Debugging in embedded en native systemen met GDB

A.M. ten Doesschate

July 15, 2015

Abstract

Een korte beschrijving :

- intro
- gebruik met welke tools en hulpmiddelen
- van de GDB setup en een summier aantal GDB commando's
- van native en embedded systemen debugging (specifiek : ARM)

1 Introductie

waarom GDB ?

- “printf” alike functie kan maar tot een bepaald punt debuggen (bijv. werkt nauwelijks tot niet voor interrupts)
- “prinf” telkens in code van target inbouwen
- hardware exceptions (segmentation faults, hard fault, etc)
- interface (API) voor hardware protocollen (JTAG, BDM, RSP)
- support voor native als remote (ethernet, serieel, etc)
- support server en client component
- draait onder diverse OS-en
- uniform voor embedded (target) en host (native)
- support voor diverse programmeer talen

2 Tools en hulpmiddelen

- opzet van fundamentele documentatie m.n. interface in multidisciplines
- gcc en gdb op native OS (GNU/Linux)
- compiler (gcc) : compile met optie “-g”
- source code listing

Mijn omgeving (back-to-the-basics) :

- alle tools : commandline georiënteerd
- vi
- Makefiles
- linker loader scripts (voordeel t.o.v. command line : vastzetten van secties. Alleen nuttig in embedded systemen !)
- gcc compiler suite voor host en ARM (specifiek : cortex-m3 van NXP)
- JTAG converter : J-Link van Segger (education versie)
- kleine adapter t.b.v. J-Link en target (bevat ISP en 20-polige ARM jtag konnektor)

3 GDB tool

GDB in 2 stappen :

- initialisatie vanuit file (zgn. ".gdbinit")
- commando's uitvoeren vanuit een shell (interactief)

3.1 .gdbinit

file ".gdbinit"
voorbeelden :

- debugging van native tools met argumenten
- verbinding leggen met IC dongles (bijv. JTAG via tcp/ip)
- flashen/programmeren van microcontroller
- displayen van registers (in embedded omgeving)
- zetten van stack en program counter
- intermix van programmeertaal (C) en assembly

3.2 GDB commando's (interactief)

- start van :
  debuggen van host applicatie (al of niet met .gdbinit) :
  - : gdb <host applicatie >
  - : zet eventuele breakpoints
  - : type "run" in de gdb shell

  debuggen van embedded systeem applicatie :
  - arm-none-eabi-gdb -se <firmware-microcontroller >
  - : zet eventuele breakpoints
  - type "cont" in de gdb shell
- disp /format <variabele >: display variable in format
  voorbeelden :
  - disp /s string : display string in ascii format (NULL terminated)
  - disp /x var : display var in hex format
  - disp /x *p : display pointer info (zonder "*" : geeft adres weer)
  - disp /x *p.member : display member in ptr naar struct
    Opm : het nummer van dit display commando geeft index in de display lijst aan
  - undisp <n >: uitzetten van displayen van variable waarbij <n >= index in de display lijst
- step (extra features) en next :
  - stepi : single step instruction machine code
  - step : (zonder i) single step van programmeer taal instructie
  - step(i) <n >: voert "n" instructies uit
  - n(ext) : als bij step maar bij een functie wordt functie uitgevoerd
    tot bij eerstvolgende instructie van programmeertaal
  - n(ext) <n >: idem dito maar dan <n >keer steppen
- <enter >: voert laatste gdb commando uit
- bt : backtrace, geeft aan waar de hierarchie van de functie is
- breakpoints :
  - br(eak) <file >:<regelnr >: set breakpoint
    op regelnummer.
    Opm : het nummer van dit breakpoint geeft index in de lijst van breakpoints aan
  - watch : doet hetzelfde maar dan op variabelen
  - del <brknr >: delete break/watch point nummer(s)
    Mag ook een reeks ofreeksen zijn bijv : "del 1 2 3" of "del 1-4 5 6"
  - info br(eakpoints) : geeft aan hoeveel en waar de breakpoints (file + regelnummer) staan
  - conditioneel breakpoint zetten bijv : br <file >: <regelnr >if i >= 6
    na breakpoint verder met commando "cont"
  - cont : verder doorgaan met laten draaien van app na een breakpoint stop
  - cont >n <: >n <aantal keren doorgaan alvorens stop bij een breakpoint (zinvol bij een loop)

4 Demo

- host debugging
- embedded debugging met ARM board (praktisch)
- debugging op native en embedded samen