Assembler-Debugging mit SSH und GDB(or CGDB)

Das UNIX-Programm GDB – der GNU Debugger – ist der De-facto-Standard-Debugger von Linux-Systemen und wurde vom GNU-Projekt entwickelt. Er unterstützt Programmiersprachen wie Assembler, C, C++, Java, Go usw. Mittels SSH kann man sich direkt auf dem Raspberry-Pi einloggen und GDB benutzen, um den Programmablauf zu kontrollieren und Programmfehler im Assembler-Programm im Kommandozeilenmode zu erkennen.

I. Compilieren des Programms für Debugging

Mit der -g Option des GCC-Compilers können zusätzliche Debug-Informationen für GDB erzeugt werden. Damit kann man einige effektive Debugging-Eigenschaften nutzen, wie Anzeige der Quellcodes, Einstellung der Haltepunkte einfach mit Zeilennummer, usw.

II. Durch SSH auf Rasp einloggen

```
$ ssh user_name@ip_addr (mit -X Option hat man GUI Unterstützung)
```

III. Debugging mit GDB

1. Wegen des Memory-mapping wird Root-Recht benötigt für Debugging.

```
$ sudo gdb programm
```

2. häufigste notwendige GDB Befehle für Assembler-Debugging

- start
  legt einen temporären Haltepunkt in main Funktion und beginnt mit der Ausführung eines Programms unter GDB.

- l(list) + Zeilennummer(nur mit -g kompiliertes Programm gültig)
  Anzeige der Quellencodes, die in der Nähe von bestimmter Zeile liegen.

- disas(disassemble)
  disassemble eine bestimmte Funktion oder einen Funktions Fragment.

- break(breakpoint)
  legt einen Haltepunkt in einer Funktion. Das Argument kann mit Zeilen-nummer oder Adresse nach Disassmble gestellt.

  z.B break 30 (nur bei mit -g kompiliertem Programm gültig)
  break *0x000083ac

- nexti
  Ausführen des nächsten Assemblerbefehls(instruction). Funktionen werden übersprungen.

- stepi
  Ausführen des nächsten Assemblerbefehls(instruction). Es wird in Funktionen gesprungen, die im Befehl aufgerufen werden.

- continue
  Fortsetzung der Programmausführung bis zum nächsten Haltepunkt.

- finish
    Finish der Ausführung einer Funktion

- info registers oder info registers name_register
    Anzeige der Informationen aller Register oder eines bestimmten Registers

IV. Beispiel

1. Debugging-Prozess starten

```
➔ button_led_control_plus  sudo gdb button_led
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 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 "arm-linux-gnueabihf".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/pi/src/button_led_control_plus/button_led...done.
(gdb) start
Temporary breakpoint 1 at 0x8504: file main.S, line 32.
Starting program: /home/pi/src/button_led_control_plus/button_led

Temporary breakpoint 1, main () at main.S:32
32        push {r4-r11, lr}
(gdb)
```

2. Anzeige des Quellcodes

```
(gdb) list 32
27
28      /* main function */
29      .global main
30      .func main
31      main:
32        push {r4-r11, lr}
33
34      /* address mapping using c function */
35      mapping_addr:
36        bl map_peripheral
(gdb)
```

3. Dissasemble von Quellcodes

```
(gdb) disas
Dump of assembler code for function mapping_addr:
=> 0x00008508 <+0>:     bl      0x8604 <map_peripheral>
   0x0000850c <+4>:     mov     r9, r0
   0x00008510 <+8>:     mvn     r1, #0
   0x00008514 <+12>:    cmp     r0, r1
   0x00008518 <+16>:    bne     0x8528 <gpio_settings>
   0x0000851c <+20>:    ldr     r0, [pc, #144]  ; 0x85b4 <end_main+8>
   0x00008520 <+24>:    bl      0x83e0
   0x00008524 <+28>:    b       0x85ac <end_main>
End of assembler dump.
```

4. Einstellung von Haltepunkten

```
(gdb) l 69
64
65      /* main loop */
66      main_loop:
67
68        /* the change of programm-status should be triggered
69         * by falling edge of the button-input: when there is 10
70         */
71
72        /* get button-input( on:1, off:0 ) */
73        mov r0, r9
(gdb) break 69
Breakpoint 2 at 0x8548: file main.S, line 69.
(gdb)
```

5. Debugging mit nexti, stepi sowie finish

```
(gdb) disas
Dump of assembler code for function main_loop:
   0x00008544 <+0>:     mov     r0, r9
   0x00008548 <+4>:     bl      0x85d8 <get_pin_level>
=> 0x0000854c <+8>:     cmp     r0, #0
   0x00008550 <+12>:    beq     0x8588 <wait>
End of assembler dump.
(gdb) nexti
76        beq wait
(gdb) nexti

Breakpoint 2, main_loop () at main.S:74
74        bl get_pin_level
(gdb) nexti
75        cmp r0, #0  /* no button input signal -> keep current status */
(gdb) nexti
76        beq wait
(gdb) nexti

Breakpoint 2, main_loop () at main.S:74
74        bl get_pin_level
(gdb) stepi
get_pin_level () at gpio_functions.S:64
64        ldr r1, [r0, #(52 + 4 * (btnPin / 32))]
(gdb) disas
Dump of assembler code for function get_pin_level:
=> 0x000085d8 <+0>:     ldr     r1, [r0, #52]   ; 0x34
   0x000085dc <+4>:     lsr     r1, r1, #15
   0x000085e0 <+8>:     and     r1, r1, #1
   0x000085e4 <+12>:    mov     r0, r1
   0x000085e8 <+16>:    bx      lr
End of assembler dump.
(gdb) finish
Run till exit from #0  get_pin_level () at gpio_functions.S:64
main_loop () at main.S:75
75        cmp r0, #0  /* no button input signal -> keep current status */
(gdb)
```

6. Anzeige von Registerinformationen

```
(gdb) info registers r0 r1 r2 pc lr cpsr
r0             0x76ff6000       1996447744
r1             0x1      1
r2             0x76ff6000       1996447744
pc             0x8558   0x8558 <check_falling_edge+4>
lr             0x855c   34140
cpsr           0x60000010       1610612752
(gdb)
```

```
(gdb) info registers
r0             0x76ff6000       1996447744
r1             0x1      1
r2             0x76ff6000       1996447744
r3             0x76ff6000       1996447744
r4             0x0      0
r5             0x0      0
r6             0x8458   33880
r7             0x0      0
r8             0x0      0
r9             0x76ff6000       1996447744
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x7efffa34       0x7efffa34
lr             0x855c   34140
pc             0x8558   0x8558 <check_falling_edge+4>
cpsr           0x60000010       1610612752
```

V. Debugging mit CGDB

CGDB ist ein (Terminal-basiertes) Interface zum GNU Debugger (GDB). Zusätzlich zur Standard-GDB-Konsole, bietet CGDB eine geteilte Bildschirmansicht, die den Quellcode während des Debugging zeigt.

Auf Raspbian kann man einfach mit apt CGDB installieren.

```
$ sudo apt-get install cgdb
```

Beispiel: