Bare-metal programiranje u mikroprocesorskim sistemima

1. Pojam bare-metal programiranja

Bare-metal programiranje podrazumeva razvoj softvera koji se izvršava **direktno na hardveru**, bez operativnog sistema i bez apstraktnih softverskih slojeva kao što su drajveri viskog nivoa, middleware ili real-time operativni sistemi (RTOS). Programer ima **potpunu kontrolu nad mikroprocesorom**, njegovim registarskim mapama, prekidima, taktnim sistemom i periferijama.

U bare-metal pristupu, program počinje izvršavanje odmah nakon resetovanja mikroprocesora i nastavlja se u beskonačnoj petlji (`while(1)`), gde aplikacija direktno upravlja perifernim modulima.

---

2. Softverski slojevi u STM32 sistemima

Kod STM32 mikrokontrolera mogu se razlikovati sledeći nivoi:

1. **Hardver (MCU)**
    - CPU jezgro (Cortex-M4 kod STM32F401)
    - Periferije (GPIO, TIM, ADC, USART…)
    - Registarska mapa
2. **CMSIS (Cortex Microcontroller Software Interface Standard)**
    - Standardizovani header fajlovi
    - Definicije registara, prekida i osnovnih funkcija (`__WFI`, `__NOP`, SysTick)
    - Ne predstavlja biblioteku visokog nivoa
3. **HAL / LL / RTOS (opciono)**
    - Nisu deo bare-metal pristupa
    - Dodaju apstrakciju i olakšavaju razvoj, ali smanjuju kontrolu

U **bare-metal režimu koristi se samo CMSIS**, dok se HAL i LL ne koriste.

---

3. Startup proces mikroprocesora

Nakon uključenja napajanja ili resetovanja, STM32 prolazi kroz sledeće faze:

1. **Reset vektor** pokazuje na adresu početka programa
2. Izvršava se **startup fajl**
    - Inicijalizuje se stack pointer
    - Postavlja se vektorska tabela prekida
3. Poziva se funkcija `main()`

U bare-metal aplikaciji, `main()` predstavlja centralnu tačku programa i obično sadrži:

- inicijalizaciju periferija
- beskonačnu petlju sa aplikacionom logikom

---

4. Upravljanje perifernim modulima preko registara

Za razliku od HAL pristupa gde se koriste funkcije poput `HAL_GPIO_WritePin()`, u bare-metal programiranju periferijama se upravlja **direktnim upisom u registre**.

Primer:

- uključivanje takta za GPIO port
- podešavanje pina kao izlaznog
- upis logičke nule ili jedinice

Svaki periferni modul ima **registarsku mapu**, definisanu u CMSIS header fajlovima. Na taj način programer tačno zna:

- koji bit u kom registru utiče na određeno ponašanje hardvera
- kako softver utiče na električno stanje pina

Ovaj pristup omogućava:

- maksimalne performanse
- deterministično ponašanje sistema
- minimalan memorijski otisak

---

5. Koncept vremena u bare-metal sistemima

Pošto ne postoji operativni sistem, **vreme mora biti realizovano ručno**. Postoje dva osnovna pristupa:

5.1 Busy-wait kašnjenje

Program se “zadržava” u petlji određeni broj ciklusa procesora. Ovaj metod je jednostavan, ali:

- nije precizan
- blokira CPU (nema multitaskinga)

5.2 Prekidno bazirano merenje vremena (SysTick)

Cortex-M jezgra poseduju ugrađeni **SysTick timer**, koji može generisati periodične prekide (npr. svakih 1 ms).

Princip rada:

1. SysTick se konfiguriše da generiše prekid

2. U prekidnoj rutini se uvećava globalni brojač vremena
3. Aplikacija koristi taj brojač za merenje kašnjenja i periodičnih događaja

Ovaj pristup je:

- precizniji
- energetski efikasniji
- bliži realnim embedded sistemima

---

6. Prekidi u bare-metal sistemima

Prekid (interrupt) predstavlja mehanizam kojim hardver **privremeno prekida izvršavanje glavnog programa** kako bi se obradio hitan događaj.

U bare-metal pristupu:

- prekidne rutine su obične C funkcije
- njihova imena moraju tačno odgovarati vektorskoj tabeli
- programer mora ručno voditi računa o globalnim promenljivama i konkurentnosti

Primer: `SysTick_Handler()` je prekidna rutina koja se poziva periodično i služi kao osnova za softverski sat sistema.

---

7. Beskonačna petlja kao glavni tok programa

Za razliku od desktop aplikacija, embedded sistemi **nikada ne završavaju rad**. Glavna funkcija ima oblik:

```
int main(void)
{
   inicijalizacija();

   while (1) {
      aplikaciona_logika();
   }
}
```

Ova petlja predstavlja “srce” sistema i u njoj se:

- čitaju ulazi
- upravlja izlazima
- reaguje na događaje koje su pripremili prekidi

---

8. Prednosti i mane bare-metal pristupa

Prednosti:

- Potpuna kontrola nad hardverom
- Minimalna latencija
- Manja potrošnja memorije
- Idealno za učenje arhitekture MCU-a

Mane:

- Veći razvojni napor
- Više mogućnosti za greške
- Manje prenosiv kod
- Otežano skaliranje na kompleksne sisteme

---

9. Bare-metal kao osnova za razumevanje viših slojeva

Razumevanje bare-metal programiranja je **ključni preduslov** za rad sa:

- HAL i LL bibliotekama
- RTOS-ima (FreeRTOS)
- sigurnosnim mehanizmima
- real-time sistemima

Programer koji razume šta se dešava “ispod haube” lakše:

- debaguje sistem
- optimizuje performanse
- donosi ispravne arhitektonske odluke

10. Zaključak

Bare-metal programiranje predstavlja **fundamentalni nivo rada sa mikroprocesorskim sistemima**. Iako moderni alati nude visok nivo apstrakcije, razumevanje rada mikroprocesora na registarskom nivou ostaje nezamenjivo u obrazovanju i industriji.

Primer sa LED-om na NUCLEO-F401RE demonstrira osnovne koncepte:

- upravljanje GPIO modulom
- upotrebu SysTick tajmera
- rad sa prekidima
- beskonačnu petlju kao osnovu embedded aplikacije

Ovaj pristup predstavlja temelj za sve naprednije embedded sisteme.

11. Bare-metal tajmeri u mikroprocesorskim sistemima

Tajmeri predstavljaju jedan od najvažnijih perifernih modula u mikroprocesorskim sistemima. Njihova osnovna uloga je precizno merenje vremena, generisanje periodičnih događaja i sinhronizacija rada sistema. U bare-metal programiranju, tajmeri se koriste bez softverskih apstrakcija, direktnim pristupom registarskoj mapi mikroprocesora.

Za razliku od softverskih kašnjenja baziranih na praznim petljama (busy-wait), hardverski tajmeri rade nezavisno od izvršavanja glavnog programa. Oni broje impulse takta procesora ili perifernog takta i generišu događaj kada dostignu unapred definisanu vrednost.

Kod STM32 mikroprocesora, tajmeri se dele na osnovne, opšte namene i napredne tajmere. U bare-metal režimu, programer ručno konfiguriše svaki aspekt rada tajmera, uključujući izvor takta, preskaler, period i način generisanja prekida.

---

12. Princip rada bare-metal tajmera

Rad sa tajmerom u bare-metal sistemu zasniva se na sledećim koracima:

1. Uključivanje takta za odgovarajući tajmerski modul putem RCC registra
2. Podešavanje preskalera, kojim se deli ulazna frekvencija takta
3. Definisanje perioda brojanja pomoću auto-reload registra
4. Omogućavanje prekida ukoliko je potrebna reakcija procesora
5. Pokretanje tajmera

Kombinacijom preskalera i perioda moguće je ostvariti vrlo precizna vremenska kašnjenja, periodične događaje ili generisanje signala tačno definisane frekvencije.

Tajmeri se u bare-metal sistemima često koriste kao osnova za:

- periodično izvršavanje zadataka
- softverske satove i brojače vremena
- generisanje PWM signala
- merenje trajanja impulsa i frekvencije

---

13. Prekidni mehanizam kod tajmera

Kada tajmer dostigne definisanu vrednost, može generisati prekidni zahtev. Prekid omogućava mikroprocesoru da privremeno zaustavi izvršavanje glavnog programa i pređe na izvršavanje prekidne rutine.

U bare-metal pristupu, prekidna rutina je obična C funkcija čije ime mora odgovarati vektorskoj tabeli prekida. Programer mora ručno da:

- očisti prekidni flag

- obradi događaj
- obezbedi da se prekid ne ponavlja nekontrolisano

Ovaj mehanizam omogućava determinističko ponašanje sistema, jer se reakcija na događaj dešava tačno u trenutku kada tajmer to signalizira, bez zavisnosti od glavne programske petlje.

---

14. EXTI prekidi – reakcija na spoljašnje događaje

EXTI (External Interrupt) mehanizam omogućava mikroprocesoru da reaguje na promene logičkog stanja spoljašnjih pinova. Ovo je osnovni način obrade događaja koji dolaze iz okruženja, kao što su pritisak tastera, signal senzora ili promene digitalnih ulaza.

Za razliku od stalnog proveravanja stanja ulaza (polling), EXTI prekidi omogućavaju procesoru da ostane u stanju mirovanja ili da obavlja druge zadatke sve dok se ne desi konkretan događaj.

U bare-metal sistemima, EXTI konfiguracija zahteva ručno podešavanje:

- GPIO pina kao ulaznog
- mapiranje GPIO pina na odgovarajuću EXTI liniju
- izbor aktivne ivice (rastuća, opadajuća ili obe)
- omogućavanje prekida u NVIC kontroleru

---

15. Princip rada EXTI prekida

Kada se detektuje definisana promena na ulaznom pinu, EXTI modul generiše prekidni zahtev. CPU automatski:

1. čuva trenutno stanje izvršavanja
2. prelazi na izvršavanje EXTI prekidne rutine
3. nakon obrade događaja vraća se u glavni program

U prekidnoj rutini, programer mora obavezno da:

- identifikuje koja EXTI linija je izazvala prekid
- očisti odgovarajući pending flag
- izvrši minimalnu obradu događaja

Na ovaj način se postiže brza i efikasna reakcija sistema bez nepotrebnog opterećenja procesora.

---

16. Kombinovanje tajmera i EXTI prekida u bare-metal sistemima

Kombinacija hardverskih tajmera i EXTI prekida predstavlja osnovu real-time embedded sistema. Dok EXTI prekidi omogućavaju detekciju spoljašnjih događaja, tajmeri obezbeđuju preciznu vremensku osnovu za njihovu obradu.

Tipični primeri primene ove kombinacije uključuju:

- eliminaciju odbijanja kontakta tastera (debounce)
- vremensko ograničenu obradu spoljašnjih signala
- merenje vremena između dva događaja
- periodično uzorkovanje senzora

U bare-metal sistemima, ova kombinacija omogućava visok stepen determinističnosti i predvidljivosti ponašanja.

---

17. Prednosti bare-metal pristupa kod tajmera i prekida

Rad sa tajmerima i EXTI prekidima u bare-metal režimu ima nekoliko ključnih prednosti:

- minimalna latencija reakcije sistema
- potpuna kontrola nad hardverskim resursima
- jasno razumevanje odnosa između softvera i hardvera
- efikasno korišćenje procesorskog vremena

Ovaj pristup je naročito pogodan za edukaciju, jer studentima pruža jasan uvid u to kako mikroprocesor zaista funkcioniše na najnižem nivou.

---

18. Zaključak – uloga tajmera i EXTI prekida u bare-metal sistemima

Tajmeri i EXTI prekidi predstavljaju ključne gradivne elemente svakog mikroprocesorskog sistema. U bare-metal programiranju, oni omogućavaju realizaciju vremenski determinističkih i događajima vođenih aplikacija bez oslanjanja na složene softverske okvire.

Razumevanje ovih mehanizama omogućava programerima da:

- dizajniraju pouzdane real-time sisteme
- efikasno koriste hardverske resurse
- lakše pređu na složenije softverske arhitekture kao što su HAL biblioteke ili RTOS sistemi

Ovo znanje predstavlja prirodan nastavak osnovnih principa bare-metal programiranja i temelj za dalji rad u oblasti embedded sistema.