Essential ARM Cortex-M Debugging with GDB

Kevin Townsend
Linaro
Overview

● Core Concepts
  ○ Connecting
  ○ Basic Navigation
  ○ Breakpoints
  ○ Watchpoints
  ○ Contextual Information
  ○ Stack Backtrace
  ○ Printing
  ○ Examining Memory
  ○ Examining Source Code
  ○ Searching Memory
  ○ Multiple Image Support

● Practical Examples
  ○ ISR Stack Rollback
  ○ Fault Identification

● Python Scriptability
  ○ Core GDB Functions
  ○ Minimal Python GDB Command
  ○ Loading Python GDB Commands
  ○ ARMv8-M (Cortex M33) Fault Handler
Core Concepts
Connecting

In one terminal window start GDB server:

$ pyocd gdbserver --target=lpc55s69
or $ JLinkGDBServer -if SWD            
  -speed 4000 -USB -device lpc55s69

Then connect to it in a second window:

$ arm-none-eabi-gdb-py              
  -s build/zephyr/zephyr.elf           
  -ex "target remote tcp:localhost:3333"

(gdb) monitor reset halt
(gdb) break main
(gdb) continue

# Various tools like `pyocd` and `JLinkGDBServer` can be used to start a new GDB server. Parameters and requirements vary by tool, target hardware and debugger.

# Connect to the server to start a new (local or remote) debug session over TCP. Check the GDB Server output for the correct TCP port!

# Reset the firmware and stop after reset
# Set a breakpoint at `main()`
# Restart execution
Basic Navigation

- ctrl+c
- c/continue
- s/step
- s 10
- n/next
- n 10
- u/until 20
- f/finish

💡 c/continue format with a ‘/’ indicates the shortcut command and the full command in this presentation. Either of the two values can be used to the same effect.

# Halt current program execution
# Resume execution
# Step into function
# Step next 10 sources lines
# Run next line in func (step over)
# Run next 10 lines in current func
# Run until line 20 of current file
# Run to the end of func/stack frame
Breakpoints (fast!)

- b/break main
- b main.c:func
- b main.c:18
- b main.c:18 if foo > 20
- tbreak main
- info breakpoints
- ignore 2 20 ⭐️
- disable 2
- delete 2

# Break on main() entry
# Break on func() in main.c
# Break on line 18 of main.c
# Break if ‘foo’ > 20 (boolean cond)
# Fires once, deletes itself
# List all breakpoints
# Ignore bp 2 the first 20 hits
# Disable bp 2
# Delete bp 2
Watchpoints (Powerful, but sloooow)

Execution halts when variable is accessed or modified

- watch foo
- watch myarray[10].val
- watch *0x1000FEFE
- watch foo if foo > 20 ⭐
- watch foo if foo + x > 20
- info watchpoints
- delete 7

# Watch foo
# Watch .val in myarray[10]
# Watch memory addr 0x1000FEFE
# Conditional watch (foo > 20)
# Complex conditional expression
# List watchpoints
# Delete watchpoint 7
Contextual Information

- info locals
- info variables
- info args
- info registers

# Local variables
# Global variables
# Function argument variables
# Core registers
Stack Backtrace

- bt
  # Display a stack backtrace (function call history)
- frame
  # Display the current stack frame
- up
  # Move up the stack (to main)
- down
  # Move down (away from main)
Printing

p/print [/FMT] expression

- p foo
- p foo+bar
- p/x &main
- p/x $r4
- p/a *(uint32_t[8]*)0x1234 ⭐

# Print the value of ‘foo’
# Print complex expression
# Print address of main()
# Print register R4 in hex
# Print array of 8 u32s @ 0x1234
Examining Memory

x [/FMT] addr

FMT is a repeat count, followed by a format and size letter.

- x foo
- x/4c 0x581F
- x/4xw &main

# Show address of variable foo
# Show four chars @ 0x581F
# Show four words in hex @ main()
Examining Source Code

- list
- list *0x1234
- list main.c:func

- disas func

# Show src for current location
# Show src at the address 0x1234
# Show src for func() from main.c
# List ASM code for func()
Searching Memory

- `find /b 0x0, 0x10000, 'H', 'e', 'l', 'l', 'o'
  0x581f
  1 pattern found.

- `x/s 0x581f
  0x581f: "Hello World! %s\n"

💡 Useful when checking for stack overflow, if stack memory is pre-filled with a known pattern.

# Search for a byte pattern between 0x0 and 0x10000

# Examine string @ 0x581F
Multiple Image Support

GDB parses one ELF file at a time for symbol lookup.

Complex projects may have two or more files. For example:

1. bootloader.elf
2. trustzone_secure.elf (TF-M)
3. trustzone_nonsecure.elf (Zephyr)

The ELF file can be switched while debugging via:

- symbol-file trustzone_secure.elf  # Load new elf file
Practical Examples
ISR Stack Rollback (ARMv7-M, ARMv8-M)

- p/t $lr
  
  # Display binary value of `lr` register
  If b2 = 1, use `psp`
  If b2 = 0, use `msp`

- p/a *(uint32_t[8])*)$psp
  or
  p/a *(uint32_t[8])*)$msp

# Display the `psp` or `msp` stack frame.
7th value is the `pc` register value

- list *0x12345678
  
  # Show source for `pc` addr
Fault Identification (ARMv8-M)

- `p/x *(uint32_t*)0xE000ED24` # System Handler Control & State Register (SHCSR)
- `p/x *(uint32_t*)0xE000ED2C` # HardFault Status Register (HFSR)
- `p/x *(uint32_t*)0xE000ED28` # Configurable Fault Status Register (CFSR)

❗ JLinkGDBServer (V6.86) was used to access this memory range. Pyocd 0.30.0 seems to have access restrictions to this memory range, further investigation required as to why.
System Handler Control & State Register (0xE000ED24)
HFSR

HardFault Status Register (0xE000ED2C)

- VECTTLB = 1 means a vector table read fault has occurred
- FORCED = 1 means the processor has escalated a configurable-priority exception to HardFault
CFSR

Configurable Fault Status Register (0xE000ED28)

- MMFSR = MemManage Fault Status Register
- BFSR = BusFault Status Register
- UFSR = UsageFault Status Register
Fault Identification: Example

- (gdb) \texttt{p/x *(uint32\_t*)0xE000ED24}
  $4 = 0x50008$
- (gdb) \texttt{p/x *(uint32\_t*)0xE000ED28}
  $6 = 0x200$

# $\text{SHCSR} = 0x50008$ indicates that we have a **UsageFault**
# CFSR bits 16-31 = UFSR

Divide by zero
Python Scriptability
Python GDB Integration

Recent versions of GDB allows us to extend the GDB server with custom commands or helper functions written in Python. Functions must be loaded into the GDB session.

You interact with GDB in python via gdb.*, for example:

```python
# Read the system handler control & state register
shcsr = int(gdb.parse_and_eval("*(uint32_t *)0xE000ED24"))
print("SHCSR: 0x%08X" % shcsr)
```

❗ Not every version of GDB supports python, but many modern toolchains do. Some toolchains include Python and non-Python variants, such as the GNU Arm Embedded Toolchain with `arm-none-eabi-gdb` and `*gdb-py`.
#!/usr/bin/env python
import gdb

class Minimal (gdb.Command):
    def __init__(self):
        super(Minimal, self).__init__("minimal",
            gdb.COMMAND_USER)

    def invoke(self, arg, from_tty):
        # Get the current value of the PC register
        pc = gdb.parse_and_eval("$pc")
        print("PC: 0x%08X" % pc)

Minimal()
Loading Python GDB Commands

- (gdb) source minimal.py  # Load python script into GDB
- (gdb) minimal()  # Execute new ‘minimal’ command
  PC: 0x10001C2C

You can load a variety of commands in a single debug session.

GDB Python commands are useful to encapsulate complicated tasks involving numerous registers.
Cortex M33 (ARMv8-M) Fault Handler

(gdb) faultdetails()

Fault Status Registers:
- SHCSR: 0x000F0008
- CFSR: 0x00000000
- HFSR: 0x00000000

UsageFault exception active!
- UFSR: 0x0000

Previous Stack Frame: psp
- R0: 0x00000000
- R1: 0x00000000
- R2: 0x80000000
- R3: 0x10001821
- R12: 0x00000000
- LR: 0x1000047D // (EXC_RETURN)
- PC: 0x1000047C

Thank you

Accelerating deployment in the Arm Ecosystem