Bite My ASM
Debugger Introduction

SRLM
`srlm@anzhelka.com`

May 2, 2012

Revisions

Version	Date	Changes	Commiter
0.01	May 1, 2012	Initial layout of file was created.	SRLM

Table of Contents

1 Introduction

This document details the BMA Debugger written by Jazzed.
Version 1.91 - Date 2011.07.24
http://forums.parallax.com/showthread.php?115068

2 Usage

User can drop PASM test code into a copy of this file below the PASM dat label. The control code is in a separate module that can be used in other applications.

It is necessary to have 8 nops at the beginning of the PASM label to have room for the debugger stub. If the debugger is not used, the code will still run. The statement "long 0 [8]" would make 8 nop to reserve debugger space. The debugger startup will copy the interface handler into the reserved space. Just follow the example given here, and things should work.

The COG debug interface handler is 8 longs and uses registers INB,OUTB,DIRB. This is less invasive than other debuggers on a per cog basis, but requires some Spin.

If you have problems with this tool (and it is possible that there are bugs), you can post the issue at http://forums.parallax.com/forums
. Search for BMADebugger.

3 Description

The idea of this debugger is to keep this code as small and simple as possible while keeping all resources on chip, but make it usable.

A Windows GUI is not necessary to use this debugger. Using a compiler like BSTC with good list output makes it easier to debug complex code.

A list of commands is available by typing "?" at the *Ok* > prompt

Running code realtime is possible with the "g" command. Once you start it, it will not stop unless a breakpoint has been set in the code path.

In single step mode, the program is an emulator because it uses the debug handler to run instructions one at a time. The Propeller executes most of the instructions, but there are special cases like JMP and DJNZ that need to be directed by the "interpreter" method. These cases when not handled properly would cause us to lose control of the code because the PASM could take over completely if a JMP and friends are not handled by the stepper.

keywordstyle
```
Starting BMA Debugger ...

BMA Debugger Started.
T0.PC 008 Ok>
T0.PC 008 .      : jmp     5C7C000F   N D:000  083C01F3 S\#00F           D=000  083C01F3
T0.PC 00F .      : or      68FFEC83     D:DIRA 00000000 S\#083
T0.PC 00F Ok>
T0.PC 00F .      : or      68FFEC83     D:DIRA 00000000 S\#083           D=DIRA 00000083
T0.PC 010 .      : mov     A0BC1BF0     D:00D  00000000 S:PAR  00001FA8
T0.PC 010 Ok>
T0.PC 010 .      : mov     A0BC1BF0     D:00D  00000000 S:PAR  00001FA8 D=00D  00001FA8
T0.PC 011 .      : mov     A0BC1DF0     D:00E  00000000 S:PAR  00001FA8
T0.PC 011 Ok>
T0.PC 011 .      : mov     A0BC1DF0     D:00E  00000000 S:PAR  00001FA8 D=00E  00001FA8
T0.PC 012 .      : add     80FC1C04     D:00E  00001FA8 S\#004
T0.PC 012 Ok>
T0.PC 012 .      : add     80FC1C04     D:00E  00001FA8 S\#004           D=00E  00001FAC
T0.PC 013 .      : rdlong  08BC180E     D:00C  00000000 S:00E  00001FAC
T0.PC 013 Ok>
T0.PC 013 .      : rdlong  08BC180E     D:00C  00000000 S:00E  00001FAC D=00C  00001FB0
T0.PC 014 .      : mov     A0BC160C     D:00B  00000000 S:00C  00001FB0
T0.PC 014 Ok>
T0.PC 014 .      : mov     A0BC160C     D:00B  00000000 S:00C  00001FB0 D=00B  00001FB0
T0.PC 015 .      : add     80FC1810     D:00C  00001FB0 S\#010
T0.PC 015 Ok>
T0.PC 015 .      : add     80FC1810     D:00C  00001FB0 S\#010           D=00C  00001FC0
T0.PC 016 .      : cmp     863C180B Z N D:00C  00001FC0 S:00B  00001FB0
T0.PC 016 Ok>
T0.PC 016 .      : cmp     863C180B Z N D:00C  00001FC0 S:00B  00001FB0 D=00C  00001FC0
T0.PC 017 .      : wrbyte  003C180C   N D:00C  00001FC0 S:00C  00001FC0
T0.PC 017 Ok>
```

4 List/Step explanations (by StefanL)

keywordstyle
```
(0)   (1)(2)       (3)        (4)          (5)   (6)      (7)   (8)       (9)    (10) (11)
T0.PC 00A Z      : mov     A0E85011     D:028 00000000 S\#011          D=028 00000000 C
T0.PC 013 .      : mov     A0BC5028     D:028 04C4B400 S:028 04C4B400 D=028 04C4B400 Z
```

original line of code in the propeller-tool

keywordstyle
```
  if_z mov     t0,      \#\$11
       mov     t0,      t0


 (0) Task number
 (1) program-counter
 (2) flag-condition                                             (at PC 00A "if_z"  at PC 013 ".")
 (3) mnemonic of the PASM-command                               (at PC 00A "mov"   at PC 013 "mov")
 (4) 32-bitvalue of long at PC                                  (at PC 00A "A0E85011")
 (5) Destination-adress of command BEFORE executing the command (at PC 00A "028")
 (6) Value of destination BEFORE executing the command          (at PC 013 "04C4B400")
 (7) Source_adress of the command                               (at PC 00A "S\#011" at PC 013 "028") [S\#011]
     means immediate value NO dest-adress
 (8) Value of Source                                            (at PC 013 "04C4B400"
 (9) Destination-adress of command AFTER executing the command  (at PC 013 "028")
(10) Value of destination AFTER command has executed            (at PC 00A no value as it is immediate at PC
    013 "04C4B400")
(11) status of C and Z-flag (if mentioned flag is SET           (at PC 00A "C" carry-flag is set at PC 013 Z-
    flag is set)
```

Syntax of user commands can be seen if you type ”?” at the ok prompt.

Here is a general description of commands. If there is a conflict between the syntax listed here and the "?" help command, use the command's output syntax.

```
   keywordstyle
T0.PC 01C Ok> ?
ax      : animate with x ms delay per step
bx      : toggles breakpoint at COG address x
c       : clears all breakpoints
D       : dumps all COG values
dx n    : dumps n COG values from x
ftx n v: fill n HUB addresses with v from x with t type = b,w,l
g       : run COG and stop at any breakpoints
htx n   : dumps n HUB values from x with t type = b,w,l
ix      : step showing result of instructions at x
L       : lists/disassembles all COG values/instructions
l       : lists/disassembles 16 instructions from PC
lx n    : list n instructions s starting at x
n       : step over jmpret and call
pr      : prints special register values PAR, etc...
px      : prints content of COG register number <hex>
r       : resets pc back to starting position
R       : restarts COG
sx v    : sets value at COG address x = v
tx      : switch to COG task x
z       : shows flags
Enter   : single-step
?       : show this help screen
```

4.1 ax : animate with x ms delay per step

Allows stepping through code at a given rate while displaying instruction and effects. Once a is pressed, you have half a second to specify the step delay and hit enter. If you just type a, the animation will run at full speed after a moment.

4.2 bx : toggles breakpoint at COG address x

User sets a breakpoint if it does not already exist with this command. If breakpoint exists, the command will clear the breakpoint. Using breakpoints allows user code to run in real-time to the breakpoint address. The "g" (go!) command will stop at any breakpoints encountered and the prompt will give the PC address. If the PC ¿ $1F0, it is likely that no breakpoints were hit and to regain control of the COG, the "R" command should be used to restart.

4.3 c : clears all breakpoints

If user has set any breakpoints, this command will clear them all.

4.4 D : dumps all COG values

User can dump the contents of all addresses and in the COG. The special registers $1f0-$1ff will not show the register value such as pin bit settings for OUTA; use the "pr" command for that. The dump will show the special register shadow values only. If any breakpoints have been set, addresses corresponding to breakpoints will contain the value $5C7C0001; to see the code listing with breakpoints, use the list commands.

4.5 dx n : dumps n COG values from x

User can dump the contents of an address and range from the COG with this command. This dump has the same constraints as the dump all command.

4.6 ftx n v: fill n HUB addresses with v from x with t type = b,w,l

User can fill the contents of HUB with this command. The "t" in the syntax should be replaced with the type identifier to specify the operation. If the type is "b", one or more bytes specified by "n" beginning at the "x" address will be set to the value "v". If the type is "w", word values will be used. If the type is "l", long values will be used.

4.7 g : run COG

This is the GO! command. User can run the COG's PASM in real-time with the "g" command. If not breakpoints are found, the debugger will lose control of the COG, and user should enter the "R" command to restart the COG and debugger as necessary. The "g" (go!) command will stop at any breakpoints encountered and the prompt will give the PC address. If the PC ¿ $1F0, it is likely that no breakpoints were hit and to regain control of the COG, the "R" command should be used to restart.

4.8 htx n : dumps n HUB values from x with t type = b,w,l

User can dump the contents of HUB with this command. The "t" in the syntax should be replaced with the type identifier to specify the operation. If the type is "b", one or more bytes specified by "n" beginning at the "x" address will be dumped to the screen. If the type is "w", word values will be used. If the type is "l", long values will be used. The type "b" also prints the ASCII representation of each address value if printable; a dot "." will show if the value is ¡ 20*or* >80.

4.9 ix : step showing result of instructions at x

This command causes the program to single-step until debugger detects a key hit. It will show the result of instruction executing at PC "x". This speeds up debugging.

```
   keywordstyle
PC 018 Ok>
i 16

Step-watch Instruction @ 016
PC 016 .      : cmp     863C180B Z N D:00C  00001FBD S:00B  00001FB0 D=00C  00001FBD
PC 016 .      : cmp     863C180B Z N D:00C  00001FBC S:00B  00001FB0 D=00C  00001FBC
PC 016 .      : cmp     863C180B Z N D:00C  00001FBB S:00B  00001FB0 D=00C  00001FBB
PC 016 .      : cmp     863C180B Z N D:00C  00001FBA S:00B  00001FB0 D=00C  00001FBA
PC 016 .      : cmp     863C180B Z N D:00C  00001FB9 S:00B  00001FB0 D=00C  00001FB9
PC 016 .      : cmp     863C180B Z N D:00C  00001FB8 S:00B  00001FB0 D=00C  00001FB8
PC 016 .      : cmp     863C180B Z N D:00C  00001FB7 S:00B  00001FB0 D=00C  00001FB7
PC 016 .      : cmp     863C180B Z N D:00C  00001FB6 S:00B  00001FB0 D=00C  00001FB6
PC 016 .      : cmp     863C180B Z N D:00C  00001FB5 S:00B  00001FB0 D=00C  00001FB5
PC 016 .      : cmp     863C180B Z N D:00C  00001FB4 S:00B  00001FB0 D=00C  00001FB4
PC 016 .      : cmp     863C180B Z N D:00C  00001FB3 S:00B  00001FB0 D=00C  00001FB3
PC 016 .      : cmp     863C180B Z N D:00C  00001FB2 S:00B  00001FB0 D=00C  00001FB2
PC 016 .      : cmp     863C180B Z N D:00C  00001FB1 S:00B  00001FB0 D=00C  00001FB1
PC 016 .      : cmp     863C180B Z N D:00C  00001FB0 S:00B  00001FB0 D=00C  00001FB0 Z
PC 016 .      : cmp     863C180B Z N D:00C  00001FC0 S:00B  00001FB0 D=00C  00001FC0
```

4.10 L : lists or disassembles all COG values/instructions

User can list all COG instructions with this command. The result portion of the stepper display is ommited. Also if breakpoints are set, the line will show "PC*addr" instead of "PC addr" to let you know a breakpoint is set at the address. Lines with breakpoints will also display the command to be executed instead of $5C7C0001.

4.11 l : list 16 instructions starting at current PC or last list position.

This is the same as "lx n" without arguments.

keywordstyle
```
PC 017 Ok>
l
PC 017 .      : wrbyte  003C180C   N D:00C  00001FBF S:00C  00001FBF
PC 018 NZ     : djnz    E4D41816     D:00C  00001FBF S\#016
PC 019 .      : xor     6CFFE801     D:OUTA 00000000 S\#001
PC 01A .      : test    637FE803 ZCN D:OUTA 00000000 S\#003
PC 01B Z      : mov     A0E81411     D:00A  00000000 S\#011
PC 01C C      : mov     A0F01422     D:00A  00000000 S\#022
PC 01D .      : xor     6CFFE801     D:OUTA 00000000 S\#001
PC 01E .      : test    637FE803 ZCN D:OUTA 00000000 S\#003
PC 01F Z      : mov     A0E81433     D:00A  00000000 S\#033
PC 020 C      : mov     A0F01444     D:00A  00000000 S\#044
PC 021 .      : rdlong  08FC1400     D:00A  00000000 S\#000
PC 022 .      : wrlong  087C1404   N D:00A  00000000 S\#004
PC 023 .      : rdlong  08FC1404     D:00A  00000000 S\#004
PC 024 .      : mov     A0BC140A     D:00A  00000000 S:00A  00000000
PC 025 .      : rdword  04FC1406     D:00A  00000000 S\#006
PC 026 .      : wrword  047C1404   N D:00A  00000000 S\#004
PC 017 Ok>
l
PC 027 .      : rdlong  08FC1404     D:00A  00000000 S\#004
PC 028 .      : mov     A0BC140A     D:00A  00000000 S:00A  00000000
PC 029 .      : rdbyte  00FC1406     D:00A  00000000 S\#006
PC 02A .      : wrbyte  007C1405   N D:00A  00000000 S\#005
PC 02B .      : mov     A0BC140A     D:00A  00000000 S:00A  00000000
PC 02C .      : rdlong  08FC1404     D:00A  00000000 S\#004
PC 02D .      : call    5CFC6E37     D:037  5C7C0000 S\#037
PC 02E .      : mov     A0BC1409     D:00A  00000000 S:009  00000003
PC 02F .      : sub     84FC1401     D:00A  00000000 S\#001
PC 030 .      : tjnz    E87C142F   N D:00A  00000000 S\#02F
PC 031 .      : sub     84FC1401     D:00A  00000000 S\#001
PC 032 .      : add     80FC1401     D:00A  00000000 S\#001
PC 033 .      : tjz     EC7C1432   N D:00A  00000000 S\#032
PC 034 .      : mov     A0BC1409     D:00A  00000000 S:009  00000003
PC 035 .      : djnz    E4FC1435     D:00A  00000000 S\#035
PC 036 .      : jmp     5C7C0013   N D:000  083C01F3 S\#013
PC 017 Ok>
```

4.12 lx n : list n instructions starting at x

User can list COG instructions beginning at x for n lines with this command. The display rules are the same with this command as with the "L" command.

keywordstyle
```
l 10 10

PC 010 .      : mov     A0BC1BF0     D:00D  00001FA8 S:PAR  00001FA8
PC 011 .      : mov     A0BC1DF0     D:00E  00001FAC S:PAR  00001FA8
PC 012 .      : add     80FC1C04     D:00E  00001FAC S\#004
PC 013 .      : rdlong  08BC180E     D:00C  00001FBF S:00E  00001FAC
PC 014 .      : mov     A0BC160C     D:00B  00001FB0 S:00C  00001FBF
PC 015 .      : add     80FC1810     D:00C  00001FBF S\#010
PC 016 .      : cmp     863C180B Z N D:00C  00001FBF S:00B  00001FB0
PC 017 .      : wrbyte  003C180C   N D:00C  00001FBF S:00C  00001FBF
PC 018 NZ     : djnz    E4D41816     D:00C  00001FBF S\#016
PC 019 .      : xor     6CFFE801     D:OUTA 00000000 S\#001
PC 01A .      : test    637FE803 ZCN D:OUTA 00000000 S\#003
PC 01B Z      : mov     A0E81411     D:00A  00000000 S\#011
PC 01C C      : mov     A0F01422     D:00A  00000000 S\#022
PC 01D .      : xor     6CFFE801     D:OUTA 00000000 S\#001
PC 01E .      : test    637FE803 ZCN D:OUTA 00000000 S\#003
PC 01F Z      : mov     A0E81433     D:00A  00000000 S\#033
PC 017 Ok>
```

4.13 n : step over jmpret and call

This command will single-step through a call or jmpret instruction to the "next" instruction after the "call return" to speed debugging.

keywordstyle

```
PC 02C Ok>
PC 02C .      : rdlong  08FC1404     D:00A  000000B8 S\#004           D=00A  05B8B8B8 Z
PC 02D .      : call    5CFC6E37     D:037  5C7C002E S\#037
PC 02D Ok>
PC 02D .      : call    5CFC6E37     D:037  5C7C002E S\#037          R D=037  5C7C002E Z
PC 02E .      : mov     A0BC1409     D:00A  05B8B8B8 S:009  00000003
PC 02E Ok>
```

4.14 pr : prints special register values PAR, etc...

User can print the values that the COG will see in the special registers with this command (except for INB,OUTB,DIRB which are used by the debugger). The output of this command is in the form "NAME ADDRESS HEX-VALUE BINARY-VALUE".

keywordstyle

```
pr
PAR  1F0 00001FA8 00000000000000000001111110101000
CNT  1F1 2DD8D3FD 00101101110110100110100000101101
INA  1F2 A0000000 11100000000000000000000000000000
INB  1F3 00007FF4 00000000000000000111111111110100
OUTA 1F4 00000000 00000000000000000000000000000000
OUTB 1F5 A3837FF8 10100011100000110111111111111000
DIRA 1F6 00000083 00000000000000000000000010000011
DIRB 1F7 00000083 00000000000000000000000010000011
CTRA 1F8 00000000 00000000000000000000000000000000
CTRB 1F9 00000000 00000000000000000000000000000000
FRQA 1FA 00000000 00000000000000000000000000000000
FRQB 1FB 00000000 00000000000000000000000000000000
PHSA 1FC 00000000 00000000000000000000000000000000
PHSB 1FD 00000000 00000000000000000000000000000000
VCFG 1FE 00000000 00000000000000000000000000000000
VSCL 1FF 00000000 00000000000000000000000000000000
```

4.15 px : prints content of COG register number ¡hex¿

User can examine the value of one COG register with this command. If the address "x" is that of a special register, the dump will be the same as if the "pr" command was used.

4.16 r : resets pc back to starting position

User can reset the PC or Program Counter back to the starting position that would be set after a COG is loaded. This command will not alter the current values of the COG. If you need to restart the COG and debugger use the "R" command.

4.17 R : restarts COG

User can restart the COG and the debugger with this command. It is useful to have if the user wants to restart the COG after a "g" GO command. This allows restarting debug without downloading the program again.

4.18 sx v : sets value at COG address x = v

User can set the value of any COG address x to the value v with this command. Special registers (except INB,OUTB,DIRB) can be set to the value used by the COG instead of setting the special register shadow value. This means if you want to set a Propeller pin bit on OUTA, the bit will set assuming DIRA has the pin set to output. Registers that are marked read-only in the data sheet are not affected by this command.

4.19 tx : switch to COG task x

This command lets the user switch to a different COG debug task. This is only useful if the taskstart debug initialization procedure is followed. To use tasks, the COG debuggers must be started individually and the main utility.start method is called.

The demo shows the following debugger startup:

keywordstyle

```
bu.taskstart(@entry, @command, string("Main Task"))
bu.taskstart(@entry2,@command, 0)
bu.start                      ' start multi cog task debugger
```

The task debugging command set is the same as for single cog debugging.

4.20 z : shows flags

User can see the current state of the C carry flag and Z zero flag with this command. Z is kind of wierd for some users. Z=1 means that a zero flag set condition occured.

4.21 Enter : single-step

User can single step the PASM code by striking the Enter key. The debug output described above will be produced after each step.

4.22 ? : show this help screen

User can show the help screen with this command. The syntax on the help screen will be the syntax to follow if there is a conflict between that and these user comments.

5 Credits

BMA Multi-COG PASM Debugger was written by Jazzed, as was a large portion of this documentation.

From the source: Credit where it's due: Some portions of this code package were taken from work done by Ray Rodrick (Cluso99) but modified substantially to fit this design. His MIT license Copyright's are included where code is essentially copied.

Source: Copyright (c) 2009-2011 by John Steven Denson (jazzed)