This is the debug monitor which has been in use at TGI since the late 80-s, with incarnations for the 68020, 68340, MPC8240 and now the MPC5200.

While much of it is still what was the very first code I wrote for a 68020, it has been doing a good job over the years and can be quite useful for debugging/bringing to life an MPC5200 based design without the need for any other debug tools.

The monitor is copyright by Transgalactic Instruments, www.tgi-sci.com.

The contact person is Dimiter Popoff <**dp@tgi-sci.com**>

```
*PPC r
R0-FFFBFFFF  R8/D0-  FFFFFFFF  R16/A0-03FFE89D  R24-03FFB474  PC(26)-  03FFA344
R1-F3FFFFFE  R9/D1-  80110C00  R17/A1-000014FC  R25-03FFB5AC  MSR(27)-00088032
R2-EBFEBF7F  R10/D2-00000010  R18/A2-8000AA80  R26-333DFFF7  LR(8)-   03FFA344
R3-FFF7F7FF  R11/D3-000064EE  R19/A3-80002C00  R27-7FF7FDFF  CTR(9)-  00000000
R4-00010000  R12/D4-00000000  R20/A4-00000000  R28-FFDFB7BF  XER(1)-  00000000
R5-80000B00  R13/D5-FFFFF7FF  R21/A5-04000000  R29-FFFFFBFF  CR-      40020000
R6-03FFA348  R14/D6-FEFBFFFF  R22/A6-03FF8B20  R30-FFFFFFFF  [CCR]-nzvc
R7-03FF8B20  R15/D7-04000000  R23/A7-03FF8B20  R31-00000000
*PPC p
cr-        40020000  xer(1)-    00000000  lr(8)-      03FFA344  ctr(9)-      00000000
fpscr-     00000000  hid0(1008)- 8051C000  hid1(1009)-  10000000  hid2(1011)-  00000000
dmiss(976)- 00000000  dcmp(977)-  00000000  hash1(978)-  00000000  hash2(979)-  0000FFC0
imiss(980)- 00000000  icmp(981)-  00000000  rpa(982)-   00000000  sdr1(25)-    00000000
sprg0(272)- 03FF9F84  sprg1(273)- 03FF8B20  sprg2(274)-  00000000  sprg3(275)-  00000000
srr0(26)-   03FFA344  srr1(27)-   00088032  dar(19)-     00000000  dsisr(18)-   00000000
dec(22)-    FFFFFF21  iabr(1010)- 00000000  ear(282)-    00000000  pvr(287)-    80822011
ibat0u(528)-00000FFF  ibat0l(529)-00000012  ibat1u(530)-80000000  ibat1l(531)-80000012
ibat2u(532)-0800003F  ibat2l(533)-08000012  ibat3u(534)-08200000  ibat3l(535)-08200012
dbat0u(536)-00000FFF  dbat0l(537)-00000012  dbat1u(538)-80000003  dbat1l(539)-8000003A
dbat2u(540)-0800003F  dbat2l(541)-08000012  dbat3u(542)-08200003  dbat3l(543)-08200012
sr0-521F8044  sr3-421F8046  sr6-134F8046  sr9- 421F8441  sr12-5317A146  sr15-031F8144
sr1-531F8244  sr4-527FA142  sr7-52170044  sr10-531F8166  sr13-531F8144
sr2-503E9044  sr5-529FC044  sr8-521B8254  sr11-531F8046  sr14-521FE044
tbu(285)-00000000  tbl(284)-001554BB
*PPC f
f0-FF7FF55BDFFDFFFF  f8- FEF7F6FEF5FDFEBF  f16-DDF8FDFFFFFFFFBF  f24-BFFFEFF6CFEFFEEF
f1-F3B6FBB75BFF6720  f9- EBE7FFDFFFFEF7FB  f17-FFFFF7FFEF6DCFBF  f25-CFEFF5EFFF5E1F9F
f2-EFF79F7FFFB57FDF  f10-FFBFBFDEFFFEFF7F  f18-DFEFFF7FEFFFB7E7  f26-FFFF7FBCFFFFE7DF
f3-2BFFEDFDBBFFBFFB  f11-EFFBEFFE9FFFDFDD  f19-FFFCDDFFEEFFCFFB  f27-EFD3FBF7EFE9BF7E
f4-FFF8000000000000  f12-FE7FFF7FFF7F63FD  f20-FFBFFFFFFEFFFFBFF  f28-7FD7FFFFF67C7FFB
f5-77FB7CBBCFEDEAF9  f13-FEB3DB7EBEFEEFBF  f21-ECFFFFDFB77FFBBD  f29-CFBFFFFFB7DFFE5F
f6-F97F4EFFFFFFF7EF  f14-FFFFEFFBFBDFFFBF  f22-FFFBFFFFBEFF9FFF  f30-FF3FFFEBFFBDDEFD
f7-77DE3DFFEBF7BFFF  f15-DFFFF7B7FFDEDF3F  f23-FCB9F317FFFFECC0  f31-FF9E9F6D7F5DFC75
*PPC prnt BEG 00000000  END 0000FFFF 2f
          0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
00000000 D6  EE  26  9E  94  FB  A2  EF  5F  59  0F  F7  EF  FF  BB  DF  ..&......_Y......
00000010 14  AE  7A  EC  88  2F  A2  B2  6F  DD  B7  DD  6A  7F  F5  FD  ..z../..o...j..
00000020 9E  AF  AF  AA  BF  23  5E  B7  E9  FB  5E  5C  EF  5A  5A  7A  .....#^...^\.ZZz
*PPC prnt BEG 00000000  10 END 0000002F 3f
00000010 14  AE  7A  EC  88  2F  A2  B2  6F  DD  B7  DD  6A  7F  F5  FD  ..z../..o...j..
00000020 9E  AF  AF  AA  BF  23  5E  B7  E9  FB  5E  5C  EF  5A  5A  7A  .....#^...^\.ZZz
00000030 BF  6F  B0  EF  FC  DF  A9  BF  EB  9E  75  FB  E3  DB  D3  FE  .o........u.....
*PPC d0 FFFFFFFF  12345678
*PPC d0 12345678
*PPC
```

`quit` ← `paste` ↑ `setup` ↓ `send` → `receive` ↵ `log_on` ⏎ `log_off`

1. General

This monitor version is for the MPC5200.

In order to run the monitor, it has to be programmed into the MPC5200 flash memory at offset $100, and the flash must appear at $0 at boot time. The monitor occupies somewhat less than 40 kilobytes, and allows the user to do memory read/write, step-by-step program execution, program breakpoints (for code in RAM only), load/save memory images in S-record and TGI local binary formats, and some more.

The monitor will use some of the serial ports for I/O, which one can be configured as explained below. The default port is PSC_6, the rest have not been tested at the time of writing this text.

Depending on configuration, the monitor will run either off-flash using the on chip SRAM only or will initialize the DDRAM and move itself there. The former is useful when testing newly built hardware and perhaps debugging the DDRAM initialization code in on-chip SRAM; the latter allows full-featured monitor use.

2. Reset and Initialization

2.1. FLASH/onchip SRAM vs. DDRAM reset options

The monitor comes out of reset off the flash (flash being at $0), and initially uses the on-chip SRAM at $80008000; it takes the area from $8000AB9C to $8000BFFF. The user "stack pointer" - a7/r23 - is set to the beginning of the above mentioned monitor RAM area, in this case $8000AB9C.

After some basic initializations the monitor will examine the .l at "raminit" ($188). If it finds there either $ffffffff or 0, it will skip all DDRAM related operations and move to the next step running off flash using the onchip SRAM monitor area.

If it finds something other than 0 or $ffffffff at $188, it will call this address with the return address in the LR.

The code there must return:

a5 - address where RAM is located upon return, d7 - RAM length, a4 - address where RAM will eventually be put (typically 0).

Notice that at this stage the RAM may not overlap with the flash area at 0, this is where the currently running code is being fetched. For a 64M DDRAM, a reasonable choice would be

to intially put the RAM at $4000000 (returning this in a5), later being moved to 0 (returning 0 in a4); the length would be $4000000 (64 M), thus d7 will return $4000000.

[The default initialization code contained at $36B0 will do exactly this. It assumes two x16 DDRAMs, Microns MT46V16M16, sets their output drive to "low", and a DDR clock frequency of 132 MHz - assuming 400 MHz core clock, 132 MHz IPB clock].

Notice that running off the flash/onchip SRAM initially allows the user to download DDRAM initialization code in the onchip SRAM (say, at $80008000) and debug it there. Once ready, it can be incorporated in flash using the .l at "raminit" as described above.

If an address was found at "raminit", after initializing the DDRAM, the monitor will move itself to the top of this RAM, and will move its data section just below the lowest address it occupies there, updating the "monvars" at $180.

[In the above example (64 M RAM) it will take the range from $3FF8B20 to $3FFFFFF - this includes the monitor vars at the beginning; the first $1400 bytes of the flash, which used to occupy the vector area $100 to $14ff, are not copied as they are useless.]

2.2   I/O initialization after reset

The next step - whether the monitor did initialize the DDRAM and moved there or did nothing and continued to run off flash - is to examine the .l at "ioinit", $18C. If 0 ot $ffffffff are found there, the default I/O initialization is done, using the .l at "ioport" ($194) as the PSC_x port address (it must point to one of the PSC register sets), and the address at "ioitab" ($190) as the initialization table to use with the default, embedded I/O initialization routine.

The table consists of entries 4 .l (16 bytes0 long each, formatted as following:

$0 - initialization opcode,
$4 - offset to register or register absolute address (opcode dependent),
$8 - bit mask,
$C - data to write to the register.
Valid opcodes are:

$0 - .l register, relative address,
$2 - .w register, relative address,
$4 - .b register, relative address,
$6 - .l register, absolute address,
$8 - .w register, absolute address,
$a - .b register, absolute address,
$ffffffff - end of table (entry must not be completed, just the code is read).
Relative addresses are used by the embedded I/O init code as offsets relative the the address found at "ioport" ($194). If the bit mask portion is other than $FFFFFFFF (all ones), the register is read, AND-ed with the mask, the result is then OR-ed with the "data to write" portion, this being the final result written to the register.

If the bit mask portion is all ones, ($FFFFFFFF), the data portion is written to the register without reading it at all (i.e. the register is treated as a write-only register).

For word sized operations, the mask and data portions occupy the lowest 16 bits of the portion (bytes 2 and 3).

For byte sized operations, the mask and data portions occupy the lowest 8 bits of the portion (byte 3).

Masked bits (those which are set to 1 in the mask portion) must be set to 0 in the data portion (this will keep the respective register bits unchanged).

After the I/O initialization has been completed, the monitor will execute an initial illegal opcode and the program exception will set all the user registers as found at this moment in the CPU (this is invisible to the user). Then, the monitor will anounce itself with a message and will begin to expect user commands (prompt being "*PPC ").

3. User commands

Commands are recognized case independent; so are hex numbers.

All numbers are assummed hex.

Hold and break sensitive processes (such as listing memory range etc.) respond to ctrl-w for hold and ctrl-c for break.

The command/data input line edit recognizes ASCII RUBOUT and BS ($7F, $08) to delete the last entered character, as well as CAN (ctrl-x) to delete the entire input buffer.

3.1. Memory edit/view/search commands.

3.1.1. M - Memory byte read/change.

Syntax: M [address]

If address is not specified, the last address used before is taken. The address is displayed as hex along with the byte contents; at this point the user can enter a hex value which will be written to the address. After writing the address contents will be read again and compared to the jsut written value; if the two do not match, the read value will be displayed preceeded by a question mark.

The command can be terminated by a CR ($0D, CTRL-M), LF ($0A, CTRL-J) , dot (".", $2E) or VT ($0B, CTRL-K) character.

If terminated by a CR, the command will exit to the main monitor prompt.

If terminated by an LF, the next address will be opened on a new line.

If terminated by a dot, the next address will be opened at the same line, displaying the data only, not the address. If terminated by a VT, the previous address will be opened at a new line.

3.1.2. W - memory Word examine/change.

Syntax: W [address]

Same as 3.1.1 M command, only reads/writes/does address inc/dec in words (2 bytes) rather than a byte.

3.1.3. L - memory Long word examine/change.

Syntax: L [address]

Same as 3.1.1 M command, only reads/writes/does address inc/dec in long words (4 bytes) rather than a byte.

3.1.4. I - Insert data into memory range.

Syntax: I [long word to insert].

The command prompts for BEG and END address (offering the last BEG and END address which have been entered), then writes the long word (32 bits) to the specified range.

If no data has been specified (just I <CR>), the region will be written with 0-s.

Neither the BEG nor the END address need be aligned. The data will be written using BYTE memory accesses only.

3.1.5. S - Search memory.

Syntax: S <long word to search>

The command searches the memory starting with the last entered

BEG and END addresses under control of the search mask (see

3.1.6 MASK command). This is done by reading 4 bytes at each address in the range (incrementing by 1), AND-ing the resulting long word with the mask and comparing this result to the and result of the value being searched for and the mask. [Thus if the mask is 0 any data will be found to match any address]
When a match is found it is displayed at a new line along with the address where it was found.

The process is hold/break sensitive.

3.1.6. MASK command - enter search MASK and range.

Syntax: MASK

The current mask value is displayed for editing. The user can either enter a new long word or leave the current value unchanged by typing just a CR. Then on a new line the BEG and END addresses are prompted in the same manner, offering the current values for change.

This mask along with the BEG/END range (notice it can be altered by many other commands using the BEG/END input) is used during memory search (see 3.1.5 S command).

3.1.7. O - Output data to memory command.

Syntax: O.<size> <address> <data>

This command writes data to a specific address without reading the address first (useful for writing to write-only registers which may not be read).

Size may be .b, .w or .l; the access is respectively byte, word or long word.

No misalignment precautions are taken, i.e. the access will be done exactly at the address which has been specified.

3.1.8. PRNT - PRiNT memory range command.

Syntax: PRNT

The command prompts for BEG and END address, offering the latest BEG and END values, after which it lists the memory range as hex and ASCII. The process is hold/break sensitive (CTRL-W for hold, CTRL-C for break).

3.1.9 LV - Long word memory byte reVersed examine/change.

Same as 3.1.3 L command, but uses byte reversed read/write to access the .l location.

3.2. User register commands.

3.2.1. R - display user gpR-s.

Syntax: R

Lists r0-r31 and some more. The list includes VPA register conventions, e.g. r8 is also shown to be D0, PC is shown (which is SRR0), along with the cr also the VPA equivalent CCR (68k) is shown etc.

3.2.2. F - display Floating point registers.

Lists f0 to f31. VPA conventions are not shown (which are FP0 to FP7 being f8 to f15).

3.2.3. P - display sPr-s.

Lists all the 603e special purpose registers.

3.2.4. Register change commands.

All user registers can be individually examined/modified. After entering the respective command - which is the register name, either the native PPC one or its VPA convention equivalent - the register contents will be displayed allowing a new value to be entered; a single CR will leave the register unchanged.

Here is a list of the accepted register commands:

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15
D0 D1 D2 D3 D4 D5 D6 D7 (these map to R8 - R15)
R16 R17 R18 R19 R20 R21 R22 R23 R24 R25 R26 R27 R28 R29 R30 R31
A0 A1 A2 A3 A4 A5 A6 A7 (these map to R16 - R23)
SRR0 PC (PC is just alternate syntax for SRR0)
MSR LR CTR CR XER FPSCR HID0 HID1 HID2
DMISS DCMP HASH1 HASH2 IMISS ICMP RPA SDR1
SPRG0 SPRG1 SPRG2 SPRG3 SRR0 SRR1 DAR DSISR
DEC IABR EAR PVR
IBAT0U IBAT0L IBAT1U IBAT1L IBAT2U IBAT2L IBAT3U IBAT3L
DBAT0U DBAT0L DBAT1U DBAT1L DBAT2U DBAT2L DBAT3U DBAT3L
SR0 SR1 SR2 SR3 SR4 SR5 SR6 SR7 SR8 SR9 SR11 SR12 SR13 SR14 SR15
TBU TBL
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15
F16 F17 F18 F19 F20 F21 F22 F23 F24 F25 F26 F27 F28 F29 F30 F31
These are the regiter at the point of writing this description.

Use the "P" command to list all the special purpose registers known by the monitor, their names on the list and as commands for editing are identical.

3.3. Load/save commands.

3.3.1. Load - load memory image.

Syntax: LOAD

Loads a memory image by a very simple binary protocol defined by TGI. Error detection/correction is symbolic, the protocol is meant for short distance connections such as an RS-232 cable not longer than a few metres.

After issuing the command (without an argument, it will recognize an argument and initiate a dialogue no longer supported and not published, nor very useful, and will hang), it will be waiting for an "S" ($53) character. At this stage, the process is still break/hold sensitive.

After the S is received, a 12-byte packet is expected as following:

bytes 0 to 3 - address to load the image to (MSB first),

bytes 4 to 7 - execution address (if inapplicable, set this equal to

the load address),

bytes 8 to 11 - byte count following this packet (i.e. memory image length).

After the packet is received, the following data are placed in the address range specified in the packet as they are received.

Throughout the entire process (including the packet but not including the "S" character) a checksum byte is calculated, its starting value being 0. Every received byte (including those in the 12-byte header) is just added to the checksum byte.

After filling the entire range, one more byte is input which should be the checksum of the data just received. It is compared to the calculated checksum; if they do not match, an error message is shown. If they match, the BEG, END and EXEc address are listed and the command is done.

3.3.2. Save - save memory image.

Syntax: SAVE

Outputs a memory image to the console interface in the same format as the LOAD commands (see 3.3.1).

After entering the command, the user is propmted for BEG/END addresses, after which the data are sent (precedins S, header, data and checksum).

3.3.3. LXLOAD - load an S-record formatted image.

Syntax: LOAD [offset].

The S-record is loaded at the addresses it specifies itself + the offset which was entered with the command. If no offset has been entered the offset during load is 0.

This command will only accept S0, S1 and S9 records. This limits the record size to 2^{16}; and if no offset is specified, the load range is in the first 64k of the memory map.

3.3.4 LXSAVE - save memory image as an S-record.

Syntax: LXSAVE

The user will be prompted to enter the address range to be saved with the current BEG/END values suggested as defaults. After that, the memory range will be encoded in S-records and sent to the console interface. Depending on address size, S1 (address within 2^{16}), S2 (address within 2^{24}), or S3 (address within 2^{32}) will be used.

3.4.   Go to address, Trace and breakpoint commands.

Single step tracing is done using the trace exception while breakpoints are set by replacing the respective opcodes by an illegal opcode (0). The illegal opcodes per breakpoint are installed only after a "G" command has been entered, while the original data at all breakpoints are restored as soon as a program exception takes place so the code modifications are transparent to the user. Obviously breakpoints work in RAM only, while trace works anywhere.

3.4.1 Go to address command.

Syntax: G [address]

All user register values are loaded into the respective CPU registers and control code execution is set to begin at the specified address. If no address is specified, the current PC (srr0) is taken.

3.4.2. N - run Next instruction.

Syntax: N [number of instructions to exectute]

Traces 1 or more (if some number is entered) instructions. The first instruction executed is the one currently pointed to by the user PC (srr0).

After each instruction is executed, the GP registers are listed like be the R command if the by convention SP (A7, r23) is within the "stack level", which is initialized upon reset to accomodate the entire 32 bit address range.

3.4.3. T - trace continuously.

Syntax: T

Same as N, but the number of traced instructions is unlimited.

Hold/break sensitive.

3.4.4. Set breakpoint(s)/display all breakpoints

Syntax: V [addresse(s)]

Sets a breakpoint at the specified address(es).
If no address is specified, all currently active breakpoints are listed. More than one address may be specified at the same line.

3.4.5. Delete breakpoint(s)

Syntax: K [address(es)]

Delete the breakpoint(s) at the specified address(es).
If no address is specified, delete all breakpoints.

4.   Exception handling and monitor service calls.

The monitor deals with the following exceptions:

reset
trace
program
system call.

The decrementer exception is initialized with an "rfi" opcode, it is used to take the CPU out of nap during input (it naps for 10 mS every time it has to wait for a character to be received if the PSC FIFO is empty).

Reset behaviour has been discussed in detail in 2., reset and initialization.

The trace exception is used when tracing step-by-step with the N or T commands.

The program exception is handled in one of two ways:

- breakpoint - if there is a breakpoint at the address which caused the program exception,
- program abort and exit to monitor with an "illegal opcode" message preceding the register list (which is the same as wehn stopping for a breakpoint or trace, that is, just the GPR list is produced).

The system call (SC) exception is used for monitor service calls. It is up to other software which uses the same exception (i.e. an OS like DPS etc.) to recognize the system call destination and if the call is for the monitor, to pass control to it. The monitor call is passed via r31 (and so are the system calls in DPS). Values 0 to $fff are reserved for the monitor, although the upper limit can be moved if necessary.

An sc handler can locate the monitor sc handler address at the hardware exception address ($C00) + $f0. The new sc handler can then process the exception, and after deciding it is a monitor service call it can jump to the monitor exception handler address, having restored all registers.

This is a scheme applicable to all (except for reset) exceptions (valid only after the monitor has moved itself to RAM, none of this can work off flash).

Here is the code which is copied to the sc exception handler upon reset:

```
00000C00                            *
00000C00                            r5sc:   ppv      scaex                sc
00000C00 0000 0B00                          set      ppv$ca,*
00000C00 4800 0CD6                           jmp      *-r5reset+$100+($100-44)
00000C00: 48000CD6  ba       *-r5reset+$100+($100-44)
00000C04 0000 0000 0000 0000           bc.b     $100-44-(*-ppv$ca),0
00000CD4                            *
00000CD4 9080 0CF8                           move.l-  r4,(ppv$ca-r5reset+$100+$f8).w save r4
00000CD4: 90800CF8  stw        r4,(ppv$ca-r5reset+$100+$f8),r0
00000CD8 7C88 02A6                           move.l-  lr,r4                then LR
00000CD8: 7C8802A6  mfspr      256,r4
00000CDC 9080 0CF4                           move.l-  r4,(ppv$ca-r5reset+$100+$f4).w  save LR
00000CDC: 90800CF4  stw        r4,(ppv$ca-r5reset+$100+$f4),r0
00000CE0 8080 0CF0                           move.l-  (ppv$ca-r5reset+$100+$f0).w,r4  address to go
00000CE0: 80800CF0  lwz        (ppv$ca-r5reset+$100+$f0),r0,r4
00000CE4 7C88 03A6                           move.l-  r4,lr                in LR
00000CE4: 7C8803A6  mtspr      r4,256
00000CE8 3880 0C00                           lea      (ppv$ca-r5reset+$100).w,r4      pass handler
addres
00000CE8: 38800C00  addi       (ppv$ca-r5reset+$100),r0,r4
00000CEC 4E9F 0020                           rtl                  *go there*
00000CEC: 4E9F0020  bclr       20,31
00000CF0 0000 0000                           dc.l     rscaex  address to go to
00000CF4 0000 0000 0000 0000           dc.l     0,0,0   last 2 used for temp. save, last
00000C00                     * but by handler to save r5 or something, resttore LR, then r5..
00000C00                            *
```

It preserves all registers except for the LR; it is up to the handler code to restore it from where it was saved (at offset f4 from the hardware exception address), typically using the reserved location at offset $FC to save/restore an intermediate register, r5 in the case of the monitor handlers.