EECS 192: Mechatronics Design Lab

Discussion 8: GDB Debugging

GSI: Andrew Barkan

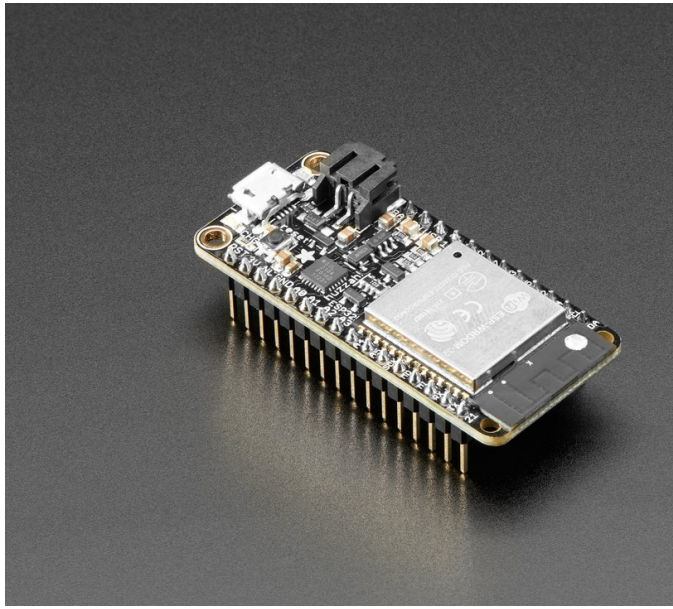
10 Mar 2019 (Week 8)

- GDB and Core Dump
- GDB Demo

GNU Debugger

Debugging

- General idea: locate bugs in your program by stopping it at particular points and looking at values
- GDB (gdb) "GNU Debugger" for C, C++, and other languages:
  - GDB should already be installed w/ ESP-IDF)
  - Run GDB script with core dump
  - Examine program with GDB
- (Documentation link)



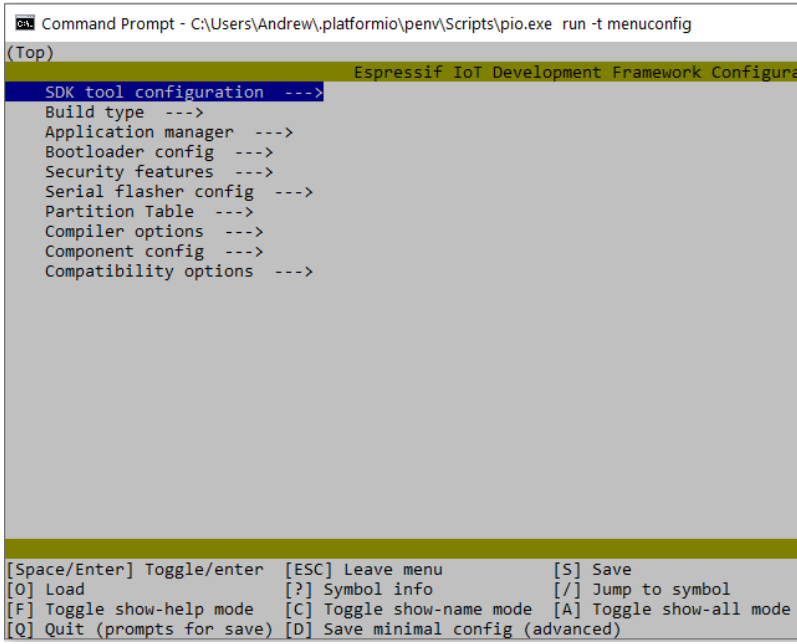
What's wrong with my code?

Setting Up GDB

- You may have to set your paths to run from terminal
- Alternatively, you can run the ESP-IDF export script after opening a new terminal
  - `Navigate to /.platformio/packages/framework-espidf`
  - `Run the export script (On Windows => export.bat)`
- The export script may have you run install.bat first
- You should then be able to run the gdb core dump script

Preparing Your ESP32

- We need to enable core dump
- Compiler config variables
  - /.platformio/penv/Scripts/pio.exe run -t menuconfig
  - Make sure you run from your project folder!
- Take a look at the SkeletonHuzzah32 wiki for more information



Setting compiler config variables

Preparing Your ESP32

- Here are the specific variables you need to set:
  - CONFIG_ESP_COREDUMP_TO_FLASH_OR_UART (change to UART)
  - CONFIG_ESP_SYSTEM_PANIC_PRINT_REBOOT or ESP_SYSTEM_PANIC_PRINT_HALT (change to print registers and halt)
  - CONFIG_ESP_COREDUMP_DATA_FORMAT (set to ELF)
  - Component config -¿ Core dump -¿ ELF format ELF format (Executable and Linkable Format file for core dump)
  - Compile with debugger (default option)
- After setting variables, save (s) and quit (q)
- Restart VS Code and allow CMakeLists to rebuild

Generating A Core Dump

- Probably already seen a core dump from serial output
- Big wall of seemingly meaningless characters (actually base64)
- What if I want to examine a specific point?
  - `*((int *) 0) = 0;`
  - Will cause a panic and core dump!

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAlxuo=
B8IQIgVCEAwAgYAPFcIgDBO+z8BAAAABAAAANxB+z8KAA
E77P+AA8D8BAAAAZAH7PwEAAAAgAAYAIwcGAAAAAAD//w
AAAAKghCEABAAAA4E0IQJzM+j8AAAAAAAAAAAAAAAD//z
AAAAAAAAAAAAAAAAAAAAKgfCEC0SwhAMAAFAAAAAAD7HQi
AAAAMgdCEABAAAAAAAAAALxLCICQTvs/AQAAAAAAAAAAA
////wAAAAAAAAAAAAAAAChL+z8AAAAAAQAAAAAAAAAAAA
AcGAMxB+z84T/s/AAAAAAAAAADQTvs/AAAAAAAAAAAAAA
BMIgIB9/j8qAAAAKgAAAAAAAAAAAAAA3E77PwAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAA==
Jr7P2Cc+z8AAAAAaD/7P0BP+z+Mgvs/YD/7PxgAAACYhfs
QAAAMiV+z91YXJ0X2xvZ190YXNrAD8A////f8Sc+z8AAA
QAAAAAAAAAAAAAAAAAAAB8AAAAAAAAAAuOj6PyDp+j+I6fc
AAAAAAAAAAAAAAASB0AQAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACD+z8=
B8IQIgVCEAwAAYAPFcIgCCb+z8BAAAABAAAANxB+z/cN/s
Jv7P+AA8D8BAAAAZAH7PwEAAAAgAAYAIwIGAAAAAAD//w
AAAAKghCEABAAAA4E0IQCwa+z8AAAAAAAAAAAAAAAD//z
AAAAAAAAAAAAAAAAAAAAKWlpaWlpaWlpaWlpaWlpaXQDw2
aWlpaWlpaWlpaWlpaWlpbxLCICAm/s/AAAAAAAAAAAAAA
////6WlpaWlpaWlpaWlpbyF+z8AAAAAAQAAACMIBgAAAA
aWlpaWlpaWlpaWlpaWlpaWlpaWlpaWlpaWlpaWlpaWoHwh
Jz7PwAAAAAAAAAAAAAAAAAAAAC8Dw1AAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Br7PwAAAAAAAAAAAAAAACMABgAAQPs/CGb7PwAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Ej7P7BJ+z9QRo4IxPr6P2g/+z9Y/vo/YD/7PwEAAAAo/vc
AAAABxG+z9pcGMwAFvSqN0Fb075g1kAAAAAABhK+z8AAAA
AAAAAAAAAAAAAAAAAAAAGAoAAAAAAAAAAuOj6PyDp+j+I6fc
AAAAAAAAAAAAAAASB0AQAAAAAAAAAAAAAAAAAAAAAAAAA
```

Some core dump text

Processing Core Dump

- When your program generates a core dump, the result is printed to over UART to whatever serial interface you are using
- Copy and save the core dump to a .txt file
- Remove:

  ```
  ================= CORE DUMP START =================
  ```

  and

  ```
  ================= CORE DUMP END ===================
  ```

  from text file

Processing Core Dump

Now that we have the core dump saved, we can use the ESP-IDF core dump script to begin debugging

- Located in /.platformio/packages/framework-espidf/components/espcoredump
- Here is the command template:

```
python espcoredump.py dbg_corefile -t b64 -c
    </path/to/saved/base64/text> </path/to/
    program/elf/file>
```

GDB Commands

- Keep in mind that you are NOT running the program; you're just looking at a snapshot taken at the instant of core dump
- Useful commands for debugging core dump with GDB:
  - `help` : Brings up help interface w/ more commands
  - `help <command>` : Gives helpful info on given command
  - `list` : Lists 10 lines around the error line
  - `list <line #>` : Lists 10 lines around the given line #
  - `bt` : Backtrace function calls from error location
  - `info <subject>` : Print out info on given subject (e.g. locals)

More GDB Commands

- You can also examine specific frames and variables!
- Some more commands:
  - `frame <frame #>` : Brings you to specified frame #
  - `print <expression>` : Print the value of an expression (e.g. a variable)
  - `thread <thread #>` : Brings you to specified thread

Some Debugging Tips

- When working on new functionality, take small controlled steps
  - Start with functional code (make sure it is committed)
  - Make small change so that ensuing errors can be isolated
- Math is hard
  - Implicit casting and integer math can cause issues
  - If debugging math, inspect intermediate values
  - e.g. a, b, c, (a + b), c * (a + b)
- Print statements are still useful tools!
- Preemptive error handling can be powerful (even when prototyping)

GDB Example