

GDB & Skunkboard integration with Code::Blocks

ATARI JAGUAR

Windows version

Written by Dilinger, Proof-Read by JagChris
03/02/2015

Avant propos

The purpose of this (humble) document is to provide an easy and comfortable way to debug an Atari Jaguar application. GDB, by itself, is nice but working with the command line can be tedious. This is the reason, I¹ wanted to explore a way to have the best of both worlds: The power of the Skunkboard and a visual debugging tool.

After various attempts with the available IDEs, I²ve found Code::Blocks to be quite adapted for the task. But it is possible another IDE can offer the same or better results. It is up to you to give them a try and to share the results of course.

Also, the ELF format has been used but it is possible another format (such as COF) can be used; since no efforts have been attempted, it is up to you to give it a try.

Special thanks (in random order)

Usually special thanks come at the end, but this time let's put them right away.

- **Kskunk**, for his design of the Skunkboard
- **Tursi**, *the lion*, for his hard work and commitment to making the Skunkboard a reality along with GDB support.
- **JagChris**, who pointed out the ability to use GDB with the Skunkboard

¹ Selon un philosophe francais, le "Le Moi est haïssable"; this is the 1st and last time, I will use the "I" or "Je".

² This is the 2nd and, hopefully, last time, I will use the "I" or "Je" in the document.

Requirements

HW

- Obviously a Jaguar and a Skunkboard are required.
 - Please ensure the communication between your PC and the Jaguar is working correctly since this document doesn't describe that process.

SW

- Code::Blocks
 - It can be found at <http://www.codeblocks.org/>
 - The version used is the 13.12
- GDB with M68K ELF support
 - The version used is from the m68k-elf-gcc4.8.0 package
 - Package can be found at <http://gnutoolchains.com/m68k-elf/>
 - The version used is the 4.8.0
- jserve
 - It is found within the Skunkboard tools
 - The 2013, or more recent, version must be used
 - The tool must be launched in Administrator mode otherwise an error will occur
 - It is available at <http://www.harmlesslion.com/cgi-bin/onesoft.cgi?110>
 - Alternative version is at <http://3do.cdinteractive.co.uk/viewtopic.php?f=35&t=3598>

Jaguar application

- The executable must have been compiled and linked, with debug information, in ELF format

Limitations

The world is not perfect, there are limitations. But, since the journey has just started, such limitations can be overcome in the future.

Skunkboard / jserve and GDB capabilities

- ✓ Return current 68K registers
- ✓ Write new 68K registers
- ✓ Read memory contents
- ✓ Write memory contents
- ✓ Insert breakpoint
- ✓ Remove breakpoint
- ✓ Single step
- ✓ Continue (and single step)
- ✓ Shut down / kill

Notes

1. Breakpoints can be set only when code is located in RAM
2. Only M68K code is currently traceable
 - a. GPU information is not accessible (at time of writing)
 - b. DSP information is not accessible (at time of writing)

Configuring Code::Blocks

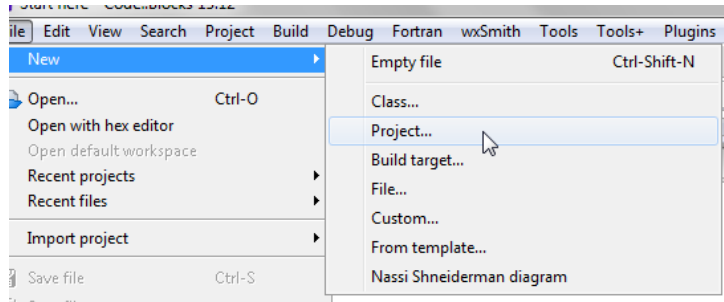
This IDE can be also used as a traditional editor, but this part won't be addressed.

Working with GDB

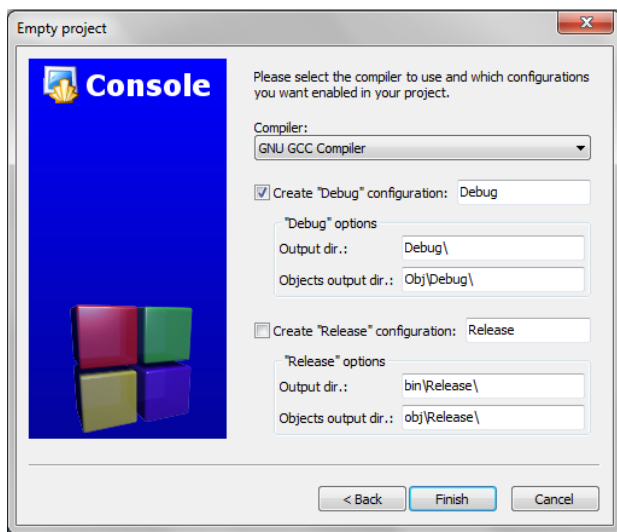
Before integrating and using GDB, a project needs to be created. This project can be either a full project or a “dummy” project.

Create a “Dummy” project

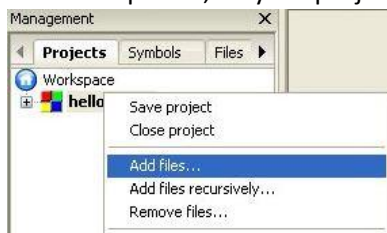
1. Launch Code::Blocks and go to the “New / Project”



2. Select “Empty project”
3. Fill the project information and reach the end of the project information
4. As is:

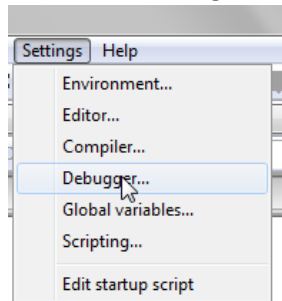


5. Fill the debug fields in accordance with your project directories
 - a. The “Release” field is up to you to be filled or not
6. Once completed, fill your project with a least one source file

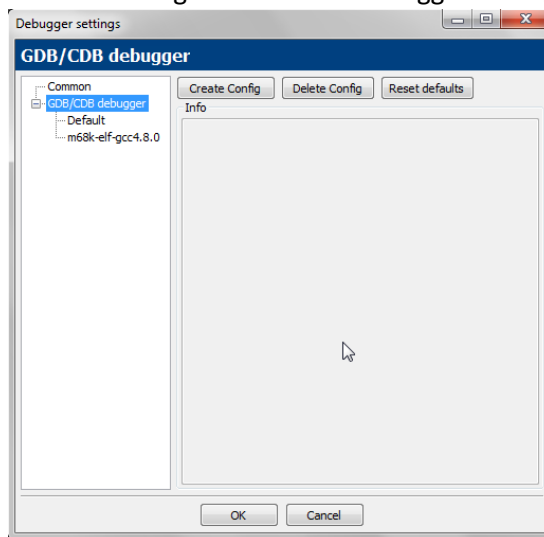


Configuring GDB

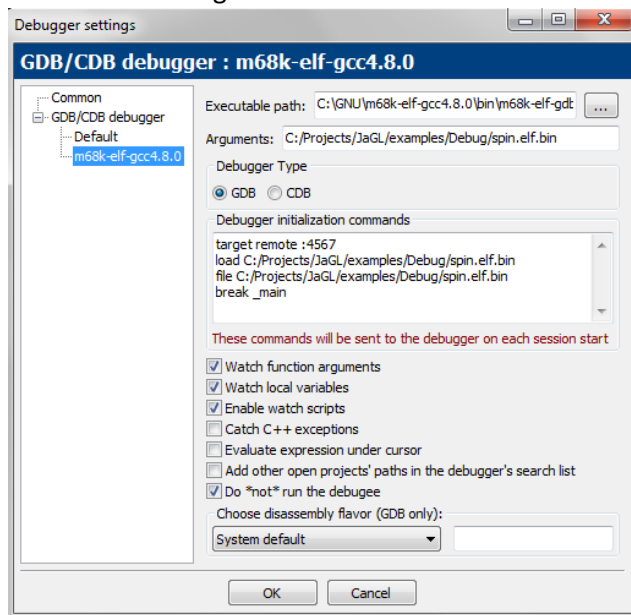
1. Once your project is configured
 - a. Go to the “Settings / Debugger”



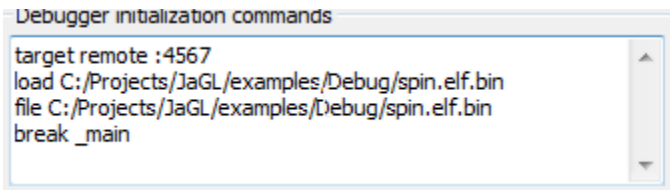
- b. Create a configuration in the debugger settings



- c. Name it as your convenience
 - d. Fill the GDB configuration



- Executable path must point to the GDB executable
- Arguments must be filled / point to the Jaguar application executable
- Debugger type must be **GDB**
- Debugger initialization commands



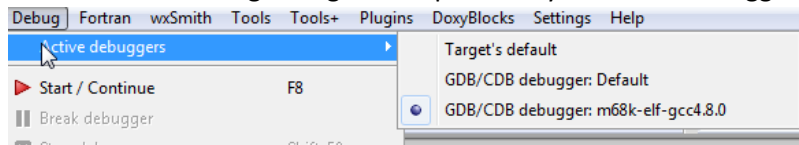
```
Debugger initialization commands
target remote :4567
load C:/Projects/JaGL/examples/Debug/spin.elf.bin
file C:/Projects/JaGL/examples/Debug/spin.elf.bin
break _main
```

- “target remote” code (4567 in our example) must be the same as the one displayed by jserve
- “load <...>” must point to the executable
- “file <...>” must point to the executable
- The ‘/’ character must be used
- “break” is optional
 - It can be useful to set a breakpoint at the beginning of the source code
 - Any valid address can be used instead
 - There is no known (reasonable) limit of the breakpoints to be set in the initializations command
- Additional GDB command can be added

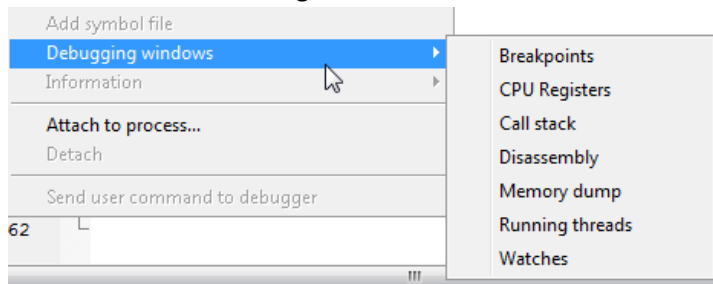
Debugging session

In order to start a debug session, you have to

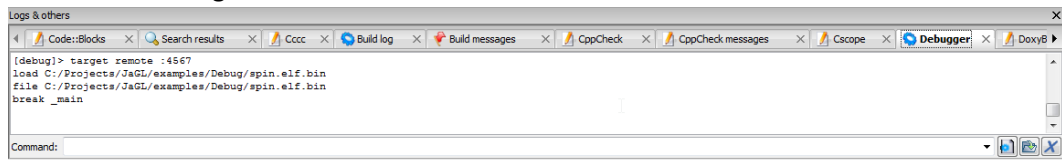
1. Launch jerve and make sure it is pending for a target remote
2. Then, once in Code::Blocks
 - a. Make sure the debug configuration points at your active debugger



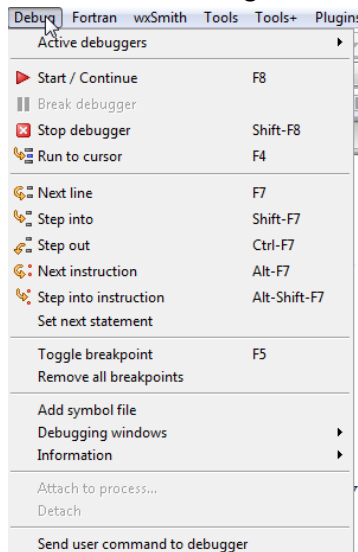
3. Select the debug windows of your preference
 - a. Windows available listing



- b. The selection can be modified, at any time, during your debug session
4. Use the "Start / Continue" command to launch the debug
5. GDB and jserve will communicate and load the executable and debug information
6. GDB & jserve will then be paused and ready to accept user command
7. At this point, you can
 - a. Use the GDB integrated command line



- b. Use the visual Debug command from Code::Blocks



Tips & Workaround

The memory window displays registers instead of the memory content

It happens sometimes; workaround is to close the memory window and reopen it. Note: The memory address is kept when a window is closed and reopened.

Can we see the content of a structure pointer allocated in memory?

Yes, but it requires to turn on the option “Evaluate expression under cursor” in your GDB configuration. Please note, in some occasion you may lose the functionality. Also, this option will apply for all expressions present in the source code.

1. Go or stay in the source code window
2. Move your cursor / mouse pointer to the expression name
3. GDB, via jserve, will look for the information
 - a. Depend the structure size and complexity, this may require a delay
4. A window will appear close by the expression name
 - a. This window lists the structure content
 - b. Depend the structure size and complexity, window may be quite large
5. It will be possible to navigate in the structure and see his entire content

Can we see the handmade ASM source code when tracing?

Such operation doesn't seem possible if the ASM object has been generated using SMAC (aoutnull format). So, it is only possible to trace such code via the Disassembly window.

- Since no another assembler has been tried, it is difficult to say if only SMAC is concerned.

Sometimes the assembly code is messed up in the Disassembly window

It can happen when looking at code assembled from handmade ASM source code (SMAC was used in order to get the problem). The code is still intact in memory but the Disassembly window displays duplicate code from the same address; so, only a portion of the code is displayed. However, tracing the code will continue to work.

There is no clear workaround, but tracing the code will clean-up the window after a while. Using “next instruction” rather than “step into instruction” options may help to avoid the problem.

Debugging experience is somewhat slow

The PC HW configuration may be the cause but these following tips can help

1. Select only the required windows for your session
2. If possible, turn off the split code option in the Disassembly window
3. When displaying the memory window, try to keep the length by default (32 bytes)
 - a. The more data you want to display, the larger the window will be required.

From time to time, a message “Can’t find compiler ... GCC ... warning” will be displayed



1. If Code::Blocks is used only for debug, this warning can be ignored.
2. If Code::Blocks is used also for programming, then you have to fix this warning.

Room for improvements

jsolve

1. To handle GPU/DSP information
 - a. This statement needs to be validated first by Tursi & Jagchris

GDB

2. To handle GPU/DSP information

Code::Blocks

3. To handle windows to display GPU/DSP information
4. To do a new window to display stack pile instead to use a memory window
5. To do a new window to display the HW registers