Debug in Vector CAST

Wednesday, October 16, 2024 4:47 PM

To debug your test case in VectorCAST, you can use the built-in debugging capabilities. Here are the steps to help you get started:

1. **Set the Debug Command**:
   - You can override the default debug command that VectorCAST uses. For example, if you prefer to use a different debugger than the default, you can set this using the ADA_DEBUG_CMD option.
   - Example command: clicast option ADA_DEBUG_CMD <debugger command>
2. **Execute the Test Case with Debugging**:
   - Right-click on the test case you want to debug and select Execute with Debug. This will launch the debugger with the VectorCAST test harness code loaded.
   - You can then perform all standard debug commands as you would normally.

Debug in Vector CAST

- => Right-click on the test case you want to debug and select Execute with Debug. This will launch the debugger with the VectorCAST test harness code loaded.

It looks like you're running a debug session using GNU Debugger (GDB) on a Windows system (mingw32). The console output you've shared indicates that GDB has successfully loaded the symbols for your executable.
Here are a few steps you can take next:

1. **Set Breakpoints**: You can set breakpoints in your code where you want the execution to pause. For example:
   break main
2. **Run the Program**: Start the execution of your program:
   run
3. **Step Through Code**: Use commands like next to step over lines of code or step to step into functions:
   next
   step
4. **Inspect Variables**: Check the values of variables at any point:
   print variable_name
5. **Continue Execution**: Resume the execution until the next breakpoint:
   continue
6. **Quit GDB**: Exit the debugger when you're done:
   quit

7. **When the Breakpoint is Hit**: Once the execution pauses at the breakpoint, use the following command to list all local variables.
   info locals
   This will display the names and current values of all local variables in the function where the breakpoint is set.

Arguments passed: If you need to see the arguments passed to the function, you can use:
info args
To set a breakpoint inside a specific function in GDB, you can specify the function name and the line number where you want the breakpoint. Here are a few ways to do it:

1. **Set a Breakpoint at the Start of a Function**:
   break function_name
2. **Set a Breakpoint at a Specific Line Number within a Function**:
   break function_name:line_number

   For example, to set a breakpoint at line 10 inside my_function:
   break my_function:10
3. **Set a Breakpoint at a Specific File and Line Number**:
   break file_name.c:line_number

   For example, to set a breakpoint at line 20 in main.c:
   break main.c:20
4. **Set a Breakpoint at a Specific Address**: If you know the memory address, you can set a breakpoint there:
   break *0xaddress
5. **Set a Breakpoint at a Specific Condition**: You can also set conditional breakpoints that only trigger when a certain condition is met:
   break function_name if condition

   For example, to break when x is greater than 5:
   break my_function if x > 5

1. **File Not Loaded Yet**: The source file might not be loaded into the debugger yet. By choosing "yes" to make the breakpoint pending, GDB will set

the breakpoint once the file is loaded.

2. **Incorrect Path**: The path to the source file might be incorrect or the file might be located in a different directory than GDB expects.
3. **File Renaming or Preprocessing**: If you're using a tool like VectorCAST, it might preprocess or rename files for its purposes, which could cause GDB to not recognize the original file name.

To troubleshoot this, you can:

- **Verify the File Path**: Ensure that the file DGU_SupportCheckDIDs.c is in the expected directory and that GDB has access to it.
- **Check the Compilation Directory**: Make sure GDB is looking in the correct directory for the source files. You can use the directory command in GDB to add directories to the search path:
  ```
  directory /path/to/source/files
  ```
- **List Source Files**: Use the info sources command to list all source files that GDB knows about:
  ```
  info sources
  ```

If the file is indeed renamed or moved by VectorCAST, you might need to adjust your debugging setup to account for these changes.

To list the source file for a function where you've successfully set a breakpoint, you can use the info breakpoints command to get details about the breakpoints, including the file and line number. Here's how you can do it:

1. **List Breakpoints**: Use the info breakpoints command to see all the breakpoints you've set.
   ```
   info breakpoints
   ```
   This will display a list of breakpoints with their numbers, types, addresses, and locations (including the source file and line number).
2. **Check the Source File**: Look for the entry corresponding to your breakpoint. It will show the source file and line number where the breakpoint is set.

For example, the output might look something like this:

```
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000401176 in main at main.c:10
```

In this example, the breakpoint is set in the main function at line 10 of main.c.
If you need to view the source code around the breakpoint, you can use the list command:

```
list function_name
```

or

```
list file_name:line_number
```

This will display the source code around the specified function or line number.

To print a value in hexadecimal format in GDB, you can use the print command with the /x format specifier. Here's how you can do it:

1. **Print a Variable in Hexadecimal**:
   ```
   print /x variable_name
   ```
2. **Print a Memory Address in Hexadecimal**:
   ```
   print /x *address
   ```
3. **Print an Expression in Hexadecimal**:
   ```
   print /x expression
   ```

For example, if you have a variable int_var and you want to print its value in hexadecimal:

```
print /x int_var
```

This will display the value of int_var in hexadecimal format.

```
### Basic Commands
1. **Run the Program**:
run
Starts the execution of your program.
2. **Quit **:
quit
Exits the debugger.
### Breakpoints
3. **Set a Breakpoint at the Start of a Function**:
```

```
break function_name
Example:
break main
4. **Set a Breakpoint at a Specific Line Number within a Function**:
break function_name:line_number
Example:
break my_function:10
5. **Set a Breakpoint at a Specific File and Line Number**:
break file_name.c:line_number
Example:
break main.c:20
6. **Set a Breakpoint at a Specific Address**:
break *0xaddress
7. **Set a Breakpoint with a Condition**:
break function_name if condition
Example:
break my_function if x > 5
### Inspecting Code and Variables
8. **List Source Code Around a Function**:
list function_name
Example:
list main
9. **List Source Code Around a Specific Line**:
list file_name:line_number
Example:
list main.c:20
10. **Print a Variable**:
print variable_name
Example:
print x
11. **Print a Variable in Hexadecimal**:
print /x variable_name
Example:
print /x int_var
12. **Print All Local Variables**:
info locals
13. **Print Function Arguments**:
info args
### Controlling Execution
14. **Continue Execution**:
continue
Resumes execution until the next breakpoint.
15. **Step Over a Line of Code**:
next
Executes the next line of code, stepping over function calls.
16. **Step Into a Function**:
step
Steps into the function call.
### Managing Breakpoints
17. **List All Breakpoints**:
info breakpoints
18. **Delete a Breakpoint**:
delete breakpoint_number
Example:
delete 1
### Source Files
19. **List Source Files**:
info sources
20. **Add Directory to Search Path**:
directory /path/to/source/files
Sure! Here are 20 more useful  commands that can help you with debugging:
### Advanced Breakpoints and Watchpoints
1. **Set a Temporary Breakpoint**:
tbreak function_name
Example:
tbreak main
2. **Set a Hardware Breakpoint**:
hbreak function_name
Example:
hbreak my_function
3. **Set a Watchpoint for a Variable**:
watch variable_name
Example:
watch x
4. **Set a Watchpoint for an Expression**:
watch expression
Example:
watch *ptr
5. **Set a Conditional Watchpoint**:
```

```
watch variable_name if condition
Example:
watch x if x > 5
### Inspecting and Modifying Variables
6. **Display a Variable Every Time Execution Stops**:
display variable_name
Example:
display x
7. **Remove a Display**:
undisplay display_number
Example:
undisplay 1
8. **Change the Value of a Variable**:
set variable_name = value
Example:
set x = 10
### Stack and Frames
9. **Print the Call Stack**:
backtrace
or
bt
10. **Print the Call Stack with Full Arguments**:
backtrace full
11. **Select a Stack Frame**:
frame frame_number
Example:
frame 2
12. **Move Up the Stack**:
up
13. **Move Down the Stack**:
down
### Memory Inspection
14. **Examine Memory**:
x/nfu address
- `n` is the number of units to display
    - `f` is the format (e.g., `x` for hexadecimal, `d` for decimal)
    - `u` is the unit size (e.g., `b` for byte, `h` for halfword, `w` for word, `g` for giant word)
Example:
x/4xw 0x600000
15. **Print Memory as Characters**:
x/s address
Example:
x/s 0x600000
### Controlling Execution
16. **Step Out of a Function**:
finish
17. **Continue Until a Specific Line**:
until line_number
Example:
until 50
18. **Continue Until a Specific Address**:
until *0xaddress
### Miscellaneous
19. **Print the Value of an Expression**:
print expression
Example:
print x + y
20. **Evaluate an Expression Without Printing**:
call function_name(arguments)
Example:
call my_function(5)
```

To set multiple breakpoints inside a function within a C file using GDB, you can specify the exact line numbers where you want the breakpoints. Here's how you can do it:

1. **Set Breakpoints at Specific Line Numbers**:
   break file_name.c:line_number

   Repeat this command for each line number where you want to set a breakpoint.
   Example:
   break myfile.c:10
   break myfile.c:20
   break myfile.c:30
2. **Set Breakpoints Using Function and Line Number**: If you know the function name and the line numbers within that function, you can set breakpoints like this:

```
break function_name:line_number
```

Example:

```
break my_function:15
break my_function:25
```

3. **Set Breakpoints Using Addresses**: If you have the memory addresses, you can set breakpoints at those addresses:

   ```
   break *0xaddress
   ```

   Example:

   ```
   break *0x400123
   break *0x400456
   ```

Example Scenario

Suppose you have a function process_data in data.c and you want to set breakpoints at lines 10, 20, and 30 within this function:

```
break data.c:10
break data.c:20
break data.c:30
```

Or, if you prefer to use the function name:

```
break process_data:10
break process_data:20
break process_data:30
```

Listing and Managing Breakpoints

After setting the breakpoints, you can list them using:

```
info breakpoints
```

To delete a specific breakpoint:

```
delete breakpoint_number
```

Example:

```
delete 1
```

To delete all breakpoints:

```
delete
```