



## Zephyr GDB Debugger Lab

Rev1.0, May 29, 2022

### Objectives

In this lab, you will learn

- Flash and debug a Zephyr application using gdb command-line

### Pre-Requisites

This lab is written for Windows10, but Zephyr and this lab can easily be done in Linux or MacOS. This lab guide was written for Zephyr release v3.0.0.

- Follow Zephyr\_Installation\_Lab.pdf
- Terminal Program, like [PuTTY](#) or [Tera Term](#)
- Follow Zephyr\_hello\_world\_Sample\_Lab.pdf to prepare the hardware, verify environment variables, connect the terminal to the board

### Hardware Requirements

- MIMXRT1060-EVKB
- Micro-USB cable (may need two cables, see Zephyr\_hello\_world\_Sample\_Lab.pdf)

### Running the lab

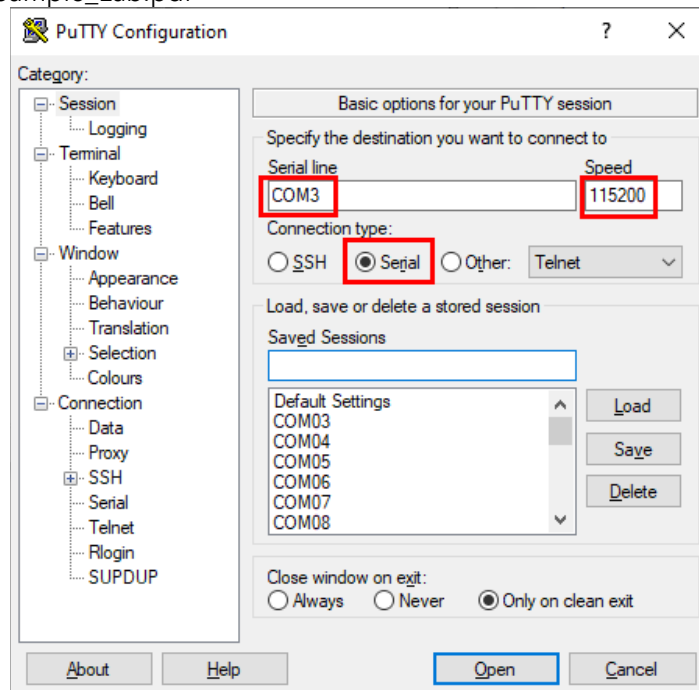
#### Prepare Command Line

Before using the West commands, a command window should be opened, python virtual environment activated, and environment variables set. These steps only need to be done once after a command window is opened. If already done in a previous lab, skip to the next section. See Zephyr\_hello\_world\_Sample\_Lab.pdf for more details.

1. Open a normal command prompt in Windows
2. Navigate to the zephyr directory:  
**cd %userprofile%\zephyrproject\zephyr**
3. Activate the python virtual environment. After activating, the prompt should now include (.venv), see screenshot below:  
**..\venv\Scripts\activate.bat**
4. Set the environmental variables. Note, when using a command file like this to set the variables, the command file needs to be executed once anytime a new command window is opened:  
**zephyr-env.cmd**



5. Connect a terminal program to the board's COM port, for help see [Zephyr\\_hello\\_world\\_Sample\\_Lab.pdf](#)



## Debugging an application with GDB

In this section, we will walk through using GDB command line to connect to your target, set breakpoints, and perform a stack backtrace.

6. Build the synchronization example using west:  
**west build -b mimxrt1060\_evk -d ../build-sync samples\synchronization --pristine**
7. Debug the application:  
**west debug -d ../build-sync**



8. You should see the J-Link GDB Server pop up

The image shows two overlapping windows. The top window is a Command Prompt titled "Command Prompt - west debug -d ..\build-sync". It displays the output of the command "west debug -d ..\build-sync". The output shows that the J-Link GDB server is running on port 2331, and the GNU gdb (GNU Arm Embedded Toolchain 10-2020-q4-major) 10.1.90.20201028-git is being used. The gdb interface shows the loading of symbols from the ELF file and the start of remote debugging using port 2331. The bottom window is the SEGGER J-Link GDB Server V7.62c application. It shows the GDB connection status as "127.0.0.1, 1 client connected". The J-Link status is "Connected". The device is "MIMXRT1062xxx6A (Halted)". The voltage is "3.30V". The transfer rate is "2000 kHz". The endianness is "little endian". The log file is "(Not enabled)". The status bar at the bottom indicates "23 KB downloaded" and "Connected to target".

```
Command Prompt - west debug -d ..\build-sync

(.venv) C:\Users\... \zephyrproject\zephyr>west debug -d ..\build-sync
-- west debug: rebuilding
ninja: no work to do.
-- west debug: using runner jlink
-- runners.jlink: JLink version: 7.62c
-- runners.jlink: J-Link GDB server running on port 2331; no thread info available
GNU gdb (GNU Arm Embedded Toolchain 10-2020-q4-major) 10.1.90.20201028-git
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ..\build-sync\zephyr\zephyr.elf...
Remote debugging using :2331
__DSB ()
   at C:/Users/.../zephyrproject/modules/hal/cmsis/CMSIS/Core/Include/cmsis_gcc.h:271
271  __ASM volatile ("dsb 0xF:::memory");
Resetting target
Loading section rom_start, size 0x22c0 lma 0x60000000
Loading section text, size 0x32f4 lma 0x600022c0
Loading section .ARM.exidx, size 0x8 lma 0x600055b4
Loading section initlevel, size 0x58 lma 0x600055bc
Loading section devices, size 0xa8 lma 0x60005614
Loading section sw_isr_table, size 0x500 lma 0x600056bc
Loading section device_handles, size 0x2e lma 0x60005bbc
Loading section rodata, size 0x200 lma 0x60005bec
Loading section datas, size 0x10 lma 0x60005dec
--Type <RET> for more, q to quit, c to continue without paging--
Loading section device_states, size 0x1c lma 0x60005dfc
Loading section k_sem_area, size 0x20 lma 0x60005e18
Start address 0x600034e0, load size 24118
Transfer rate: 1 KB/sec, 2192 bytes/write.
(gdb) _
```

SEGGER J-Link GDB Server V7.62c

File Help

GDB 127.0.0.1, 1 client connected ☐ Stay on top

J-Link Connected ☐ SWD 2000 kHz ☐ Show log window

Device MIMXRT1062xxx6A (Halted) ☐ 3.30V little endian ☐ Generate logfile

Logfile (Not enabled) ☐ Verify download

Clear Log

23 KB downloaded Connected to target



9. Reset the device  
**monitor reset**

10. Open the source file **%userprofile%/zephyrproject/zephyr/samples/synchronization/src/main.c**.  
If you don't have a preferred editor, you can use [Notepad++](#) to view the code with line numbers. We will set a breakpoint in the next step in `helloLoop()` at the highlighted line 50 below.

```
void helloLoop(const char *my_name,
               struct k_sem *my_sem, struct k_sem *other_sem)
{
    const char *tname;
    uint8_t cpu;
    struct k_thread *current_thread;

    while (1) {
        /* take my semaphore */
        k_sem_take(my_sem, K_FOREVER);

        current_thread = k_current_get();
        tname = k_thread_name_get(current_thread);
#if CONFIG_SMP
        cpu = arch_curr_cpu()->id;
#else
        cpu = 0;
#endif

        /* say "hello" */
        if (tname == NULL) {
            printk("%s: Hello World from cpu %d on %s!\n",
                  my_name, cpu, CONFIG_BOARD);
        } else {
            printk("%s: Hello World from cpu %d on %s!\n",
                  tname, cpu, CONFIG_BOARD);
        }

        /* wait a while, then let other thread have a turn */
        k_busy_wait(100000);
        k_msleep(SLEEPTIME);
        k_sem_give(other_sem);
    }
}
```

11. Set a breakpoint in application. We will place the breakpoint at the highlighted line above:  
**break main.c:50**

```
(gdb) break main.c:50
Breakpoint 1 at 0x600025ea: file C:/Users/.../zephyrproject/zephyr/samples/synchronization/src/main.c, line 50.
```



- Continue your app. This will halt at your breakpoint.

**c**

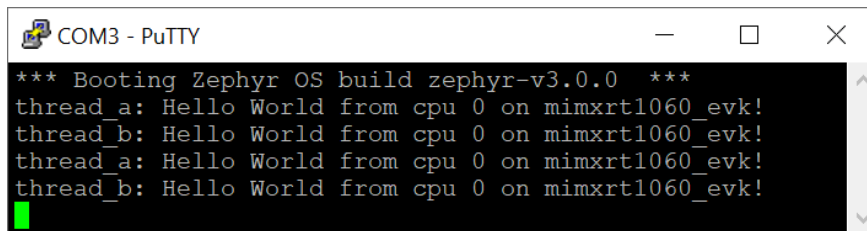
- Continue a few more times. You will repeatedly halt at this breakpoint in the loop. Notice the thread calling this loop toggles between threadA and threadB. And the app will print a line to the terminal with each run through the loop

**c** <repeat a few times>

```
Breakpoint 1, helloLoop (my_name=my_name@entry=0x60005d55 <__func__.0> "threadA", my_sem=my_sem@entry=0x8000002c <threadA_sem>,
other_sem=other_sem@entry=0x8000003c <threadB_sem>) at C:/Users/.../zephyrproject/zephyr/samples/synchronization/src/main.c:50
50      tname = k_thread_name_get(current_thread);
(gdb) c
Continuing.

Breakpoint 1, helloLoop (my_name=my_name@entry=0x60005d5d <__func__.1> "threadB", my_sem=my_sem@entry=0x8000003c <threadB_sem>,
other_sem=other_sem@entry=0x8000002c <threadA_sem>) at C:/Users/.../zephyrproject/zephyr/samples/synchronization/src/main.c:50
50      tname = k_thread_name_get(current_thread);
(gdb) c
Continuing.

Breakpoint 1, helloLoop (my_name=my_name@entry=0x60005d55 <__func__.0> "threadA", my_sem=my_sem@entry=0x8000002c <threadA_sem>,
other_sem=other_sem@entry=0x8000003c <threadB_sem>) at C:/Users/.../zephyrproject/zephyr/samples/synchronization/src/main.c:50
50      tname = k_thread_name_get(current_thread);
(gdb)
```



- Display some local variables while debugging

**display current\_thread**

**display tname**

**c** <two times>

```
Breakpoint 1, helloLoop (my_name=my_name@entry=0x60005d5d <__func__.1> "threadB", my_sem=my_sem@entry=0x8000003c <threadB_sem>,
other_sem=other_sem@entry=0x8000002c <threadA_sem>) at C:/Users/.../zephyrproject/zephyr/samples/synchronization/src/main.c:50
50      tname = k_thread_name_get(current_thread);
1: current_thread = (struct k_thread *) 0x800000f8 <threadB_data>
2: tname = <optimized out>
(gdb) c
Continuing.

Breakpoint 1, helloLoop (my_name=my_name@entry=0x60005d55 <__func__.0> "threadA", my_sem=my_sem@entry=0x8000002c <threadA_sem>,
other_sem=other_sem@entry=0x8000003c <threadB_sem>) at C:/Users/.../zephyrproject/zephyr/samples/synchronization/src/main.c:50
50      tname = k_thread_name_get(current_thread);
1: current_thread = (struct k_thread *) 0x80000050 <threadA_data>
2: tname = <optimized out>
```

- Step into the next line

**s** (for step)

- Step over a line

**n** (for next)



17. Perform a stack backtrace to see the history of the stack:  
**bt**

18. Show the breakpoints:

**info breakpoints**

```
(gdb) info breakpoints
Num   Type             Disp Enb Address            What
1      breakpoint       keep y   0x600025ea in helloLoop at C:/Users/.../zephyrproject/zephyr/samples/synchronization/src/main.c:50
```

19. Delete your breakpoint  
**delete 1**

20. Let the app run with no breakpoints, and see the app print to the terminal  
**c**

21. Halt the app  
**Ctrl+c**

22. Quit GDB with the command below.  
**quit**  
**y**

This completes this lab.

## Additional resources:

To learn more about GDB debugging, see this presentation about Linaro:

<https://connect.linaro.org/resources/lvc21/lvc21-308/>

## Revision History

Rev	Date	Details
1.0	5/29/2022	Initial version