# Ohjelma

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

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”

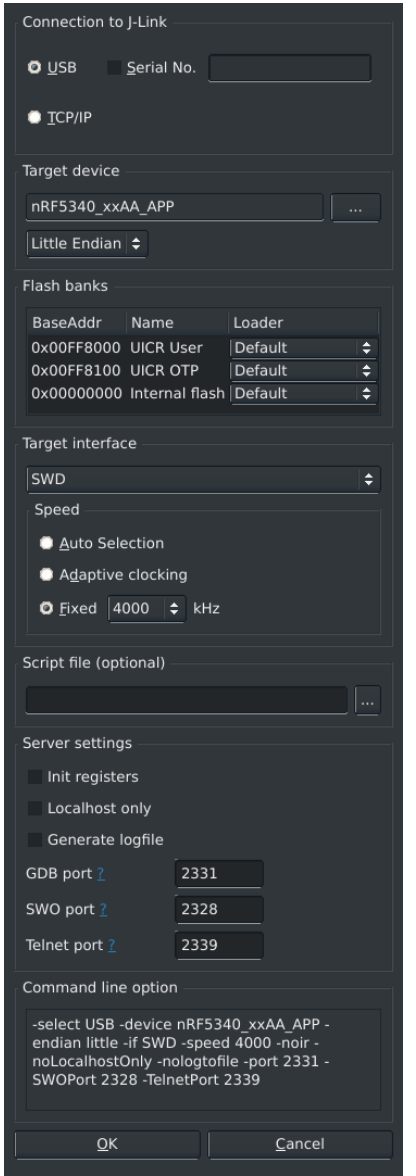
Käännä ja flashaa joko vscoden kautta tai manuaalisesti:
west -b nrf5340dk_nrf5340_cpuapp_ns && west flash --erase

# GDB:n lataaminen

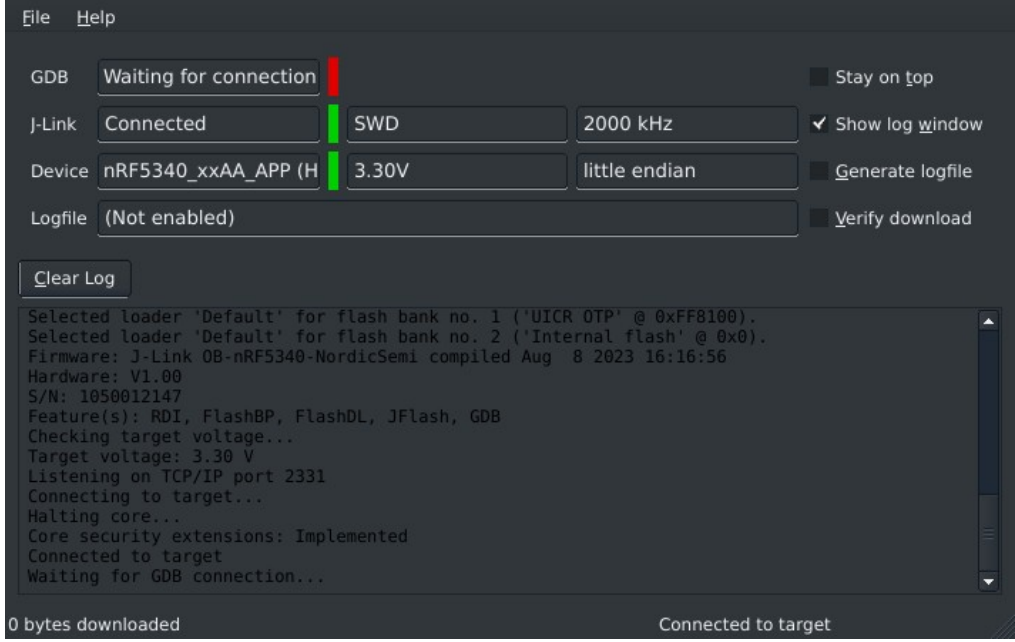
- https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads
- Windowsilla etsi sieltä uusin AArch32 bare-metal target windows toolchain lataus, luultavasti ylin
- Alempana sivulta pitäisi löytyä macOS .pkg tiedosto
- Linuxille voit joko ladata ylläolevalta sivulta tai paketinhallinnasta (suositus)
- 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



- 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 <printk>
   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 komentoon, 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 /4bx 0x23012 – lukee neljä tavua osoitteesta 0x23012 ja tulostaa ne heksana

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

# TUI

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