

```
#include <zephyr/kernel.h>
#include <stdint.h>
```

Ohjelma

```
void foo(volatile uint32_t arg){
    printk("%u\n",arg);

    return;
}

int main(void){
    volatile uint32_t var = 39;
    if(var > 20){
        printk("Hello world\n");
    }
    for(int i = 0; i < 10; ++i){
        foo(var++);
    }

    return 0;
}
```

prj.conf

`CONFIG_DEBUG_OPTIMIZATIONS=y`

Build configurationissa taitaa elää nimellä “Optimize for debugging”

Käännä ja flashaa joko vscoden kautta tai manuaalisesti:

`west -b nrf5340dk_nrf5340_cpuapp_ns && west flash --erase`

GDB:n lataaminen

- <https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads>
- Windowsilla etsi sieltä uusin AArch32 bare-metal target windows toolchain lataus, luultavasti ylin
- Alempana sivulta pitäisi löytyä macOS .pkg tiedosto
- Linuxille voit joko ladata ylläolevalta sivulta tai paketinhallinnasta (suositus)
- Debianilla paketti **gdb-multiarch**, joka sisältää version ARM tuella
- Archilla **extra/arm-none-eabi-gdb**

GDB:n yhdistäminen

- J-Linkin työkalujen mukana tulee JLinkGDBServer.exe tai JLinkGDBServerExe, jos käytät Linuxia.
- Kyseessä on graafinen työkalu, joka näyttää suunnilleen tältä.
- Ikkuna on melko pitkä, joten jos koneessasi ei satu olemaan FHD näyttöä, voit valita telnet port kohdan, painaa tabulaattoria kahdesti ja enterillä valita OK-napin. Linuxilla ikkunaa pitäisi pystyä rullaamaan.
- Varmista, että ylin vaihtoehto on USB.
- Target device on asetettu nrf5340_xxAA_APP, Little Endian.
- Target interface: SWD
- Voit vaihtaa portit halutessasi, mutta tälle ei pitäisi olla tarvetta

The screenshot shows the 'Connection to J-Link' dialog box. It is configured with the following settings:

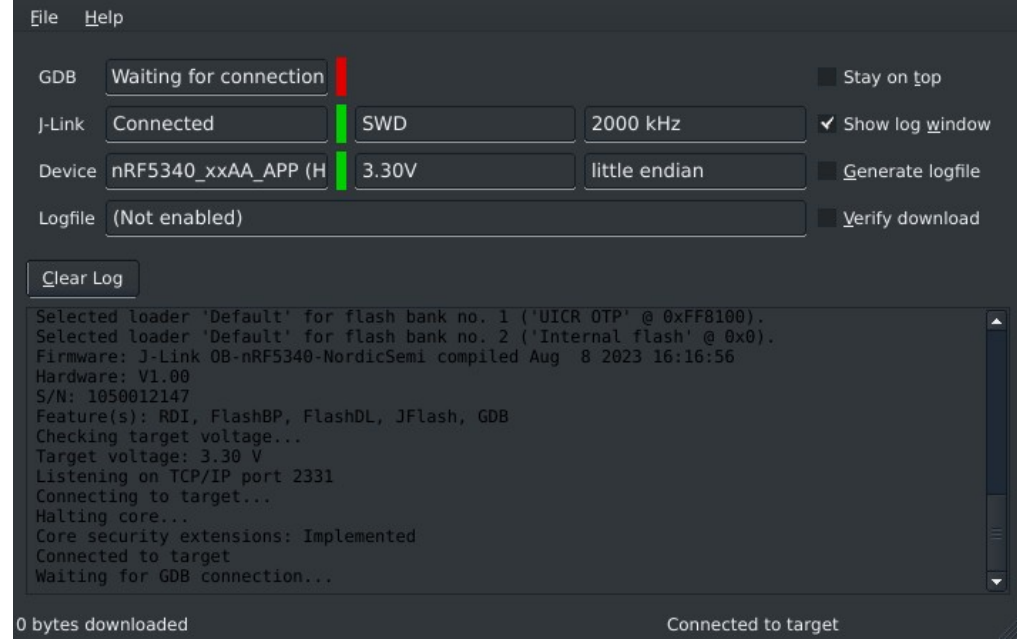
- Connection:** USB (selected), Serial No. (empty)
- Target device:** nRF5340_xxAA_APP, Little Endian
- Flash banks:**

BaseAddr	Name	Loader
0x00FF8000	UICR User	Default
0x00FF8100	UICR OTP	Default
0x00000000	Internal flash	Default
- Target interface:** SWD, Speed: Fixed 4000 kHz
- Script file (optional):** (empty)
- Server settings:** Init registers (checked), Localhost only (checked), Generate logfile (checked), GDB port 2331, SWO port 2328, Telnet port 2339
- Command line option:**

```
-select USB -device nRF5340_xxAA_APP -endian little -if SWD -speed 4000 -noir -noLocalhostOnly -nologtofile -port 2331 -SWOPort 2328 -TelnetPort 2339
```

Buttons: OK, Cancel

- Etsi ARM GNU Toolchainin mukana tullut **arm-none-eabi-gdb.exe**
- Linuxilla/Macilla se pitäisi löytyä suoraan komentolinjalta.
- Varmista, että kyseessä on **arm-none-eabi-gdb** eikä pelkkä **gdb**
- Varmista, että GDB server ei antanut mitään virheitä, jos saat virheen varmista, että sinulla ei jo ole aikaisempaa GDB serveriä käynnissä .
- Kun kaikki menee hyvin, GDB kohdassa pitäisi lukea Waiting for connection.
- Seuraavaksi gdb:ssä aja komento **file <projektikansio>/build/zephyr/zephyr.elf**
- Lopuksi aja komento **target remote :2331**, jos käytit jotain muuta porttia kuin 2331 GDB serverissä, käytä sitä myös tässä.



```
(gdb) file nrfapps/opetusdemo/build/zephyr/zephyr.elf
Reading symbols from nrfapps/opetusdemo/build/zephyr/zephyr.elf...
(gdb) target remote :2331
Remote debugging using :2331
0x00010f98 in _skip_0 ()
    at /home/j/ncs/zephyr/arch/arm/core/aarch32/cpu_idle.S:129
129      _sleep_if_allowed wfi
(gdb)
```

Komentoja

- Muutamia yleisempiä komentoja
- Lisää tietoa saat jostain komennosta komennolla **help <komento>** jossa <komento> on joku toinen komento
- Internet on myös hyödyllinen työkalu ja gdb on laajasti käytetty, joten Lontoon murteella tietoa löytyy paljon kun vain hakee “gdb how to x”

disassemble / disass

- Komento disassemble "purkaa" tietyn muistialueen
- Käytännössä aina annat sille funktion nimen esim. disassemble main, disassemble foo
- disassemble /s näyttää mukana alkuperäisen C-kielisen lähdekoodin
- disassemble /r näyttää raa'at konekieliset käskyt heksana

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x0001043c <+0>:      push    {r3, lr}
   0x0001043e <+2>:      ldr     r0, [pc, #16]    @ (0x10450 <main+20>)
   0x00010440 <+4>:      bl      0x14282 <printfk>
   0x00010444 <+8>:      movs   r0, #39 @ 0x27
   0x00010446 <+10>:     bl      0x10424 <foo>
   0x0001044a <+14>:     movs   r0, #0
   0x0001044c <+16>:     pop     {r3, pc}
   0x0001044e <+18>:     nop
   0x00010450 <+20>:     ldrsh   r4, [r5, r2]
   0x00010452 <+22>:     movs   r1, r0
End of assembler dump.
```

break

break <PAIKKA> -- asettaa breakpointin kohtaan <PAIKKA>

Rakkaalla komennolla on monta muotoa: brea, bre, br, b

<PAIKKA> voi olla lähdekoodin rivinumero tai *muistiosoite. Huomioi '*' muistiosoitteiden kanssa.

Esimerkiksi

break main -- asettaa breakpointin main() funktion alkuun

break *0x1044 -- asettaa breakpointin muistiosoitteessa 0x1044 olevaan käskyyn

break *main+2 -- asettaa breakpointin kohtaan main() funktion alku + 2 tavua

info breakpoints, info b, i b -- näyttää nykyiset breakpointit

d Num -- poistaa breakpointin Num

jump

jump <PAIKKA> -- hyppää kohtaa <PAIKKA> ja jatka suoritusta
lyhyt muoto j

<PAIKKA> voi olla rivinumero tai *muistiosoite. Huomioi tässäkin '*' muistiosoitteiden kanssa.

Toimii hyvin breakpointtien kanssa

Esimerkiksi, ohjelma on jo aloittanut suorituksen, mutta haluamme aloittaa mainin alusta ja keskeyttää suorituksen.

break main -- asettaa breakpointin main() alkuun
jump main -- hyppää main() alkuun ja osuu breakpointtiin heti

next(i), step(i) ja continue

- Kun ohjelmamme on pysäytetty, on usein hyödyllistä askeltaa sitä.
- next-komento askeltaa yhden *rivin lähdekoodia* kerrallaan, mutta ei astu funktioiden sisälle vaan suorittaa ne ja pysähtyy kutsun jälkeiselle riville
- step-komento askeltaa yhden *rivin lähdekoodia* kerrallaan, mutta astuu funktioiden sisälle.
- nexti ja stepi toimivat samalla tavoin funktioiden osalta, mutta askeltavat yhden *käskyn* kerrallaan
- continue jatkaa suoritusta kunnes osutaan breakpointtiin
- Lyhyet muodot n, s, ni, si ja c

print

print /FMT <mitä>
lyhyt muoto p

FMT voi olla x = heksa, d = desimaali, t = binääri
täysi lista: **help print**

<mitä> voi olla muistiosoite, rekisteri, muuttuja.

Tärkeä ero x komenttoon, voi tulostaa muistiosoitteista vain 32 bittiä kerrallaan.
Hyödyllisin käyttötarkoitus on muuttujien arvojen tarkistaminen tai rekisteriarvojen tarkistaminen.

Esimerkiksi

print var – tulosta muuttujan var arvo, muoto riippuu itse muuttujasta

print /x &var – tulosta muuttujan var osoite heksadesimaalina

X

x /FMT addr

Käytetään muistiosoitteiden tutkimiseen.

FMT kohdassa määritellään paljonko ja missä muodossa halutaan lukea ja tulostaa.

<numero> + b = byte, h = halfword, w = word ja g = giant

x = heksa, t = binääri, d = desimaali

Täysi lista **help x**

Esimerkiksi

x /4**bx** 0x23012 – lukee **neljä** **tavua** osoitteesta 0x23012 ja tulostaa ne **heksana**

x /2**hd** 0x301fd – lukee **kaksi** **puolisanaa** osoitteesta 0x301fd ja tulostaa ne **desimaalina**

TUI

- Layout asm
- Ctrl-x + o
- Ctrl-x + 2

```

Register group: all
r0 0x1 1 r1 0x0 0 r2 0x0 0 r3 0x1 1 r4 0x13be9 00165
r5 0x0 0 r6 0x0 0 r7 0x0 0 r8 0x0 0 r9 0x0 0
r10 0x0 0 r11 0x0 0 r12 0xd0bdab06 3582691014 sp 0x20000cb0 0x20000cb0 lr 0x1200b 75275
pc 0x10faf 0x10faf xpsr 0x21000000 533640128 msp 0x20000cb0 0x20000cb0 msp_ns 0x20000b08 0x20000b08
psp_ns 0x20000cb0 0x20000cb0 msp_s 0x20000300 0x20000300 psp 0x20000cb0 0x20000cb0 basepri 0x0 0
faultmask 0x0 0 control 0x2 2 apsr 0x20000000 536070912 epcr 0x10000000 16777216
lpsr 0x20000000 536040128 lpsr 0x10000000 16777216 mspim 0x20000308 536040128 mspim_ns 0x20000b08 536040128
msplm_s 0x20000400 536071936 msplm_ns 0x20000308 536040128 pspim_s 0x20002fe0 536083168 pspim_ns 0x20000b08 536040128
s0 0 (raw 0x00000000) s1 0 (raw 0x00000000) s2 0 (raw 0x00000000) s3 0 (raw 0x00000000) s4 0 (raw 0x00000000)
s5 0 (raw 0x00000000) s6 0 (raw 0x00000000) s7 0 (raw 0x00000000) s8 0 (raw 0x00000000) s9 0 (raw 0x00000000)
s10 0 (raw 0x00000000) s11 0 (raw 0x00000000) s12 0 (raw 0x00000000) s13 0 (raw 0x00000000) s14 0 (raw 0x00000000)
s15 0 (raw 0x00000000) s16 0 (raw 0x00000000) s17 0 (raw 0x00000000) s18 0 (raw 0x00000000) s19 0 (raw 0x00000000)
s20 0 (raw 0x00000000) s21 0 (raw 0x00000000) s22 0 (raw 0x00000000) s23 0 (raw 0x00000000) s24 0 (raw 0x00000000)
s25 0 (raw 0x00000000) s26 0 (raw 0x00000000) s27 0 (raw 0x00000000) s28 0 (raw 0x00000000) s29 0 (raw 0x00000000)
s30 0 (raw 0x00000000) s31 0 (raw 0x00000000) d0 0 (raw 0x0000000000000000) d1 0 (raw 0x0000000000000000) d2 0 (raw 0x0000000000000000)
d3 0 (raw 0x0000000000000000) d4 0 (raw 0x0000000000000000) d5 0 (raw 0x0000000000000000) d6 0 (raw 0x0000000000000000) d7 0 (raw 0x0000000000000000)
d8 0 (raw 0x0000000000000000) d9 0 (raw 0x0000000000000000) d10 0 (raw 0x0000000000000000) d11 0 (raw 0x0000000000000000) d12 0 (raw 0x0000000000000000)
d13 0 (raw 0x0000000000000000) d14 0 (raw 0x0000000000000000) d15 0 (raw 0x0000000000000000)

> 0x10faf.ldmia.w.spl, {r0, lr}
0x10faf.cpsie l
0x10faf.tsb sy
0x10faf.bs lr
0x10faf.cpsid l
0x10faf.mrr c1, r1
0x10faf.mrr BASEPRI, r1
0x10faf.push {r0, lr}
0x10faf.bl 0x10faf
0x10faf.cmp r0, #0
0x10faf.beqz 0x10faf
0x10faf.dsb sy
0x10faf.wfe
0x10faf.ldmia.w.spl, {r0, lr}
0x10faf.mrr BASEPRI, r0
0x10faf.cpsie l
0x10faf.bs lr
0x10faf.cmp r0, #0
0x10faf.bit.n #0, r0
0x10faf.and.w r1, r0, #1
0x10faf.lrrs r0, r0, #1
0x10faf.mrrs c1, #0
0x10faf.lsls c1, r1
0x10faf.ldr c1, [c1, #0] @ (0x10faf)
0x10faf.str.w c1, [r0, lsl #1]
0x10faf.db lr
0x10faf.b.n 0x11374
0x10faf.b.n 0x10faf

```