SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Zephyr RTOS

Marko Čuljak

Voditelj: Hrvoje Džapo

Zagreb, travanj, 2023.

**Sadržaj**

[1. Uvod 1](#_Toc132925679)

[2. Zašto Zephyr RTOS? 3](#_Toc132925680)

[3. Osnove Zephyr RTOS-a 4](#_Toc132925681)

[4. Razvoj aplikacije temeljene na Zephyr RTOS-u 5](#_Toc132925682)

[5. Zaključak 6](#_Toc132925683)

[6. Sažetak 7](#_Toc132925684)

[7. Literatura 8](#_Toc132925685)

# Uvod

Operacijski sustavi opće namjene su osmišljeni za upravljanje hardverskim resursima računala i pokretanju aplikacija koje se izvršavaju na računalu. Dizajnirani su za vremenski nekritične sustave. OS opće namjene mogu istovremeno (prividno istovremeno) izvršavati više zadataka i aplikacija, no nisu namjenjeni za važne, vremenski osjetljive aplikacije, već imaju vremensku fleksibilnost i mogućnost (TODO bolja rijec) da se neki zadaci ne izvrše ili kasne s izvršavanjem. Takve operacijske sustave susrećemo na osobnim računalima te neki od primjera takvih mnogobrojnih operacijskih sustava su: Windows,Linux, macOS [1].

Osim njih, u kontekstu ugradbenih računalnih sustava, puno više se susrećemo s operacijskim sustavima za rad u stvarnome vremenu. Takvi sustavi pružaju deterministički, strog odgovor potreban u stvarnome vremenu s ciljem brže reakcije na različite događaje. Vremenska ograničenja su točno određena te ih se mora strogo poštivati. To uglavnom znači, pogotovo u sustavima s manje procesorske moći kao što su ugradbeni računalni sustavi, malo vremena za obradu i pružanje odgovora na događaj. Sustav u stvarnome vremenu mora u zadanom vremenskom okviru odraditi traženi zadatak inače će sustav zakazati. Primjenu takvih sustava s operacijskim sustavima za rad u stvarnome vremenu nalazimo u sustavima upravljanja (kontrole leta), automobilima („*anti-lock brakes, air bags*“), medicinskim i oružanim sustavima gdje svako programsko čekanje ili odgoda može značiti snažne posljedice za sigurnost ili slično [2].

Dok su operacijski sustavi opće namjene prigodni za opće korisnike i potrebe ljudi, kao igranje igrica, slušanje glazbe, gledanje videozapisa i slično, operacijski sustavi za rad u stvarnom vremenu su osmišljeni za sustave gdje sustav mora izvršiti ili dati odgovor u kratkom, unaprijed definiranom vremenskom periodu te moraju biti brzi i precizni, a najveća primjena im je u ugradbenim računalnim sustavima koji su kombinacija hardvera i softvera dizajnirana za specifičnu namjenu te mogu biti dio veće cjeline (sustava).

Operacijski sustavi za rad u stvarnom vremenu uobičajeno pružaju mogućnost:

* višezadaćnosti - mogućnost izvođenja više zadataka brzom izmjenom tako da se steče dojam istovremenog izvršavanja više zadataka
* prioritiziranja zadataka
* dovoljnom broju prekida, odnosno izvora prekida

Neke od karakteristika operacijskih sustava za rad u stvarnom vremenu su [1]:

* malo zauzeće memorije, pogotovo u usporedbi s OS opće namjene
* visoke „*performanse*“ – brzi i responzivni
* deterministični
* visoka sigurnost
* raspoređivanje po prioritetima

Neki od poznatih operacijskih sustava za rad u stvarnom vremenu su Azure RTOS (ThreadX), embOS, VxWorks. Svi oni naplaćuju svoju uslugu te nisu projekti otvorenoga koda. NuttX je jedan primjer besplatnog RTOS-a te otvorenog koda. No daleko najpoznatiji, pogotovo među početnicima u ugradbenim sustavima jest FreeRTOS koji je pod MIT licencom, besplatan je za korištenje te je projekt otvorenog koda. On je poznat po svojoj jednostavnosti i malome zauzeću memorije. Ima i različite derivacije kao što su Amazon FreeRTOS (uključuje biblioteke za podršku Interneta stvari) i SAFERTOS koji je prilagođen za sustave ovisnima o visokoj sigurnosti.

Međutim, u ovome radu pričat će se o rastućem operacijskom sustavu za rad u stvarnome vremenu imenom Zephyr RTOS, a ima potencijal biti budućnost operacijskih sustava u ugradbenim računalnim sustavima.

# Zašto Zephyr RTOS?

Kada smo se bolje upoznali s operacijskim sustavima za rad u stvarnome vremenu, možemo ući dublje u razloge što čini Zephyr RTOS jedinstvenim (i boljim) od ostalih dostupnih operacijskih sustava kao što su FreeRTOS i Azure RTOS.

Prva stvar koja se kod ističe kod Zephyr-a je činjenica da je to projekt otvorenoga koda, što znači da stvatko može pristupiti, koristiti i pridonijeti unaprjeđenju projekta te time ohrabruje ljude i čitavu zajednicu inženjera da neprekidno unaprjeđuju Zephyr. Većina ostalih operacijskih sustava su poduprijeti različitim tvrtkama i zbog toga nisu potpuno neovisni. FreeRTOS je podprijet od strane Amazona, Azure RTOS od Microsofta. *Zephyr Project*-om upravlja Linux zaklada. To je neprofitna organizacija čije je misija stvoriti održivi ekosustav oko projekata otvorenog koda te tako ubrzati razvojni ciklus tehnologije i komercijalnih aplikacija. Velike tvrtke poput *Google*-a i *Facebook*-a svoje nade polažu u Zephyr. Čak je i *Nordic Semiconductors* odlučio migrirati svoje nove proizvode od *nRF5 SDK*-a i *SoftDevice*-ova na Zephyr. Velika razlika u veličini baze koda i doprinosima (za nekoliko redova veličine) može se vidjeti na slici 2.1. Može se vidjeti da je broj doprinosa Zephyr-u znatno veći od već velikih i popularnih operacijskih sustava poput FreeRTOS-a i Azure RTOS-a, s tim da Azure RTOS ima jasno vidljivo daleko najmanje pridnosa.

A picture containing text, screenshot, plot, diagram

Description automatically generated

Slika 2.1: Usporedba broja doprinosa na GitHub repozitoriju između raznih RTOS-a [4]

Druga velika prednost Zephyr-a je u sigurnosti i pouzdanosti. Zephyr ima vrlo otpornu i sigurnu jezgru koja podržava nekoliko algoritama za raspoređivanje vremena, zaštitu memorije i upravljanje iznimkama. Zephyr dolazi i sa sigurnosnim značajkama poput kriptografije, sigurnog pokretanja te ažuriranja *firmvera [6]*. Zephyr je trenutno sigurnosno certificiran PSA razinom 1 (od 3), a u taj certifikat imaju i FreeRTOS i Azure RTOS te ostale *softverske* platforme.

Zephyr podržava velik broj arhitektura i instrukcija (uključujući ARM, x86, RISC-V). Trenutno ima preko 450 podržanih pločica različitih arhitektura i proizvođača. To postiže ugledajući se na Linux te kopiranjem koncepta *devicetree*-ova kako bi pojednostavili podržavanje novih pločica i pločica po mjeri. On omogućuje pisanje istoga izvornog koda za STM, Espressif, *Nordic* pločice i ostale. Cjelokupno omogućavanje rada aplikacije s jedne arhitekture ili pločice je jednostavno te se može napraviti uređujući samo *devicetree* koji opisuje sklopovlje i dodatne hardverske komponente, bez potrebe za diranjem izvornog koda aplikacije.

Još jedna vrlo važna stvar koja daje prednost Zephyr-u ispred ostalih operacijskih sustava jest ogroman broj napisanih i implementiranih *driver libraries*-a. Zephyr ima ugrađena sučelja za upravljačke programe uređaja (*drivere*) i ogromne, odlično dokumentiranje biblioteke koda za podršku pregršt raznih senzora i uređaja. One *drivere* za koje nema podršku je opet puno lakše implementirati jer postoje SPI i I2C i ostali slični *driveri* na kojima se mogu ti novi *driveri* temeljiti. Velika stvar kod Zephyr-a su i već implementirani i certificirani komunikacijski (protokolni) stogovi za *Bluetooth Low Energy, IP stack, Wi-Fi, BLE, LoRaWAN.* Osim toga, ima i modul za *logging,* slično kao i kod ESP-a, podršku i implementacije za LVGL (pri pravljenju bogatih grafičkih sučelja) te podržavaju i C++ kod. Upravljački programi i ostale hardverske konfiguracije i inicijalizacije sklopovlja se uglavnom i izvršavaju automatski prije ulaska u samu aplikaciju te se programer ne mora brinuti o tome.

Za razliku od FreeRTOS-a koji se nudi samo u okruženju operacijskog sustava, Zephyr dolazi s podrškom za razne alate koji pomažu s upravljanjem projektima, ovisnostima, *toolchain*-ovima te *build* sustavom. To je sve integrirano u alatu naziva „*west“*.

Dok FreeRTOS nudi manji broj mogućnosti kao sami operacijski sustav i manje funkcionalosti, ipak zauzima manji prostor od Zephyr-a. Inače, FreeRTOS jezgra zauzima oko 5 do 10 kilobajta ROM-a. S druge strane, Zephyr, iako zauzima više, ne zauzima puno veću memoriju od FreeRTOS-a. Tipično zauzeće Zephyr-a je 8 do 12 kilobajta *flash* memorije, odnosno na 8 kilobajta *flash* memorije uspješno stane. Detaljnije se može vidjeti u tablici 2.1. Podešavanjem konfiguracije, memorija Zephyr-a može stati u čak 3 kB. Razlog zašto bi to napravili, a nemamo pristup mnogim resursima je na primjer podrška za različite *drivere* ili sigurnosne mehanizme operacijskih sustava.

|  |  |
| --- | --- |
| *Multithreading enabled* | 8 kB |
| *Multithreading enabled, no preemption* | 8 kB |
| *Multithreading enabled, no preemption, timers disabled* | 4 kB |
| *Multithreading disabled, timers enabled* | 5 kB |
| *Multithreading disabled, timers disabled* | 3 kB |

Tablica 2.1: Zauzeće memorije Zephyr RTOS-a s različitim konfiguracijama

# Osnove Zephyr RTOS-a

Kako bi se počeo rad sa Zephyr-om potrebne su tri glavne stvari (iako nisu sasvim nužne, preporučeno je radi jednostavnosti i praktičnosti imati ih).

To su:

* West („*Zephyr's meta-tool“*) – svestrani alat naredbenog retka za upravljanje Zephyr projektima
* Zephyr SDK – sadrži alate (*toolchains*) za svaki od različitih arhitektura koje Zephyr podržava. On dodatno uključuje i alate kao prilagođeni QEMU i OpenOCD [7]
* Zephyrov projektni kod – izvorni kod cijelog projekta, uključujući pregršt upravljačkih programa te njihovih primijenjenih aplikacija, *devicetree* kodova za preko 450 pločica i mnogo više toga

## West (Zephyr-ov meta-alat)

West je vrlo moćan meta-alat korišten kao alat naredbenog retka za upravljanje cjelokupnim projektima temeljenim na Zephyr-u. Razvija se u vlastitoj kodnoj bazi (*Git* *repositoryju*), a omogućuje mnoge funkcionalnosti te pomaže pojednostaviti proces *buildanja, flashanja,* konfiguriranja i upravljanje Zephyr projektima te podržava nadogradnje (omogućuje korisniku da implementira vlastite naredbe i funkcionalnosti) [8]. Kao što je spomenuto, jedna od važnijih funkcionalnosti je podrška pri upravljanju ovisnostima projekta. Kako je Zephyr modularno napravljen, *west* pomaže pri upravljanju različitih modula korištenim u projektu (Zephyr jezgrom, upravljačkim programima uređaja i ostalih modula). Pomoću *west*-a je moguće automatski preuzeti i integrirati ovisnosti u projekt (slično kao i „*git submodules“*). To se radi pomoću datoteke koja se naziva „*west manifest“*. U projektima ju se prepoznaje pod nazivom west.yml, a služi za opis ovisnosti projekta o drugim projekatima i modulima. Još jedna važna značajka je podrška ostalih alata kao što su CMake i Python. *West-ov build* sustav je temeljen na CMake-u, a *west* uključuje i podršku za Python skripte koje mogu služiti za automatsko testiranje i „prženje“ firmvera na pločicu. Pomoću njega je moguće vrlo lagano *buildati* i *flashati* različite pločice različitih arhitektura, sve što je potrebno je u naredbenom retku specificirati određenu (podržanu) pločicu jednom riječju te *west* sam shvati kako uspjeti u tome (većinom). Ukratko, *west* je vrlo važan alat koji programerima na Zephyr projektima omogućuje jednostavniji razvojni ciklus cjelokupne aplikacije pružajući im podršku u postavi, konfiguriranju, upravljanju projekta i različitim ovisnostima, a uz to nudi i jednostavnije *buildanje i flashanje* uređaja.

## Vrste arhitektura Zephyr aplikacija

Pri razvoju Zephyr aplikacija, postoje tri topologije arhitekture koda projekta u ovisnosti o tome gdje se nalazi aplikacijski direktorij, a navedene su u tablici 3.1.

|  |  |  |
| --- | --- | --- |
| **Topologija** | **Aplikacijski tip** | **Lokacija app. direktorija** |
| T1 | *repository* | u primjerima Zephyr repositoryja |
| T2 | workspace | u direktoriju gdje je i Zephyr |
| T3 | *freestanding* | druge lokacije, odvojeno od Zephyr-a |

Tablica 3.1: Topologije Zephyr aplikacija [9]

### T1 topologija

A screen shot of a computer program

Description automatically generated with low confidence

Slika 3.1: T1 topologija

Na slici 3.1 prikazana je T1 topologija. Aplikacijski direktorij nalazi se unutar *zephyr/* direktorija.

### T2 topologija

A picture containing text, screenshot, font, design

Description automatically generated

Slika 3.2: T2 topologija

Na slici 3.2 prikazana je T2 topologija. Aplikacijski direktorij nalazi se u *applications*/ direktoriju pokraj *zephyr/* direktorija.

### T3 topologija

A screen shot of a computer program

Description automatically generated with medium confidence

Slika 3.3: T3 topologija

Na slici 3.3 prikazana je T3 topologija. Aplikacijski direktorij nalazi se sasvim odvojen od zephyr/ direktorija.

U poglavlju 4, razvijena aplikacija ima T2 topologiju gdje je aplikacijski direktorij pokraj *zephyr/* direktorija.

## „Buildanje“ i „flashanje“ aplikacije

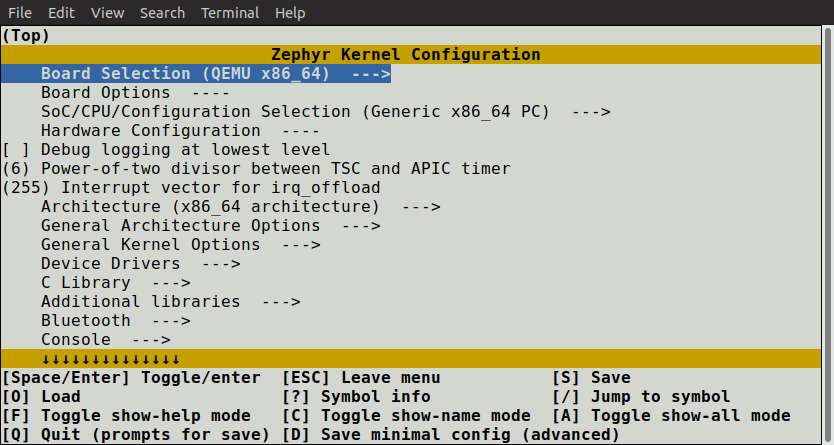
„*Buildanje*“ i „*flashanje“* aplikacije radi se pomoću *west* alata i naredbenog retka. Iako to nije jedini način, daleko je najjednostavniji. Zephyr uključuje svoj „*build*“ sustav temeljen na CMake-u te je integriran s *west* alatom. Dovoljno je samo u aplikacijskom direktoriju pokrenuti naredbu: *west build -b stm32f4\_disco -p* želimo li „*buildati“* aplikaciju za STM32F4 razvojni sustav. Ako želimo „*buildati“* za neku drugu pločicu, dovoljno je samo promijeniti taj parametar u naziv druge podržane pločice. Zastavica „-p“ nam govori da želimo očistiti cijeli *build/* direktorij prije naredbe za „*buildanje“*.

„*Flashanje“* hardvera na pločicu je isto znatno olakšano jer je sve već automatizirano i integrirano u *west* alatu. Kako Zephyr podržava mnoge različite arhitekture hardvera, proces „*flashanja“* ovisi o platformi koja se koristi. Na primjer, za STM32 koristit ćemo STLink ili JLinkExe, dok ćemo za Nordic koristiti nrfjprog. Također, podržani su i OpenOCD te pyOCD. Primjer naredbe za „*flashanje“*: *west flash* ili *west flash –runner jlink* (sa zastavicom koja govori da bismo za *„flashanje“* htjeli koristiti JLink).

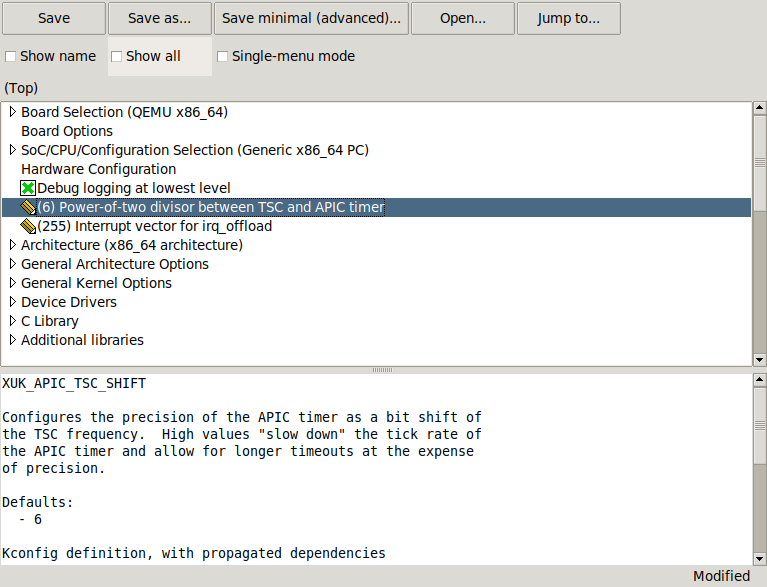
Korištenje *west-*a, Zephyrovog meta-alata znatno pojednostavljuje zadatak „*buildanja“* i *„flashanja“*.

## Kconfig

Zephyr se može konfigurirati u vremenu „*buildanja“* programa pomoću Kconfig-a. On omogućuje specificiranje različitih pločica, verzija hardvera, IC-ova i ostalih parametara (na primjer, promjena senzora ili protokola koji senzor koristi). Može mu se pristupiti iz naredbenog retka („*menuconfig*“, slika 3.4) te grafičkog sučelja („*guiconfig“,* slika 3.5). Izlaz koji specificira vrijednosti svih konfiguracijskih parametara je spremljen u datoteci *autoconf.h [10]*.



Slika 3.4: Menuconfig prikaz



Slika 3.5: Guiconfig prikaz

## Devicetree

Sve na Zephyr-u je temeljeno na *devicetree* datotekama, inspirirano (ugradbenim) Linuxom. To je hijerarhijska struktura podataka korištena za opisivanje hardverske konfiguracije uređaja, uključujući informacije o procesoru, memoriji, periferijama, korištenim senzorima i ostalim komponentama. Te informacije koristi jezgra Zephyr-a te ostale komponente kako bi omogućile modularnost i portabilnost. Sve što je dovoljno za prenijeti kod na drugu platformu i arhitekturu je napisati *devicetree* datoteku. To znatno olakšava posao programerima te uvodi stalnost i standardizaciju preko mnogo različitih platformi. Još jedna od prednosti *devicetree*-ova u Zephyr-u jest to što je omogućeno automatsko konfiguriranje i inicijalizacija hardvera i periferija na uređaju. Kada se Zephyr pokrene, čita informacije iz *devicetree* datoteke te automatski konfigurira uređaj i tako smanjuje programerski posao. Zephyr može automatski podesiti funkcije za upravljanje iznimkama, DMA kontrolere i ostale periferije. U *devicetree-*u je sadržana informacija kako čitati i pisati u određene memorijske lokacije i kako postaviti različite upravljačke uređaje.

Četiri su značajne datoteke korištene za formiranje konačnih *devicetree* datoteka [11]:

* izvorne datoteke („*sources“*, .dts) – ova datoteka obično opisuje specifični hardver koji se nalazi u sklopu pločice/uređaja, uključuju jednu ili više .dtsi datoteka
* datoteke za uključivanje („*includes“*, .dtsi) – uobičajeno opisuju CPU ili SoC na kojemu se Zephyr nalazi te mogu uključivati druge .dtsi datoteke
* datoteka preklapanja (*„overlays“*, .overlay) – nadograđuju ili modificiraju izvornu datoteku pločice, one su isto DTS datoteke, sugeriraju da premošćuju .dts datoteke te nose veću važnost od njih (na primjer, mogu uključiti neku periferiju koja je isključena u izvornoj datoteci *devicetree*-a)
* datoteke za verifikaciju („*bindings“*, .yaml) – pisane u .yaml formatu (kompatibilan s Python skriptama koje ih koriste kako bi verificirale format *devicetree* datoteka), opisuju što se treba i smije nalaziti u *devicetree* izvornim datotekama.

Na slici 3.6 je vidljiv čitav proces obrade, provjere i generiranja konačnih datoteka koje koristi aplikacija. Izvorna datoteka (.dts) se zajedno s datotekom preklapanja („*overlay“*), koja je višeg prioriteta i premošćuje izvornu datoteku, spaja te preprocesira kako bi se zatim provela verifikacija izvornih datoteka korištenjem „*bindings“* te Zephyr-ovih skripti. Na kraju se proizvodi „zephyr.dts“ datoteka koja je spoj *„.dts“* i *„.overlay“* datoteke te predstavlja konačni *devicetree*. Osim nje, generiraju se i „*C header“* datoteke koje imaju nekoliko tisuća linija koda te sadrže sve informacije i parametre zapisane u konačnoj *devicetree* datoteci. Tim parametrima se pristupa pomoću mnogobrojnih „makroa“ ispisanih u „*devicetree.h“* datoteci.

A picture containing text, diagram, screenshot, plan

Description automatically generated

Slika 3.6: Proces dobivanja konačnih "header" datoteka iz devicetree-a

## Upravljački programi uređaja (*„device drivers“*)

Upravljački programi uređaja su još jedna stvar koja razlikuje Zephyr od ostalih operacijskih sustava.

Prvobitno, imaju veliku bazu različitih upravljačkih programa te njihovih implementacija koji se lako integriraju u vlastiti projekt i koji su svi napisani po određenome standardu i slijedeći jasne modele oblikovanja.

Zephyr-ov model upravljačkih programa jasno i konzistentno specificira model upravljačkog programa, stvari koje je potrebno implementirati te je odgovoran za inicijalizaciju i pokretanje svih upravljačkih programa u sustavu (postoji više razina inicijalizacije, a oni su: PRE\_KERNEL\_1 i 2, POST\_KERNEL, APPLICATION, SYS\_INIT).

Zephyr ima i potporu za više istovremenih instanci upravljačkih programa. Na primjer, možemo imati 5 različitih instanci istih temperaturnih senzora neovisnih jedno o drugome. Također, svaki upravljački program (UART, SPI, I2C) je podržan s generičkim tipovima pristupnih funkcija („*API“*). Svaki senzor koji za koji se piše upravljački program ili se koristi, ima jasno definirane obvezne pristupne funkcije koje se moraju implementirati (uz to, mogu se i vlastite funkcije implementirati).

A picture containing text, screenshot, parallel, rectangle

Description automatically generated

Slika 3.7: Model upravljačkog programa [12]

Svaki upravljački program je obvezan implementirati i popuniti „*device“* strukturu prikazanu na slici 3.8. „*Name“* parametar predstavlja naziv upravljačkog programa. „*Config“* član je za pohranu podataka koja će se samo čitati te se postavlja u „*build time“*-u (na primjer, na kojem je „*pin“-*u prekid za SPI ili nekakve senzore). „*Data“* struktura je spremljena u RAM te se u nju mogu pohraniti bilo kakvi parametri potrebni za vrijeme izvođenja programa (brojilo, redovi poruka i slično). Struktura „*api“* predstavlja pokazivače na funkcije koje upravljački program izlaže te se za određene upravljačke programe moraju implementirati njihove obvezne funkcije i staviti u tu strukturu.

A screenshot of a computer code

Description automatically generated with low confidence

Slika 3.8: "Device" struktura [12]

# Razvoj aplikacije temeljene na Zephyr RTOS-u

U sklopu ovoga rada, napravljena je i jednostavna „demo“ aplikacija temeljena na Zephyr RTOS-u koja ilustrira iznad opisane koncepte Zephyr-a, rad u njegovom ekosustavu te posebnosti korištenja. „Demo“ aplikacija se bavi implementacijom i korištenjem SHT3x senzora koji mjeri temperaturu i vlažnost zraka te njihov ispis na UART sučelje (konzolu) na tri mikrokontrolera različitih arhitektura i proizvođača.

## Sensirion SHT3x

Senzor korišten u ovome projektu je Sensiron SHT3x senzor koji ima mogućnost mjerenja vlage i temperature te je jedan od boljih i preciznijih senzora. Senzor može biti napajan s 3V ili 5V napajanjem, a posjeduje i I2C sučelje preko kojega se može pristupiti dvjema adresama (temperatura i vlažnost).

A purple circuit board with white text

Description automatically generated with low confidence

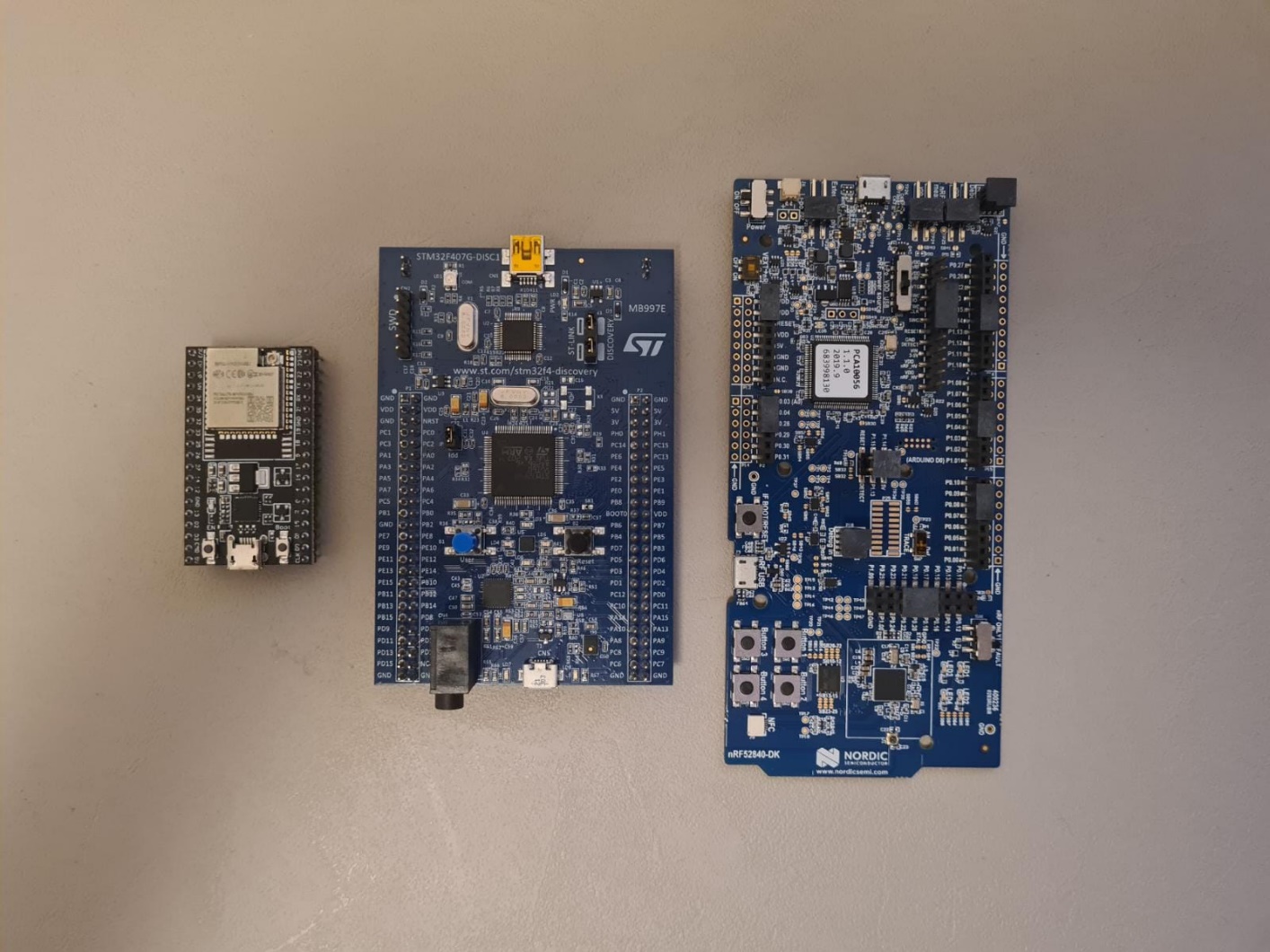
Slika 4.1: Sensirion SHT3x senzor [13]

## Korišteni mikrokontroleri (razvojni sustavi)

U sklopu ovoga rada, korištena su tri mikrokontrolera prikazana na slici 4.2:

* Espressifov ESP32-WROOM-32U
* STM32F407 razvojni sustav (*„discovery kit“*)
* Nordic Semiconductors nRF52840-DK razvojni sustav

Jedini zahtjev je da svi moraju imati I2C sučelje kako bi mogli pristupiti SHT3x senzoru, a to zadovoljavaju.



Slika 4.2: Korišteni mikokontroleri

## Razvoj aplikacije

Kao što je prije rečeno, jedna od velikih prednosti Zephyr operacijskog sustava je veliki broj napisanih upravljačkih uređaja koji se lagano implementiraju u vlastiti projekt zbog standardiziranog načina razvoja i pristupa tim upravljačkim programima uređaja. Vrijeme razvoja je bilo znatno smanjeno zbog već implementiranog upravljačkog programa za senzor SHT3x.

Topologija korištena za ovu aplikaciju je topologija T2, prikazana na slici 3.2. Razvijena aplikacija nalazi se odmah pored *zephyr/* direktorija koji je povučen s „*Github*“ poslužitelja pomoću *west* manifesta (*west.yml* datoteke) prikazane na slici 4.3. Dio „*remotes“* nam govori s kojeg *link*-a ćemo uvoziti dijelove koda. U „*projects“* dijelu specificiramo da želimo v3.3.0 Zephyr projekta te koje sve module želimo uvesti, a koje su nam potrebne za ovaj projekt.

A screen shot of a computer program

Description automatically generated with low confidence

Slika 4.3: Sadržaj west.yml datoteke

Osim izvornog koda aplikacije, najjednostavnija aplikacija mora sadržavati:

* *CMakeLists.txt* datoteku - ona *build* sustavu govori gdje se nalaze ostale aplikacijske datoteke te povezuje aplikacijski direktorij sa Zephyr-ovim CMake *build* sustavom (mora imati u sebi naredbu: *find\_package(Zephyr)* )
* Konfiguracijsku datoteku jezgre (*prj.conf,* slika 4.4) – *Kconfig* konfiguracijska datoteka koja zadaje konfiguracijske vrijednosti jezgre (u ovome slučaju preko nje omogućujemo I2C, senzor, izlaz na konzolu)

A black background with white text

Description automatically generated with low confidence

Slika 4.4: Sadržaj prj.conf

Za ovaj projekt, obzirom da koristimo 3 različite pločice različitih arhitektura i proizvođača, moramo prilagoditi i dodati naš senzor tim pločicama. To se radi preko uređivanja *devicetree*-a. U ovome slučaju, preko *BOARD.overlay* datoteka smo podesili parametre potrebne SHT3x senzoru (SDA, SCL *pinove*, adresu I2C i slično).

To nam je omogućilo da programski (izvorni) kod programa ostane sasvim jednak za sva tri razvojna sustava (mikrokontrolera). Odvajanjem i standardiziranjem struktura upravljačkih programa i opisa hardvera te periferija postigli smo jednostavnu i laganu portabilnost koda s jednog razvojnog sustava na drugi (i arhitekture) samo mijenjanjem četiri linije u *devicetree* datoteci.

A screen shot of a computer program

Description automatically generated with medium confidence

Slika 4.5: Izvorni kod glavnog programa

Na slici 4.5 prikazan je kod glavnog programa koji ima funkciju dohvaćanja vrijednosti temperature i vlage sa senzora te njihov ispis na serijsko sučelje.

U liniji broj 6 možemo vidjeti kako služeći se makro-om *devicetree*-a dohvaćamo jednu instancu senzora te ga pretvaramo u tip „*device“* koji ima unaprijed određenu strukturu.

U liniji broj 21, pomoću generičke *API* funkcije tražimo senzor da napravi jedno očitanje, a kasnije pozivom funkcije: *sensor\_channel\_get,* dohvaćamo dio već unaprijed propisanih enumeracija svih tipova vrijednosti koje neki senzor može imati (u ovome slučaju temperatura i vlažnost, a u slučaju akcelerometra to može biti akceleracija na X, Y i Z osi).

Kasnije tu vrijednost samo ispisujemo opet već implementiranom funkcijom ispis u „*double“* formatu.

*„Buildanje“* i pokretanje aplikacije se radi jednostavno za sva tri mikrokontrolera koristeći se *west* meta-alatom. Dovoljno je samo u naredbeni redak upisati naredbu: *west build -b „BOARD“ -p*

Parametar „BOARD“ je u našemu slučaju jedno od ponuđenih: esp32, stm32f4\_disco ili nrf52840dk\_nrf52840. Zatim se za kraj aplikacija *„flasha“* na pločicu naredbom: *west flash*.

# Zaključak

Među mnoštvom operacijskih sustava za rad u stvarnome vremenu u zadnje vrijeme se posebno ističe jedan – Zephyr RTOS. S činjenicom da je projekt otvorenog koda te da iza njega stoji Linux zaklada, jako je primamljiv u grupi gdje su ostali operacijski sustavi kao FreeRTOS i Azure RTOS poduprijeti tvrtkama kao Amazon i Microsoft. Zephyr RTOS može poslužiti kao odličan izbor za aplikacije sa strožim memorijskim zahtjevima, iako njegova snaga leži u skalabilnosti i mogućnosti prilagođavanja cijelog operacijskog sustava. Zbog svoje izuzetno dobro oblikovane, modularne arhitekture, programeri mogu lagano zamijeniti različite komponente bez potrebe za velikim promjenama projekta. To omogućava odličnu portabilnost i održavanje sustava kroz vrijeme. Još jedna od velikih prednosti Zephyr-a je podrška za mnoge protokole kao što su Wi-Fi, Ethernet, BLE i sl. Sa već razvijenim komunikacijskim (protokolnim) stogovima i mnoštvom napisanih modularnih i generičkih upravljačkih programa (*„device drivera*“), implementiranje nekog senzora nikada nije bilo lakše. Sve to zajedno ga čini vrlo dobrim izborom, pogotovo u vrijeme sve većeg razvoja povezanih uređaja (IoT) koji moraju komunicirati s ostalim uređajima ili s računalnim oblakom. Korištenjem *devicetree-*a u Zephyr-u pojednostavljen je proces prijenosa Zephyr projekata na nove hardverske platforme i arhitekture te tako smanjujući vrijeme potrebno za ručne konfiguracije. Tako se programeri mogu više usredotočiti na razvijanje aplikacija, umjesto zabrinutosti oko hardverskih konfiguracija. Zephyr naravno ima svojih nedostataka: strma krivulja učenja, teški i poražavajući početci, teže pronalaženje odgovora na pitanje zbog relativno činjenice da je relativno nov operacijski sustav, teškog otklanjanja grešaka zbog velike upotrebe makro-a, performansi u vrlo strogim vremenskim uvjetima i slično.

Sve u svemu, iako su početci jako teški te nekada traženje odgovora zna biti frustrirajuće, Zephyr RTOS je vrlo moćan, dobro posložen, siguran, pouzdan operacijski sustav potpuno otvorenog koda, dobre dokumentacije koda i aktivne zajednice. Programerima predstavlja i korak prema *ugradbenom Linux-u*, a rastom *Interneta stvari,* gdje će operacijski sustavi za rad u stvarnome vremenu biti vrlo važni, sve je izglednije da vrijedi slijediti primjer velikih firmi i „igrača“ na tržištu u podržavanju Zephyr RTOS-a ispred ostalih operacijskih sustava za rad u stvarnome vremenu.

# Literatura

1. *Real-Time Operating System*, Alexander S. Gillis, TechTarget, (2022), Poveznica: [*https://www.techtarget.com/searchdatacenter/definition/real-time-operating-system*](https://www.techtarget.com/searchdatacenter/definition/real-time-operating-system)*;* [pristupljeno 05. Travnja 2023.]
2. *What is a Real-Time Operating System (RTOS)?,* ni.com, (2022), Poveznica: <https://www.ni.com/en-rs/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-labview-real-time-module/what-is-a-real-time-operating-system--rtos--.html>; [pristupljeno 05. Travnja 2023.]
3. *Zephyr Project Documentation*, Zephyr Project, (2015). Poveznica: <https://docs.zephyrproject.org/latest/index.html>; [pristupljeno 05. Travnja 2023.]
4. *Why developers should choose Zephyr RTOS*, Byte Lab, (2022). Poveznica: <https://www.byte-lab.com/why-developers-should-choose-zephyr-rtos/>; [pristupljeno 05. Travnja 2023.]
5. *The Zephyr RTOS is awesome*, Florian Kromer, (2021). Poveznica: <https://medium.com/geekculture/the-zephyr-rtos-is-awesome-931bce3a695f>; [pristupljeno 05. Travnja 2023.]

1. <https://zephyrproject.org/why-we-moved-from-freertos-to-zephyr-rtos/#:~:text=While%20FreeRTOS%20is%20a%20popular,with%20over%20450%20supported%20boards>; [pristupljeno 25. Travnja 2023.]
2. *Zephyr Project Documentation*, Zephyr Project, (2015). Poveznica: <https://docs.zephyrproject.org/3.2.0/develop/toolchains/zephyr_sdk.html>; [pristupljeno 25. Travnja 2023.]
3. *Zephyr Project Documentation*, Zephyr Project, (2015). Poveznica: <https://docs.zephyrproject.org/latest/develop/west/index.html>; [pristupljeno 25. Travnja 2023.]
4. *Zephyr Project Documentation*, Zephyr Project, (2015). Poveznica: <https://docs.zephyrproject.org/latest/develop/application/index.html>; [pristupljeno 25. Travnja 2023.]
5. *Zephyr Project Documentation*, Zephyr Project, (2015). Poveznica: <https://docs.zephyrproject.org/2.7.0/guides/build/kconfig/menuconfig.html>; [pristupljeno 28. Travnja 2023.]
6. *Zephyr Project Documentation*, Zephyr Project, (2015). Poveznica: <https://docs.zephyrproject.org/3.2.0/build/dts/intro.html>; [pristupljeno 28. Travnja 2023.]
7. *Zephyr Project Documentation*, Zephyr Project, (2015). Poveznica: <https://docs.zephyrproject.org/3.1.0/kernel/drivers/index.html>; [pristupljeno 03. Svibnja 2023.]
8. *SHT3x Humidity & Temperature Sensor*, Stratasys Inc., (2023.). Poveznica: <https://grabcad.com/library/sht3x-humidity-temperature-sensor-1>; [pristupljeno 03. Svibnja 2023.]

# Sažetak

Operacijski sustavi za rad u stvarnome vremenu