The GNU Project Debugger - GDB

Amine Gaizi

December 5th, 2019

Summary

- Introduction
- What is GDB ?
- What can GDB do ?
- How GDB could have been useful for this course

Introduction

- Every program in C is compiled using GNU Compiler Collection (GCC)
- For microcontrollers: generate a hexadecimal source file (hex file)
- Debugging information can be generated as well in a separate file

```
amine@ubuntu:~/opt/lab4/lab4_v2$ make all
avr-gcc -g -Wall -O2      -mmcu=atmega128  -DF_CPU=16000000UL    -c -o lab4.o lab4.c
avr-gcc -g -Wall -O2      -mmcu=atmega128  -DF_CPU=16000000UL    -c -o hd44780.o hd44780.c
avr-gcc -g -Wall -O2      -mmcu=atmega128  -DF_CPU=16000000UL -Wl,-Map,lab4.map -o lab4.elf lab4.o hd44780.o
avr-objdump -h -S lab4.elf > lab4.lst
avr-objcopy -j .text -j .data -O ihex lab4.elf lab4.hex
avr-objcopy -j .text -j .data -O binary lab4.elf lab4.bin
avr-objcopy -j .text -j .data -O srec lab4.elf lab4.srec
avr-objcopy -j .eeprom --change-section-lma .eeprom=0 -O ihex lab4.elf lab4_eeprom.hex
avr-objcopy: --change-section-lma .eeprom=0x0000000000000000 jamais utilisé
avr-objcopy -j .eeprom --change-section-lma .eeprom=0 -O binary lab4.elf lab4_eeprom.bin
avr-objcopy: --change-section-lma .eeprom=0x0000000000000000 jamais utilisé
avr-objcopy -j .eeprom --change-section-lma .eeprom=0 -O srec lab4.elf lab4_eeprom.srec
avr-objcopy: --change-section-lma_.eeprom=0x0000000000000000 jamais utilisé
```

avr-gcc –g -> Generate debugging information
avr-gcc –g0 -> Generate no debugging information
avr-gcc –g1 -> Generate minimal debug information
avr-gcc –g3 -> Generate maximal debug information

This information is stored in an Executable Linkable Format: elf file

```
amine@ubuntu:~/opt/lab4/lab4_v2$ make program
avrdude -p m128 -c usbasp -e -U flash:w:lab4.hex

avrdude: warning: cannot set sck period. please check for usbasp firmware update.
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9702 (probably m128)
avrdude: erasing chip
avrdude: warning: cannot set sck period. please check for usbasp firmware update.
avrdude: reading input file "lab4.hex"
avrdude: input file lab4.hex auto detected as Intel Hex
avrdude: writing flash (5018 bytes):

Writing | ################################################## | 100% 3.05s

avrdude: 5018 bytes of flash written
avrdude: verifying flash memory against lab4.hex:
avrdude: load data flash data from input file lab4.hex:
avrdude: input file lab4.hex auto detected as Intel Hex
avrdude: input file lab4.hex contains 5018 bytes
```

What is GNU De-Bugger (GDB)?

- Program that allows to view the source code as it executes
  - Make your program stop under specified conditions
  - Examine why your program crashed
  - Change values in your program
- More flexible than the usual IDE
  - IDE Debuggers are based on gdb with a restrained instruction list
  - Support community

What can GDB do ?

- Gives all the possible information regarding your system at any time
- Allows you to see everything your microcontroller sees
- Control the execution of your program

Basic instructions

- list linenumber: prints out the 10 lines around the line number specified
- step: executes one instruction at a time
- next: executes one line at a time
- until linenumber: executes the program until a specified line number
- print expression: prints a value, could be a variable
- info register: prints the value of a specific register
- backtrace: view the call frame

Conditional interruptions

- break instruction: sets a breakpoint in the program, could be for a line, when a function is called, can be associated with a condition
- watch expression: sets a watchpoint, program stops when the expression is called or changed, can be associated with a condition
  - Example: watch variable>10

How it could be useful ?

- When the code is not running as expected, track line per line what is happening
  - Catch every change of values of your flags and analyze the behavior of the system
- Check the register values to see if data was read by the ADC, SPI, TWI
- See the changes in PORTA and PORTB when it's being initialized differently for a peripheral
- Find the line that makes your program crash
  - Debug traps can give information on the type of error
- View register values to analyze the behavior of your peripherals
  - Example: if a button is seen as pushed

Bibliography and useful links

- http://winavr.sourceforge.net/AVR-GDB_and_AVaRICE_Guide.pdf
- http://luniks.net/avr-debug.jsp
- https://sourceware.org/gdb/current/onlinedocs/gdb/
- https://web.eecs.umich.edu/~sugih/pointers/summary.html
- https://www.youtube.com/watch?v=xQ0ONbt-qPs&t=200s
- https://www.youtube.com/watch?v=sCtY--xRUyI&t=1s

Thank you for listening !
Merci pour votre attention !