selected by previous WRITE command
Resume Execution	GO	`0C00`	Pipe is re-filed from RPC location
Patch User Code	CALL	`0800`	Current program counter is stacked at the locationof the current stack pointer and two additional wordsdefine subroutine start address
Reset Peripherals RST Asserts	RST	`0400`	Asserts RESET for 512 clock cycles,but the CPU is not reset by this command
No Operation	NOP	`0000`	NOP performs no operation and may beused as a null command

Table 3: CPU32 BDM Commands Summary

The schematic diagram of one of possible PD cable implementations is in figure 3. It is a simple implementation with 8 pin cable only, but it works for me without serious problems with 1 m cable from PC and 20 cm cable to MC68332.

The cable compatible with Motorola ICD32 system can be seen in figure 4. I have seen an original schematic for ICD32 cable, but I do not have this cable, so GAL16V8 function is only my own solution. In my experience, it works with all free software I have (Linux BDM driver, DB32, BDM library and TPU debugger). I am not sure about legal state of this cable, but it can be used with free Motorola software and free source for BDM library describes its function, so it should be free. This cable works better than the previous one for the following three reasons:

- logic levels of all signals are sharped by GAL16V8
- bidirectional CPU32 DSI/IFETCH signal is controlled by tristate buffer
- breakpoint and step logic use better level controlled mechanism

4 GDB and BDM Driver for Linux

GNU debugger is used in many native and cross development tool-chains in UNIX type environment. It is a very powerfull debugger controlled from its command line. There exist many interactive menu-driven and mouse-driven user interfaces for this debugger, too (for example GDBTk, DDD, Rhide and XXGDB). This debugger is very well suited for the cross-development

Symbol	Value	Mnemonic	Meaning
r	0 to 7	D0 to D7	Data Register
	8 to F	A0 to A7	Address Register
s	0	RPC	Return Program Counter points where execution will continue
	1	PCC	Current Instruction Program Counter points to first byte of last executed instruction it contains 00000001 when double bus fault appears immediately after reset
	8	ATEMP	Temporary Register A
	9	FAR	Fault Address Register
	A	VBR	Vector Base Register
	B	SR	Status Register
	C	USP	User Stack Pointer
	D	SSP	Supervisor Stack Pointer
	E	SFC	Source alternate function type of bus cycle MOVES instruction and BDM memory transfers
	F	DFC	Destination alternate function of bus cycle MOVES instruction and BDM memory transfers
tt	00	BYTE	8 bit data in least significant byte of one word
	40	WORD	16 bit data transferred in one word
	80	LONG	32 bit data transferred in two words

Table 4: Values Ored with BDM Commands

for 32 bit embedded targets. It recognize most of the Motorola MCUs with CPU32 and ColdFire processor cores. These targets may be connected by the serial line using Motorola board ROM monitor or special protocols for some operating systems (for example VxWorks). Such target debugging can be achieved by the GDB remote target debugging by any usual serial stream connection (RS-232 or TCP/IP connection).

To use BDM interface by GDB, two problems must be solved. First, it is not good practice to directly manipulate by ports under UNIX systems. It means that the kernel mode BDM driver should be written to implement the BDM character device. Such device can accept and perform regular read/write system calls and for special action (for example single step) use IOCTL interface. The second part must be done to enable GDB to understand and send BDM commands by read/write interface to the BDM driver and controll target state through the driver IOCTL interface.

In future, such two layer implementation can be usefull for GDB independence on the host system, because only the BDM driver will be host specific. Recently, this driver exists for Linux operating system. Patch files for GDB-4.16 and GDB-4.17 exist to use this driver. The authors of the BDM driver and GDB target interface are stated bellow

- Scott Howard, original author of Motorola BDM library and utilities, Feb 93
- M. Schraut, original author of BDM driver
- Gunter Magin <magin AT skil.camelot.de>, maintainer of BDM and GDB
- W. Eric Norum <eric AT skatter.usask.ca>, who did enhancements and Next-Step-Port in Jun 95
- Pavel Pisa <pisa AT cmp.felk.cvut.cz>, some enhancements and GDB-4.17 patch update May 98

- Peter Shoebridge <peter AT zeecube.com>, wrote Windows NT version of driver for CPU32 and ColFire in Jan 99

The original version of GDB patches and BDM driver are stored in Gunter Magin's archive[1]. My modified patches for GDB with Linux BDM driver source can be found under names `gdb-4.17-bdm-patches.tar.gz` and `gdb-4.16-bdm-patches2.tar.gz`.

Gunter Magin is preparing new updated version of patches now. He is developing new version of the ICD compatible cable with ispGAL22V8. I suggest to wait for this more reliable design or use my gdb-4.17 version of patches with GAL16V8 cable for impatient ones.

The new experimental version for gdb-4.18 is based on latest Magin's code and contains all my changes for the Linux 2.2.x support and better timing (at least I hope so). This version can be foud in the archive `gdb-4.18-bdm-patches-pi1.tar.gz`. There is an initial version of the ispGAL22V8 based EFICD, as well.

5 GDB with BDM Setup

To start debugging session, next things must be set-up correctly. The board with one of the Motorola MC683xx processors must be connected to a PC printer port by one of debugging cables mentioned above (ICD32 or PD cable). The BDM driver must be compiled for a correct Linux kernel version (provided driver sources should work with most of 2.0.xx and 2.1.xx kernels, version for gdb-4.17 was tested with 2.2.1 kernel too). The sources of BDM driver can be found in "gdb-4.17-bdm-patches.tar.gz". Next commands can be used to compile driver.

```
cd /usr/src
tar -xzf gdb-4.17.tar.gz
cd gdb-4.17-bdm-patches/bdm_driver
make
```

Makefile should be checked and edited before compilation. Lines of highest importance are

```
INTERFACE+= -D PD_INTERFACE
INTERFACE+= -D ICD_INTERFACE
AUTOLOADING=-DMODVERSIONS
#BDM_DEFS += -D BDM_TRY_RESYNCHRO
#CFLAGS+= -D__SMP__
```

Every line can be commented out by "#" character. The first two lines enables both possible cable types. The third line is needed if kernel is compiled with kernel symbols versions. The fourth line can help if there are lost of BDM sync after single-stepping and break. The last line is needed for SMP kernels.

The compiled BDM driver "bdm.o" should placed to "/lib/modules/<kernel_version>/misc" directory. The driver must be inserted into the kernel when modules are used (`# insmod bdm`). When kernel autoloading of missing modules is used (kmd or kerneld), next line can be inserted into file "/etc/modules.conf".

```
alias char-major-53 bdm
```

Then "`depmod -a`" must be run. No manual insertion of the BDM driver is needed in such case.

Special character files must be created. Standard names for above described cables and driver sources are pd_bdm0, 1, 2 and icd_bdm0, 1, 2 (special files can be created by provided MAKEDEV script). Ending numbers of special files select used printer port base address (0..378h, 1..278h, 2..0x3BCh). Special files select between public domain (pd_bdmx) and ICD32 cable (icd_bdmx). In most cases, only one target processor is used, so it is a good practice to make symbolic link bdm to the used interface special file (for example `ln -s /dev/pd_bdm0 /dev/bdm`).

Patched version of the GDB must be compiled. Next command sequence can be used to prepare and compile the GDB.

```
cd /usr/src
tar -xzf gdb-4.17.tar.gz
patch -p <gdb-4.17-bdm-patches/gdb-4.17.patch
cd gdb-4.17
./configure --target=m68k-bdm-coff
make
make intall
```

This procedure should install m68k-bdm-coff-gdb executable into /usr/local/bin, but no warranty is given. The best way is to check results after each step and install gdb manually.

6 GDB with BDM Usage

The compiled GDB executable must be start and the target must be connected. Setting up of the BDM interface can be prepared as a GDB script or automatic initialization script. For the target without on-board memory setup code, chipselects and system integration module (SIM) must be initialized (from gdb .init script or .bdmmb file). A step by step manual initialization from GDB prompt is shown in the next example

```
target bdm /dev/bdm
set remotecache off
bdm_timetocomeup 600000
bdm_autoreset off
bdm_setdelay 70
bdm_reset
set $sfc=5
set $dfc=5
```

The first line connects GDB to BDM driver. The second one disables caching of retrieved values in GDB (better for initial tests). Next lines select wait for memory and SIM initialization by on-board ROM monitor after reset, setting speed of BDM driver communication with the CPU32 and reset of target, which enables the background debug mode. Setting of SFC and DFC registers to 5 means, that next BDM accesses to the memory will be done in the supervisor privileged memory space mode.

If you have compiled target memory image with debug information (for example RT system absolute COFF image) you can start it by the following commands.

```
file sp01.exe
break main
run
```

You will be asked for download of image into the target and after your answer and successful start breakpoint in function main is reached and the GDB message and prompt appear. All GDB wonder is ready for you now!

7 Detailed GDB Invocation

Next paragraphs give expanded description of GDB initialization and first steps with BDM. It is mentioned for user with troubles and is based on my mail exchange with them.

Start GDB first. If it is compiled and installed right, GDB can be started from shell prompt by command "`m68k-bdm-coff-gdb`". Check, that BDM target is compiled in. List of included GDB targets is obtained by typing "`help target`" after GDB prompt "`(gdb)`". Result is something like next output for GDB configured for m68k-bdm-coff

```
List of target subcommands:

target bdm -- Debug with the Background Debug Mode
target cpu32bug -- Debug via the CPU32Bug monitor
target exec -- Use an executable file as a target
target extended-remote -- Use a remote computer via a serial line
target remote -- Use a remote computer via a serial line
```

There is some more explanation of command "`target`" before this list. You can use something like "`help target bdm`" for more help about specific target. BDM specific command starts with "`bdm_`" prefix. If you type "`bdm_`" without enter and press "Tab" key, GDB tries to complete command or to offer possible alternatives starting with this typed text . You will see

```
(gdb) bdm_
bdm_autoreset bdm_entry bdm_release bdm_status bdm_checkcable bdm_init
bdm_reset bdm_timetocomeup bdm_debug_driver bdm_log bdm_setdelay
(gdb) bdm_
```

More info about every command can be obtained after typing "`help <cmd_name>`".

You need to insert the BDM driver module into the Linux kernel to open connection to BDM target. Section 5 explains that. Some message should be seen in "/var/admsyslog". There may be problems, when BDM and printer drivers compete for same parallel port. Removing of "`lp`" module by "`rmmod lp`" can help in such case.

Next step is connect to the target board. You need the cable (PD or ICD) and 6833x (CPU32) board. I suggest to connect RS232 cable to board and PC first, then connect BDM cable. It protects BDM pins and PC printer port against damage (at least CPU32 BDM pins are very sensitive). You can type "`target bdm /dev/bdm`" (it expects link /dev/bdm pointing to used icd_bdm? or pd_bdm? device as suggested above). You should set speed of BDM communication by "`bdm_setdelay 75`" or more for beginning. If you have no problems later, you can set little delay. (I can use delay 1 on my K6 166 HX with onboard PP and on some more 586 and 486 configurations with above described ICD compatible cable.) If your 6833x board has onboard EPROM with initialization of chipselects, you can set "`bdm_timetocomeup 600000`" or to what necessary for initialize. If there is no onboard initialization then "`bdm_timetocomeup 0`" will be reasonable. "`bdm_reset`" should be entered for safety. You should be able to read accessible memory after that by "`x /100lx 0x1234`" command. If range is accessible and EPROM is mapped there, you should see its contents. If RAM is in tested range, you can try "`p *(int*)0x1234=0xAA55BB66`" then "`x /lx 0x1234`" should confirm that value was stored and read again. If range is not mapped to some internal chipselect and external circuitry does not acknowledge access, message "`Error accessing memory address 0x1234 : Unknown error 616`" or similar should appear. You need not to have loaded any program for this test.

Driver and BDM library (part of GDB) have full logging facilities for finding of problems with both software and hardware. Two logging commands can be entered at GDB prompt. The first one is "`bdm_log {on|off}`". Most of GDB-BDM driver activity is logged into file "`bdm-dbg.log`" in current directory, when "`bdm_log`" is on. Second command "`bdm_debug_driver` <level>" is designed for control of the kernel BDM driver logging. Level 0 mean no logging, 1 basic logging and 2 log all driver activity. Driver logging messages are processed by `klogd` and `syslogd` and in most cases appears in the "`/var/log/syslog`" file.

Command for GDB and BDM initialization can be stored into file. You can invoke interpretation of this file by "`source <filename>`" from GDB prompt. If script is stored under name ".gdbinit68" (or ".gdbinit" - depends on configuration of GDB) in current directory, then it is invoked at every GDB start.

8 Chipselects Initialization

You should read this section, if you have 68332 board without onboard chipselect initialization. It can be useful to read next if you want to program your own CPU32 initialization in "C" source code. I expect, that everybody seriously interested in 68332 have downloaded PDF or printed copy of [2] and [3]. Examples are written such way, as they can be used in GDB script file or typed after GDB prompt. Another way will be described later at end of this section.

Essential system control registers must be initialized first. The first line in every example is register address and its name. The second and third line contains definition of bitfields in register. Next is used value in binary notation and last is GDB syntax.

```
#0xFFFA00 - SIMCR - SIM Configuration Register
# 15    14    13    12 11     10 9..8 7    6  5 4 3     0
# EXOFF FRZSW FRZBM 0  SLVEN  0  SHEN SUPV MM 0 0 IARB
# 0     0     0     0  DATA11 0  0  0 1    1  0 0 1 1 1 1
set *(short *)0xfffa00=0x42cf
```

Disable watchdog

```
# 0xFFFA21 - SYPCR - System Protection Control Register
# 7   6      5.4 3   2   1 0
# SWE SWP    SWT HME BME BMT
# 1   MODCLK 0 0 0   0   0 0
set *(char *)0xfffa21=0x06
```

We have watchdog disabled above, so no need to care about it.

```
# 0xYFFA27 - SWSR - Software Service Register
# write 0x55 0xAA for watchdog
```

Used source and frequency must be configured. Next example select PLL generated 16.7 MHz system clock from 32.768 kHz crystal connected to 68332.

```
# 0xFFFA04 - SYNCR Clock Synthesizer Control Register
# 15 14 13........8 7    6 5 4     3     2     1     0
# W  X  Y           EDIV 0 0 SLIMP SLOCK RSTEN STSIM STEXT
# 0  0  1 1 1 1 1 1 0    0 0 U     U     0     0     0
set *(short *)0xfffa04=0x7f00
```

Decide, which chipselect outputs will be used and which are left for another functions. The third line describes which signal of data bus at RESET time presets configuration bit. Fifth is chipselect signal and six are alternative functions.

```
# 0xYFFA44 - CSPAR0 - Chip Select Pin Assignment Register 0
# 15 14 13.12    11.10    9.8      7.6      5.4      3.2      1.0
# 0  0  CSPA0[6] CSPA0[5] CSPA0[4] CSPA0[3] CSPA0[2] CSPA0[1] CSBOOT
# 0  0  DATA2 1  DATA2 1  DATA2 1  DATA1 1  DATA1 1  DATA1 1  1 DATA0
#       CS5      CS4      CS3      CS2      CS1      CS0      CSBOOT
#       FC2 PC2  FC1 PC1  FC0 PC0  BGACK    BG       BR
#
# 00 Discrete Output
# 01 Alternate Function
# 10 Chip Select (8-Bit Port)
# 11 Chip Select (16-Bit Port)
#
set *(short *)0xfffa44=0x3fff
```

```
# 0xFFFA46 - CSPAR1 - Chip Select Pin Assignment Register 1
# 15 14 13 12 11 10 9.8      7.6      5.4      3.2      1.0
# 0  0  0  0  0  0  CSPA1[4] CSPA1[3] CSPA1[2] CSPA1[1] CSPA1[0]
# 0  0  0  0  0  0  DATA7 1  DATA76 1 DATA75 1 DATA74 1 DATA73 1
#                   CS10     CS 9     CS8      CS7      CS6
#                   A23 ECLK A22 PC6  A21 PC5  A20 PC4  A19 PC3
#
set *(short *)0xfffa46=0x03ff
```

It is time to configure chipselect address range and access type. Ranges are divided between three 64 kB RAM regions (organized 32 kB $\times$ 16) and one 128 kB EPROM block (organized 64 kB $\times$ 16). There is one chipselect for read enable of both bytes of every RAM region. There are two chipselect for write per region. One enables write to odd bytes second to even bytes of RAM. It is necessary to define speed of access, refer to [2]. Chipselect can be used to acknowledge and autovector IRQ requests too.

```
#
# Chip selects configuration
#
# 0xFFFA48 - CSBARBT - Chip-Select Base Address Register Boot ROM
# 0xFFFA4C..0xFFFA74 - CSBAR[10:0] - Chip-Select Base Address Registers
# 15  14  13  12  11  10  9   8   7   6   5   4   3   2   0
# A23 A22 A21 A20 A19 A18 A17 A16 A15 A14 A13 A12 A11 BLKSZ
# reset 0x0003 for CSBARBT and 0x0000 for CSBAR[10:0]
#
# BLKSZ Size Address Lines Compared
# 000   2k   ADDR[23:11]
# 001   8k   ADDR[23:13]
# 010   16k  ADDR[23:14]
# 011   64k  ADDR[23:16]
# 100   128k ADDR[23:17]
# 101   256k ADDR[23:18]
# 110   512k ADDR[23:19]
# 111   1M   ADDR[23:20]
#
#
# 0xFFFA4A - CSORBT - Chip-Select Option Register Boot ROM
# 0xFFFA4E..0xFFFA76 - CSOR[10:0] - Chip-Select Option Registers
# 15   14 13  12 11 10   9       6  5  4  3 1  0
# MODE BYTE   R/W   STRB DSACK      SPACE IPL  AVEC
# 0    1  1   1  1  0    1 1 0 1 1  1  0  0 0  0  - for CSORBT
#
# BYTE  00 Disable, 01 Lower Byte, 10 Upper Byte, 11 Both Bytes
# R/W   00 Reserved,01 Read Only,  10 Write Only, 11 Read/Write
# SPACE 00 CPU,     01 User,       10 Supervisor, 11 Supervisor/User
#
set *(long *)0xfffa48=0x0e0468b0
# BOOT ROM 0x0e0000 128k  RO UL

set *(long *)0xfffa4c=0x0003503e
# CS0  RAM 0x000000 64k   WR U

set *(long *)0xfffa50=0x0003303e
```

```
# CS1  RAM 0x000000 64k   WR L

set *(long *)0xfffa54=0x0003683e
# CS2  RAM 0x000000 64k   RO UL

set *(long *)0xfffa58=0x02036870
# CS3  RAM 0x020000 64k   RO UL

set *(long *)0xfffa5C=0xfff8680f
# CS4

set *(long *)0xfffa60=0xffe8783f
# CS5

set *(long *)0xfffa64=0x02033030
# CS6  RAM 0x020000 64k   WR L

set *(long *)0xfffa68=0x02035030
# CS7  RAM 0x020000 64k   WR U
set *(long *)0xfffa6c=0x01036870
# CS8  RAM 0x010000 64k   RO UL

set *(long *)0xfffa70=0x01033030
# CS9  RAM 0x010000 64k   WR L

set *(long *)0xfffa74=0x01035030
# CS10 RAM 0x010000 64k   WR U
```

The second way is to setup chipselects from special macro files. They are processed through reset or download to the target. If "`bdm_autoreset`" is set, then reset take place before every "`run`" command too. The files must be in same directory as your executable and should have names "`myexec.bdmmb`", "`myexec.bdmme`" and "`cpu32init`", where `myexec` is name of debugged program. The first of macro files is processed before download, the second one after download of executable and third one at every reset of target. The syntax of the macro file is:

```
<cmd-letter> <num1> <num2> <num3>
```

The nums are either in hex (form 0x), in decimal (form 123) or in octal (form 0234)

The meaning of the nums depends on the command letter:

w or W means: (write)

write to address (num1) contents (num2) length is (num3) bytes. Only 1, 2, 4 bytes are permitted.

z or Z means: (zap, zero)

fill memory area beginning at (num1) with byte value (num2) length (num3) bytes.

c or C means: (copy)

copy memory area from (num1) to (num2) with length (num3) bytes.

Empty lines and lines with a leading '#' are ignored. See [1] for more information.

9 Utility BDM-Load

This utility is mainly designed for programming of FLASH memories mapped in address space of CPU32. It can be used for fast and simple loading and starting programs in RAM too. Original authors are G. Magin and D. Jeff Dionne. There is presented rewritten preliminary version by Pavel Pisa. The new version can program more than one FLASH memory mapped in address space and is able to autodetect type of memory and can take start address of FLASH regions from 683xx chipselect registers.

Utility uses the BFD object files management library and can load any format configured into the BFD library (iHEX, S-record, coff-m68k, elf-m68k etc.). The current version uses recent version of "bdmlib" library and it will be possible to integrate flash functions directly into GDB.

Next packages and files are needed to compile and install "bdm-load":

- mc683xx target system with BDM interface
- GCC compiler for your Linux system.
- BDM driver - latest version is attached into "bdm-load" archive
- BFD object file handling library configured for m68k
- some time and experience with Linux system and make utility

More informations about building "bdm-load" can be found in an associated README file.

The "bdm-load" accepts command line parameters. If there is no error in parsing of the parameters it processes requested actions and switches into interactive mode. Switching into interactive mode can be suppressed by the "–script" parameter. Description of command line interface follows.

```
Usage bdm-load [OPTIONS] file1 ...
-h --help - this help information!
-i --init-file=FILE - object file to initialize processor
-r --reset - reset target CPU before other operations
-c --check - check flash setup (needed for auto)
-e --erase - erase flash
-b --blankck - verify erase of flash
-l --load - download listed filenames
-g --go - run target CPU from entry address
-s --script - do actions and return
-d --bdm-delay=d - sets BDM driver speed/delay
-f --flash=TYPE@ADR - select flash
```

Choice for flash TYPE@ADR can be {<TYPE>|auto}[@{csboot|cs<x>|<start>{+<size>|-<end>}}] Examples auto@csboot amd29f400@0x800000-0x87ffff auto@0+0x80000 If auto type or cs address is used, check must be specified to take effect. Present known flash types/algorithms are amd29f040, amd29f400, amd29f800 and amd29f010x2 (two amd29f010 in parallel configuration). There should be no problem to add more algorithms in future.

Possible commands in interactive mode :

run starts CPU32 execution at address taken from last downloaded object file. If no file is loaded, it starts at address fetched during last reset command.

reset resets CPU32 and if no entry address is defined, PC address is remembered

erase sets erase request and starts erase procedure

blankck check all flash regions for unerased bytes

load [<object-file>] sets load request and starts download of files from object file list. if <object-file> is specified, object file list is cleared and specified file is added on-to list

exit/quit exit interactive mode and return to shell

dump <address> <bytes> dumps memory contents from specified address. bytes specifies number of bytes to print.

stat shows CPU32 state, does not require CPU32 to stop

check checks flash memories at specified ranges, if auto type or "cs" address is specified for some flash, CS address is fetched and flash autodetection is run

autoset same as check, but all flash types are revalidated

stop stops CPU32 and clears all reset, erase, load and run requests

make make in current directory is called

The simplest way to initialize CPU32 chipselect subsystem and other SIM parameters is to provide "cpu32init" file in same directory as "bdm-load" is started from. The "cpu32init" file is processed at every reset of target. The syntax of the macro file is described in last paragraphs of section 8.

10 Comparision of Different BDM Interfaces

There can be found more different types of BDM interfaces on many of Motorola MCUs except the CPU32 BDM described above.

The HC12 family of microcontrollers uses the one wire open collector BDM interface. It has very curious timing which enables bidirectional asynchronous bit transfer over one wire. It can be connected to RS-232 port through simple converter. BDM mode instruction are not executed by special HC12 microcode, but program counter of the regular HC12 CPU is vectored into special on chip ROM code.

The HC16 uses the BDM interface more similar to CPU32 one.

ColdFire MCUs have BDM mode implemented similar way as CPU32. The BDM cable is simplified, because of the single-step flip-flop and some logic are integrated directly into ColdFire MCUs. BDM command set is enhanced with real time monitoring functions. These functions and target memory access can be performed in parallel to running ColdFire CPU.

PowerPC MCUs have BDM interface too. I lack more information about it.

The BDM drivers and cables combinations available for GDB and Linux operating system are summarized in table 5.

Cable	MCU Type		
	CPU32(6833x)	CPU32+(68360)	ColdFire
PD (public Domain)	Gunter's driver [1][11]	Eric's driver [12]	-
ICD32 (P&E)	Gunter's driver [1][11]	*	-
ColdFire ICD	-	-	Eric's driver [12]

Table 5: Drivers for CPU32(+) and ColdFire MCUs

Port of BDM driver for Window NT can be found on page [14].

References

[1] Documentation and Release notes for the bdm-gdb-patches gdb-4.13 and gdb-4.16, Gunter Magin 20.04.95 and 20.07.96
`ftp://ftp.lpr.e-technik.tu-muenchen.de/pub/bdm/`

[2] M68300 Family MC68332 User's Manual, MOTOROLA, INC. 1995
`http://www.mot-sps.com/`

[3] CPU32 REFERENCE MANUAL MOTOROLA, INC., 1990, 1996

[4] TPU TIME PROCESSOR UNIT REFERENCE MANUAL, MOTOROLA, INC., 1996

[5] User's Manual : MC68376, MOTOROLA, INC. 1998

[6] MCF5206 USER'S MANUAL MOTOROLA, INC., pre-final version

[7] RCPU RISC CENTRAL PROCESSING UNIT REFERENCE MANUAL, MOTOROLA, INC. 1994, 1996

[8] Great Microprocessors of the Past and Present (V 10.1.1),
John Bayko (Tau), March 1998,
`http://www.cs.uregina.ca/~bayko/`

[9] Frequently Asked Questions FAQ, For the Internet USENET newsgroup: comp.sys.m68k, Robert Boys, Ontario, CANADA, August 24, 1995, Version 19,
`http://www.ee.ualberta.ca/archive/m68kfaq.html`

[10] Debugging with GDB The GNU Source-Level Debugger Fifth Edition,
for GDB version 4.17 , Richard M. Stallman and Roland H. Pesch, April 1998
Copyright (C) 1988-1998 Free Software Foundation, Inc
Free Software Foundation 59 Temple Place - Suite 330, Boston, MA 02111-1307 USA
ISBN 1-882114-11-6

[11] Location of Gunter Magin's Linux BDM driver for CPU32 modified by Pavel Pisa
`http://cmp.felk.cvut.cz/~pisa/m683xx/gdb-4.18-bdm-patches-pi1.tar.gz`

[12] Location of Eric Norum's and Chris John's Linux BDM driver for CPU32+ and ColdFire:
`ftp://skatter.usask.ca/pub/eric/BDM-Linux-gdb/`

[13] Motorola ColdFire Development Resources :
`http://fiddes.net/coldfire/`

[14] Page for Peter Shoebridge's version of BDM driver for Windows NT with ColdFire and CPU32 support:
`http://www.zeecube.com/bdm.htm`

[15] This document in PDF (Portable Document Format) is also available
`http://cmp.felk.cvut.cz/~pisa/m683xx/bdm_driver.pdf`

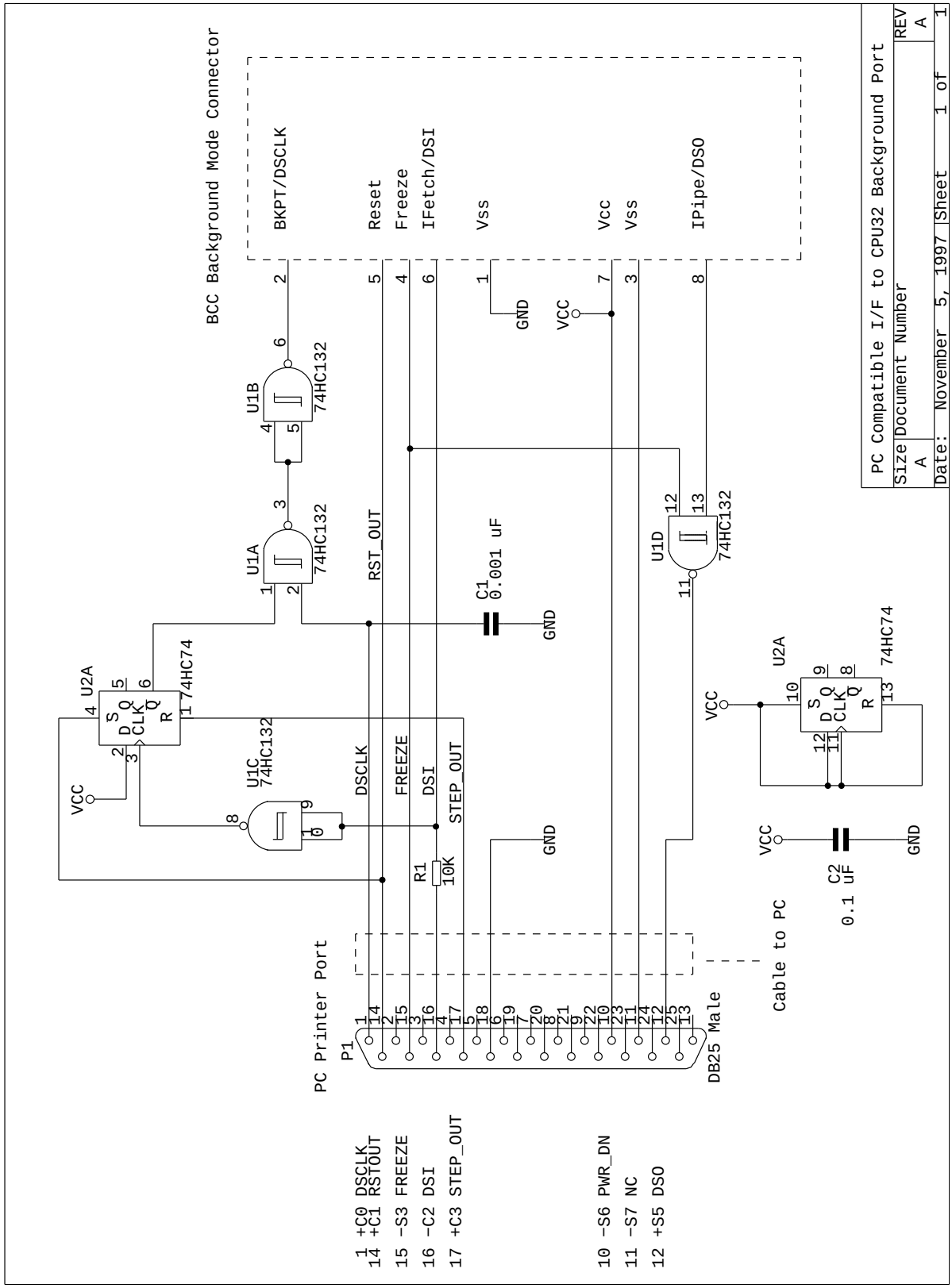
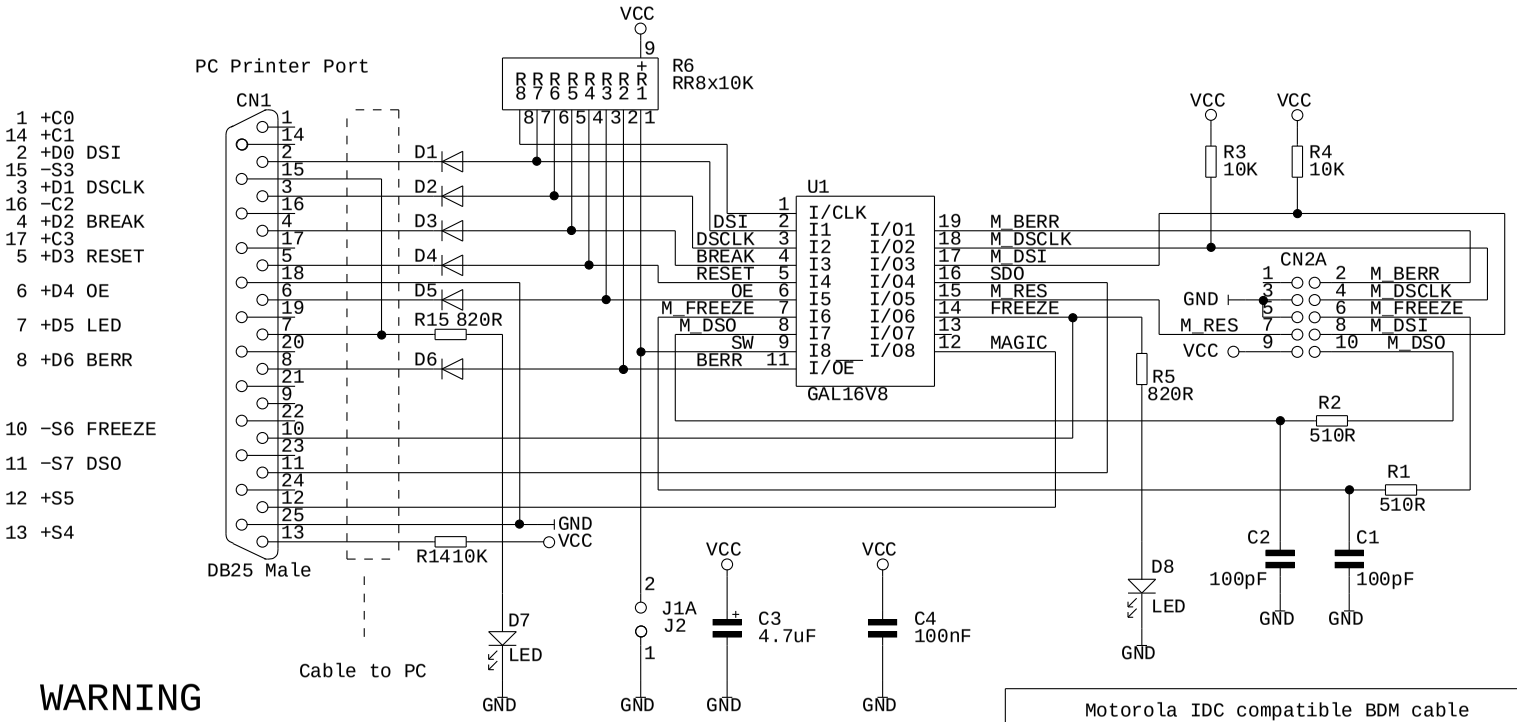


Figure 3: Possible BDM_PD Implementation

OPAL file definition

```
begin definition
  device GAL16V8
  inputs
    DSI=2,DSCLK=3,BREAK=4,RESET=5,OE=6,M_FREEZE=7,M_DSO=8,SW=9,BERR=11;
  outputs (com)
    M_BERR=19,M_DSCLK=18,DSO=16,FREEZE=14,MAGIC=12;
  feedback (com)
    M_DSI=17,M_RESET=15,FF_BREAK=13;
end definition
begin equations
  M_DSI=DSI;      M_DSI.oe=M_FREEZE&OE;
  M_DSCLK=DSCLK&FF_BREAK&M_RESET;
  FF_BREAK=BREAK|(FF_BREAK&(M_FREEZE|M_DSI));
  M_RESET=RESET;  M_RESET.oe=/RESET;
  M_BERR=/BERR;   M_BERR.oe=BERR;
  FREEZE=M_FREEZE;DSO=M_DSO;      MAGIC=0;
end equations
```



WARNING

This is not oficial schematic and GAL definition.
GAL definition is my own, I have not had any original ICD cable.

Motorola IDC compatible BDM cable		
Size A	Document Number	REV A
Date: September 18, 1999	Sheet 1 of 1	

Figure 4: ICD32 Compatible Cable