ODGOVORI NA PITANJA ZA VJEŽBU

PURS

2 Procesori i kontroleri

2.3 Pitanja za provjeru znanja

1. Objasniti razliku između procesora odnosno mikroprocesora i kontrolera odnosno mikrokontrolera?

- granica između P i uP jer u širini sabirnice(arhitektura) ili implementacije
- širina sabirnice kaže, 8 i 16 su mikro, ostale su obične
- prema implementaciji uP i uC imaju komponente u jednom integriranom sklopu
- razlika između procesora i kontrolera, kontroler ima pored CPU i ROM,RAM i periferije
- procesor ima CPU i / ili cache, a ROM,RAM su vanjski

2. Što se podrazumijeva pod pojmom mikro u nazivu mikroprocesor odnosno mikrokontroler?

- aritekturalno gledano – sabirnica
- implementacijski gledano – komponente na jednom integriranom sklopu

3. Koje izvedbe procesora postoje obzirom na implementacijske tehnologije? Kakve vrste procesora razlikujemo obzirom na područje primjene?

- s mekom jezgorom (soft core)
  - formalni model korištenjem jezika za opis sklopovlja (VHDL, verilog)
  - može se konfigurirati prije same implementacije
- fiksno ožičeni procesori (hard core)
  - fizička komponent.implement u VLSI tehnologiji
- područje primjene:
  - general purpose processors
  - embedded processors (RT, low energy consumption)
  - application processors (complex OS)

4. Što je System on Chip Design? Kakve vrste komponenata se mogu naći na istom chip-u? Koje su prednosti takvog načina dizajna? U kojim tehnologijama srećemo takav dizajn?

- SoC – više elektroničkih sustava / komponenti na fizički jednom integriranom sklopu
- vrste komponente:
  - analogne - pojač, regul.napajanj
  - digitalne – proces, mem, perif
  - analogno-digitalne – AD, DA pretv.

- prednosti:
  - cijena proizdovda
  - kraće vrijeme razvoja
  - manje problema s miješanjem tehnologija
- nailazimo ih u: general purpose microcontrolers, perilica rublja, FPGA...

5. Na primjeru ARM tehnologije, objasniti razliku između pojmova: arhitektura, procesorska jezgra i (υ)procesor odnosno (υ)kontroler.

• ARM arhitekturu
⇒ arhitektura definira programski model
⇒ u osnovi, arhitektura predstavlja ponašajni (*Behavioral*) model

• ARM procesorsku jezgru
⇒ procesor s mekom jezgrom koji implementira neku inačicu arhitekture

• ARM mikroprocesor ili mikrokontroler
⇒ fizička implementacija procesora s mekom jezgrom, koja može uključivati i dodatne komponente

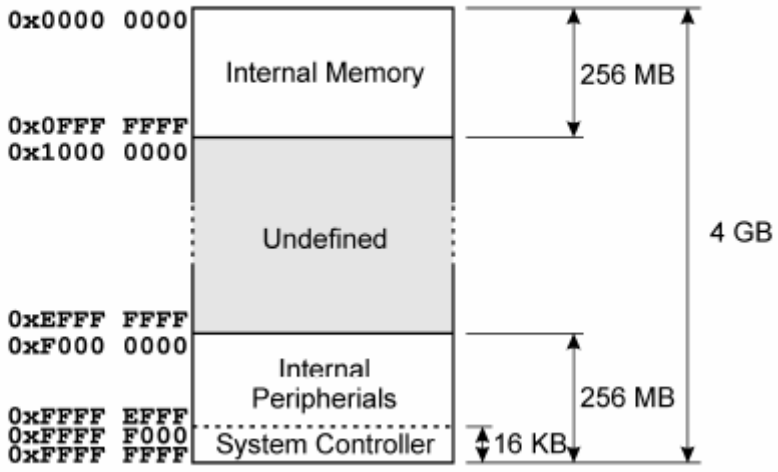
6. Opisati način označavanja ARM arhitektura.

inačica + ekstenzije

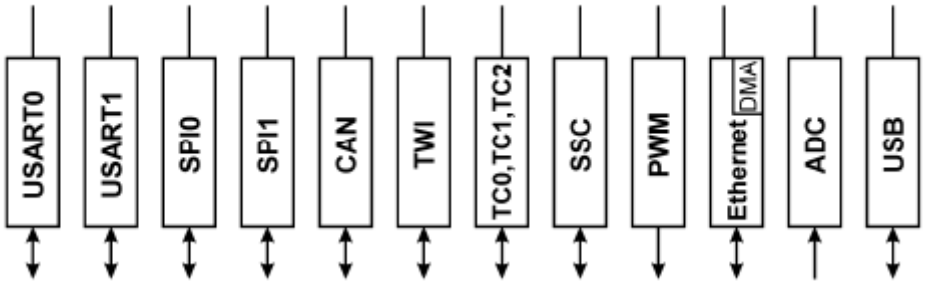
3 Mikrokontroleri familije AT91SAM7X

```
B labela     ; skoči bezuvjetno na lokaciju labela
BEQ labela   ; skoči ako je Z=0
BL labela    ; poziv potprograma
```

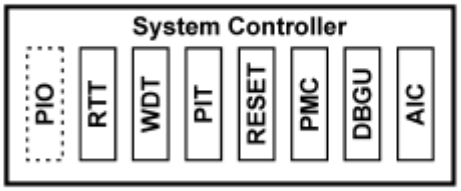
- adresni memorijski prostor



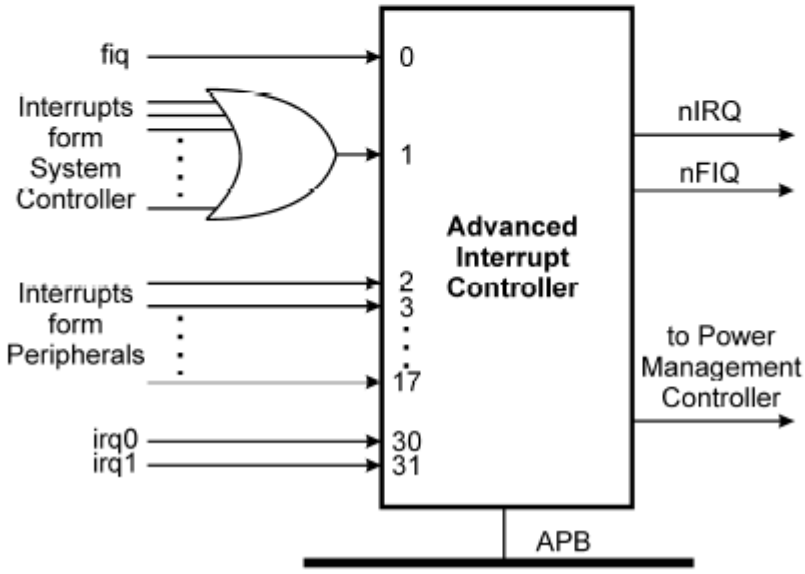
- periferije

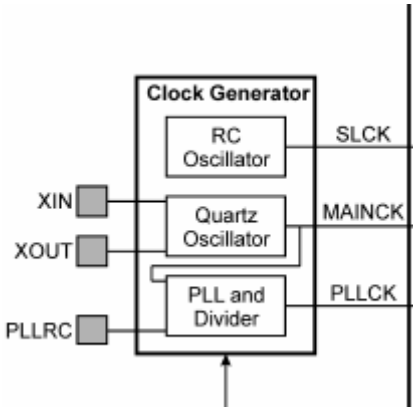


- *System Controller*



- sklop za upravljanje prekidima (AIC)
  ⇒omogućuje obradu zahtjeva koje daju do 32 jedinice
- kod mikrokontrolera SAM7X128, AIC obrađuje
  - 3 vanjska izvora prekida (vanjski priključci mikrokontrolera)
    ⇒IRQ0 (=30), IRQ1 (=31) i FIQ (=0)
  - 16 izvora prekida periferija
    ⇒POIA (=2), PIOB, SPI0, SPI1, USART0, USART1, SSC, PWMC, UDP, TC0, TC1, TC2, CAN, EMAC, ADC(=17)
  - 1 izvor prekida od upravljačkog sustava (*System Controller*) (=1)
    ⇒PIT, RTT, WDT, DBGU, PMC, RSTC, EFC

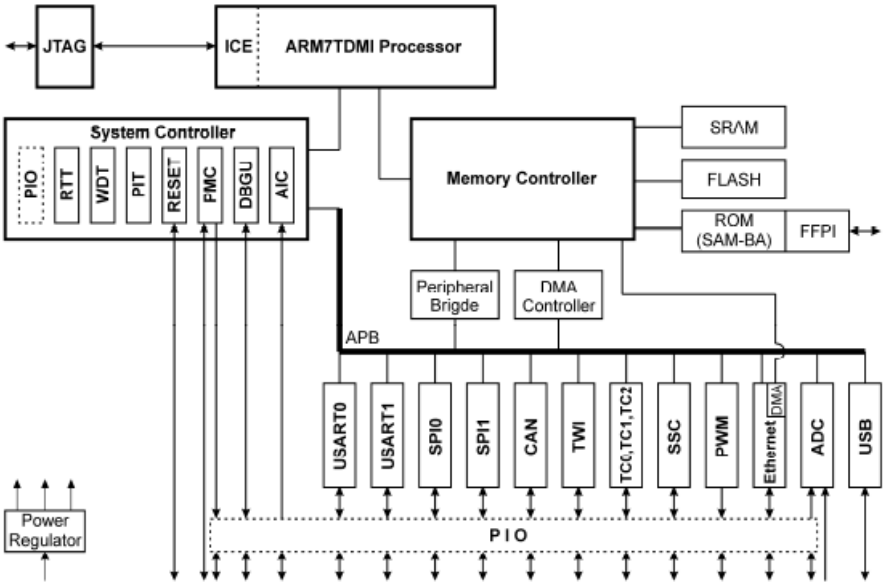




PMC – power managment controller

3.6 Pitanja za provjeru znanja

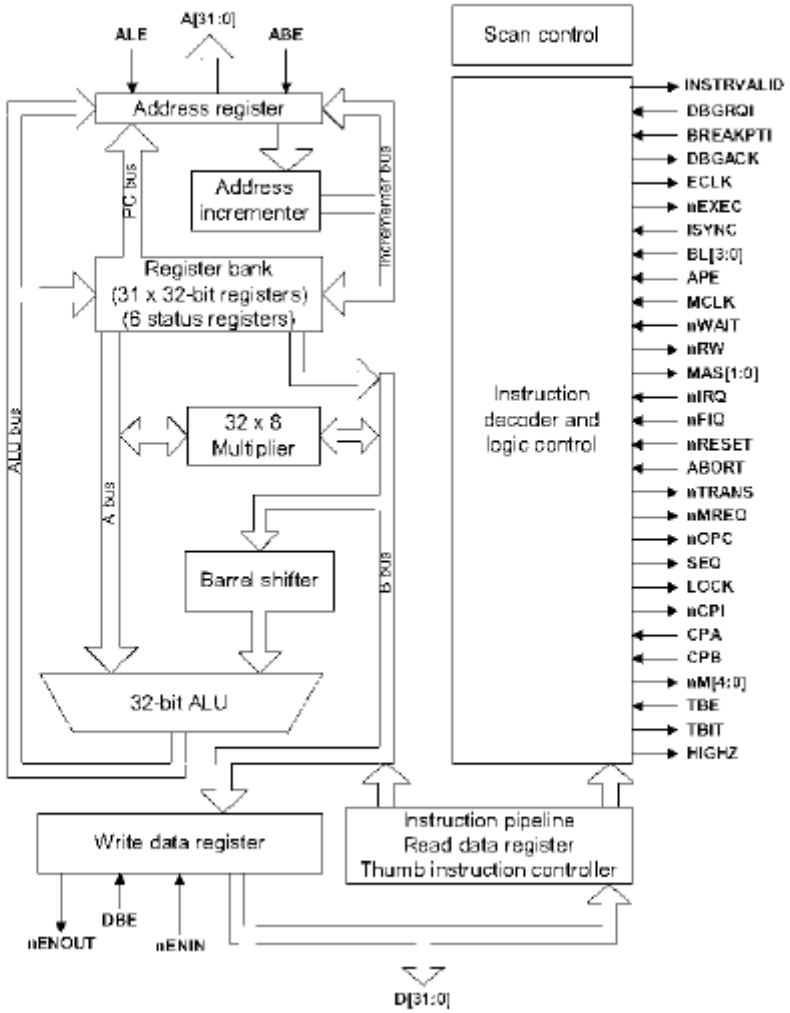
1. Nacrtati osnovnu blokovsku shemu mikrokontrolera familije AT97SAM7X i ukratko opisati funkciju pojedinih sklopova.



- sklopovi / komponente mikrokontrolera:
  - **procesor**
  - **sklopovlje za upravljanje memorijom** (*Memory Controller*) – dekodiranje adrese, mapiranje neke od memorija, arbitraža pristupa memoriji, rad s FLASH memorijom
  - **sklopovlje za upravljanje radom sustava** (*System Controller*) - (reset, real time timer, watch dog timer, periodit interval timer, reset, power managment controller, dbgu, advanced interrupt controller...)
  - **periferno sklopovlje** (*Peripherals*) – spojeno na sabirnicu APB, komunicira s okolinom preko priključaka mikrokontrolera
  - **unutrašnji izvori napajanja** (*Power Regulator*)

2. Nacrtati blokovsku shemu procesorske jezgre ARM7TDMI. Kakva je arhitektura ove procesorske jezgre obzirom na memoriju? Koje širine podataka koristi? Kakav zapis podataka u memoriji koristi? Što je poravnavanje?

- blokovska shema procesorske jezgre



- Von Neumannova arhitektura (zajednička progr. i podatk. memorija)
- radi s podacima širine(32, 16 i 8 bitova)
- podaci moraju biti poravnati – alignment, dakle npr. 32 bitni podatak = 4B, početi će od adrese koja je višekratnih broja 4 uključujući i 0 !
- koristi se little / big endian
- little endian (češće)– niži bajt na nižu adresu

3. Opisati registre opće namjene procesora ARM7TDMI. Navesti "bankirane" registre u pojedinim načinima rada. Koja je uloga registara R13, R14 i R15?

- ukupno 37 registara (svi su 32 bitni)
  - 31 registar opće namjene
  - 6 statusnih registara
- u praksi se koriste nazivi viši / niži registri
- bankirani registri – su registri koji se razlikuju ovisno o pojedinim načinima rada
- R13 – SP stack pointer, pokazivač na vrh stoga

- R14 – LR link register, u njoj se sprema povratna adresa potprograma ili programa
- R15 – PC, program counter, pokazuje na adresu sljedeće instrukcije

4. Opisati statusne registre procesora ARM7TDMI.

- Organizacija statusnih registara:

Neprivileg. način rada	Privilegirani način rada					
		Iznimke				
User	System	Supervisor	Abort	Indefined	Interrupt	Fast Interrupt
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

- postoji registar trenutnih stanja (CPSR) i registar za pohranu stanja (SPSR)
- registar trenutnih stanja, **CPSR** (Current Program Status Register)
  - vidljiv je u svim načinima rada
  - sadrži zastavice i upravljačke bitove
- registar za pohranu stanja, **SPSR** (Saved Program Status Register)
  - služi za pohranu stanja registra CPSR kod obrade iznimaka
  - svaki način rada ima svoju "kopiju" registra SPSR

raspored bitova registra CPSR

31	30	29	28	27-8	7	6	5	4-0
N	Z	C	V	Reserved	I	F	T	M[4:0]

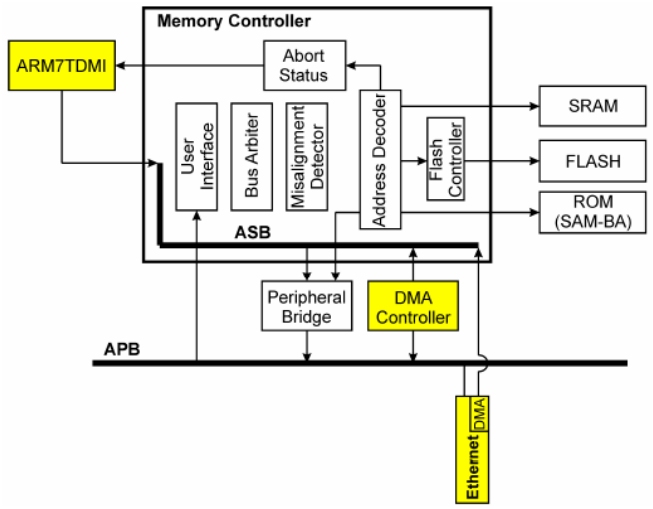
5. Objasniti razliku između ARM i Thumb skupova instrukcija. Čime je određeno koji skup instrukcija se trenutno koristi?

- jezgra podražava oba dva skupa instrukcija
- ARM 32 bita, Thumb 16 bita
- koji skup instrukcija se koristi definirano je bitom T u CPSR-u (thumb)
- 1=thumb, 0=ARM

6. Opisati iznimke kod procesora ARM7TDMI.

- nastaje kad se normalan tok programa mora privremeno prekinuti (reset, zahtjev za prekid, nepostojeća instrukcija)
- koristi se ARM skup instrukcija
- svaka iznimka definirana je svojim imenom, prekidnim vektorom, načinom rada u koji ulazi te izvorom iznimke
- IRQ i FIQ se mogu onemogućiti postavljanjem I=1 odnosno F=1
- prioriteti prekida, reset – najviši, SWI – nedefinirana instrukcija – najniži

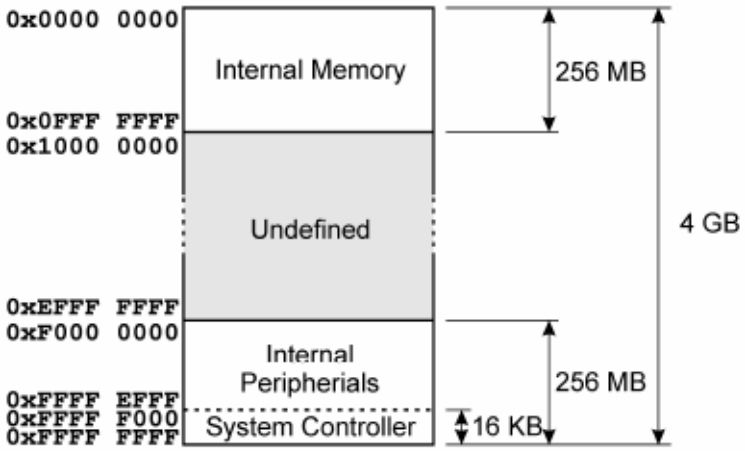
7. Nacrtati pojednostavljenu blokovsku shemu memorijskog kontrolera mikrokontrolera familije AT91SAM7. Ukratko opisati njegovu funkciju. Navesti potencijalne upravljače u sustavu.



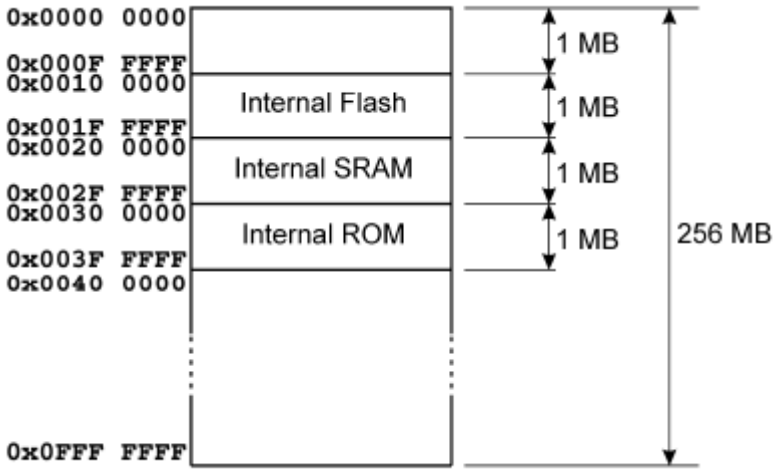
- funkcije :
  - dekodiranje adrese
  - mapiranje neke memorije u boot prostor
  - arbitražu među sklopovima (3) tko zahtjeva pristup memoriji
  - rad s flash memorijom
  - misalignment detector
- upravljači u sustavu (Bus master):
  - procesor
  - DMA kontroler
  - Ethernet MAC kontroler

8. Nacrtati memorijsku mapu mikrokontrolera AT91SAM7X i opisati mapiranje unutrašnjih memorija.

- adresni memorijski prostor



- unutrašnja memorija:
  - Sram, Flash i Rom

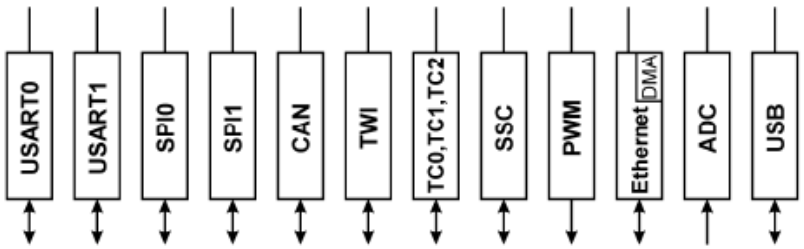


- prvi MB sadrži mapiranu jednu od memorija, što ovisi o
  - stanju zastavice GPNVM (General Purpose Non-Volatile Memory)
    ⇒ nalazi se u registru MC_FSR
  - komandi Remap koja je dana memorijskom kontroleru
    -remap se izvodi programski

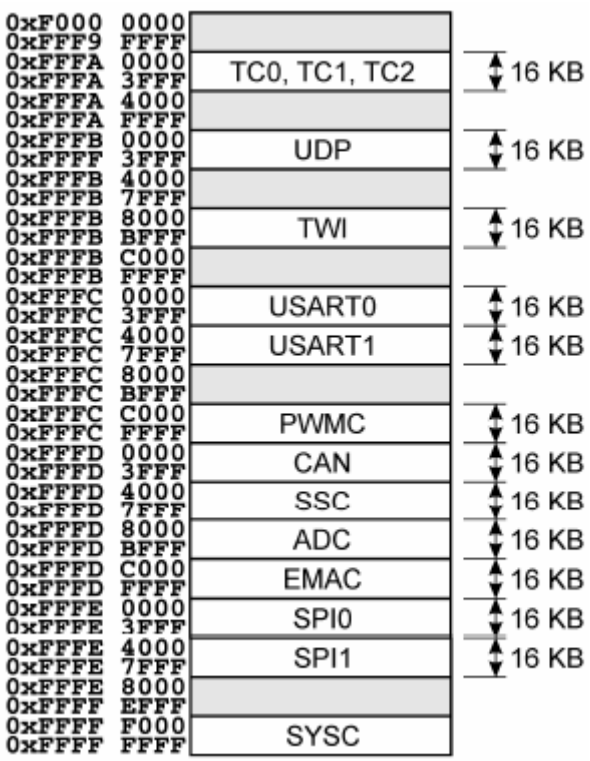
-defaultno bootanje je s ROMa, zato jer nakon brisanja FLASHa (ERASE) bit GPNVM=1

9. Opisati način adresiranja periferija. Gdje se nalazi memorijski prostor u kojem se vide periferije i upravljačko sklopovlje?

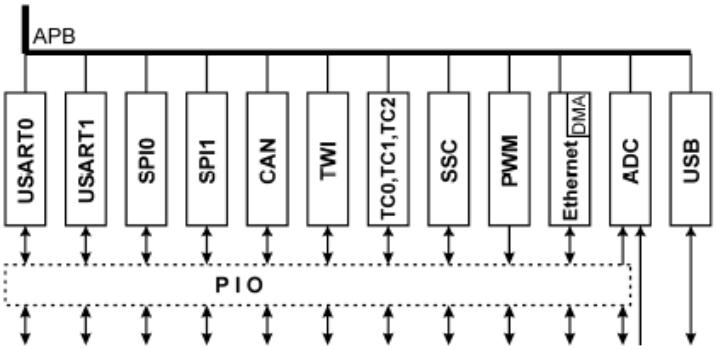
- periferije



- registri koji odgovaraju pojedinim periferijama vide se u vanjskom memorijskom prostoru
- početne adrese blokova koji pripadaju pojedinim sklopovima nazivaju se osnovne adrese (Base Address)
- uočiti
  - periferijama odgovaraju blokovi od 16 KB
  - mnogo lokacija je praznih
  - na vrhu memorije nalaze se registri upravljačkog sklopovlja (SYSC, System Controller)



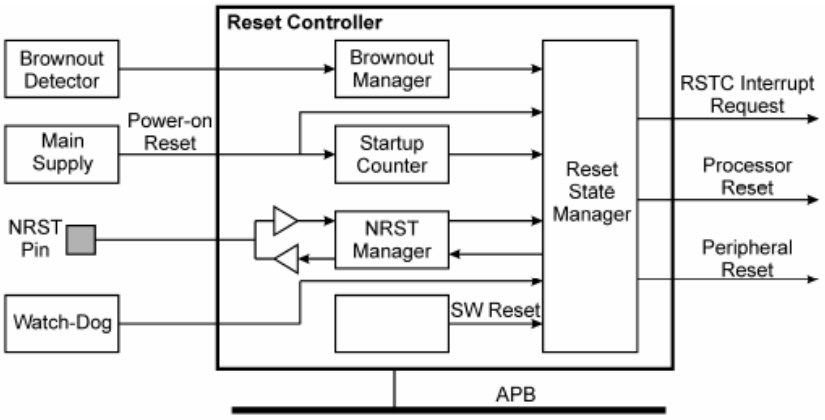
10. Opisati princip spajanja periferija s vanjskim priključcima mikrokontolera.



- periferno sklopovlje spojeno je na sabirnicu APB
- komunicira s okolinom preko priključaka mikrokontrolera

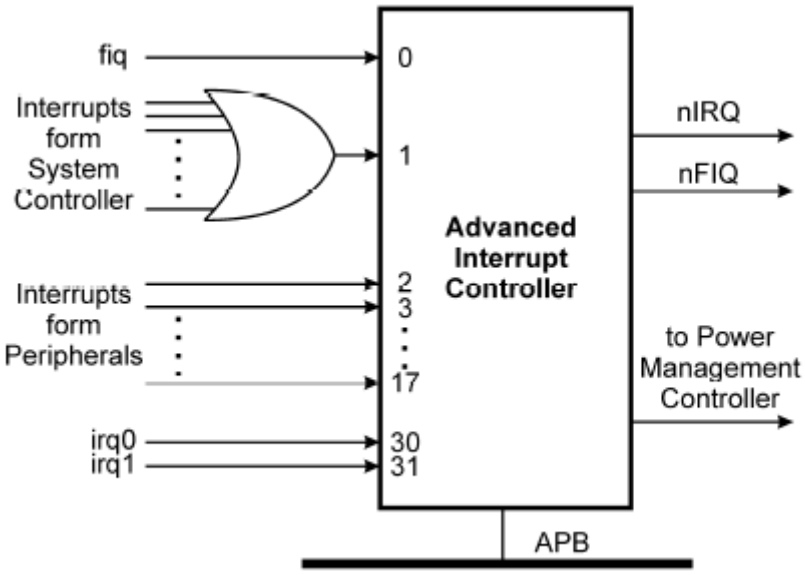
11. Nacrtati pojednostavljenu blokovsku shemu sklopovlja za upravljanjem resetom i navesti funkciju pojedinih dijelova.

- *Reset Controller*, RSTC



- RSTC obrađuje zahtjeve pojedinih izvora reseta
  - reset nakon priključivanja napajanja (*Power-up Reset*)
  - reset nakon propada napajanja (*Brownout Reset*)
  - reset koji je zatražio *Watch-dog* (*Watch-dog Reset*)
  - programski reset (*Software Reset*)
    ⇒postavljanje odgovarajućih bitova u RSTC_CR
  - reset koji traži korisnik
    ⇒priključak na mikrokontroleru (npr. tipka za reset)
- RSTC generira signale za reset koje se vode na
  - procesor
  - periferije
  - priključak mikrokontrolera

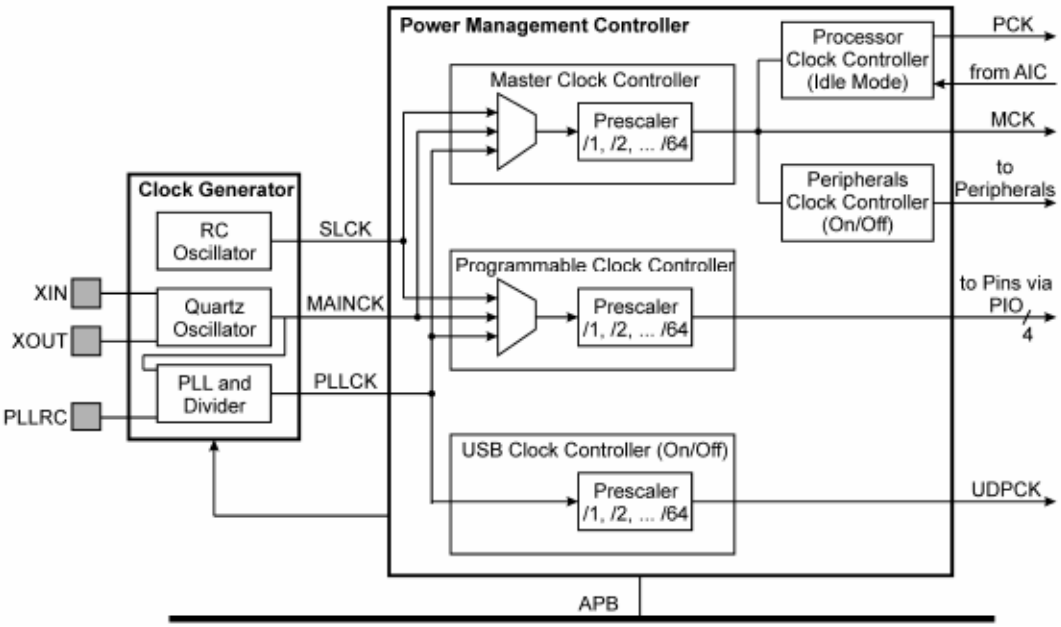
12. Skicirati i opisati pojednostavlenu blokovsku shemu sklopovlja za upravljanje prekidima.



- *Advanced Interrupt Controller*, AIC
- ARM7TDMI ima 2 ulaza za prekid
  - nIRQ
  - nFIQ
- svakom od ovih zahtjeva za prekid pripada jedan prekidni vektor
  - IRQ → 0x0000 0018
  - FIQ → 0x0000 001C
- sklop za upravljanje prekidima (AIC)
  ⇒omogućuje obradu zahtjeva koje daju do 32 jedinice
- kod mikrokontrolera SAM7X128, AIC obrađuje
  - 3 vanjska izvora prekida (vanjski priključci mikrokontrolera)
    ⇒IRQ0 (=30), IRQ1 (=31) i FIQ (=0)
  - 16 izvora prekida periferija
    ⇒POIA (=2), PIOB, SPI0, SPI1, USART0, USART1, SSC, PWMC, UDP, TC0, TC1, TC2, CAN, EMAC, ADC(=17)
  - 1 izvor prekida od upravljačkog sustava (*System Controller*) (=1)
    ⇒PIT, RTT, WDT, DBGU, PMC, RSTC, EFC

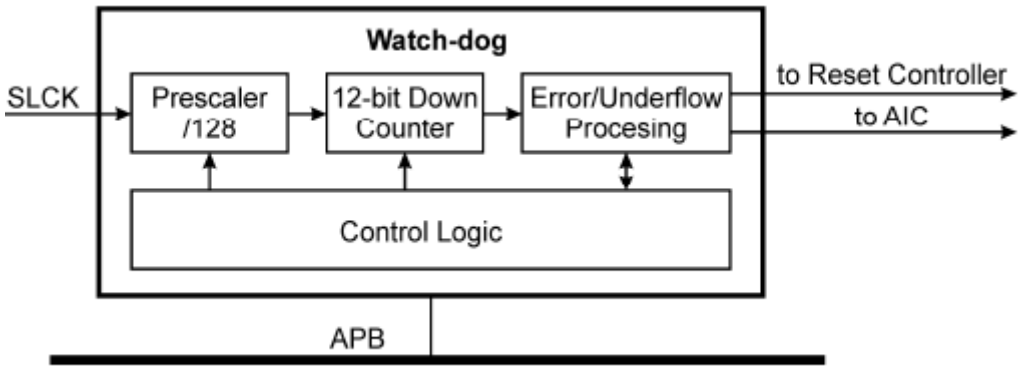
13. Skicirati i opisati pojednostavljenu blokovsku shemu sklopovlja za upravljanje taktom.
(nije pojednostavljena.... lol)

- blokovska shema sustava za sintezu i razvođenje takta

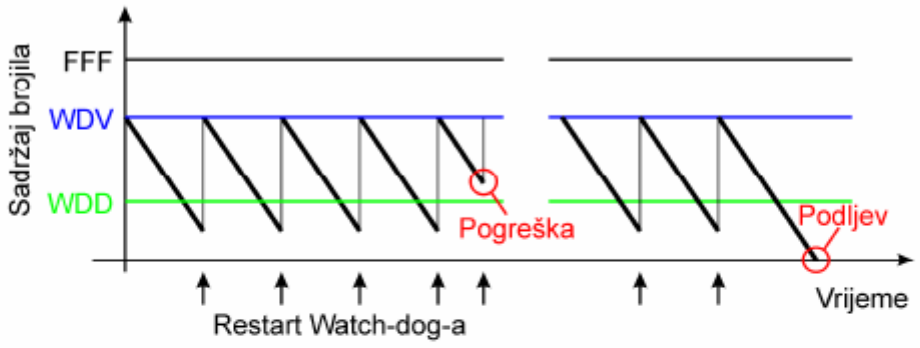


- mikrokontroler ima 3 izvora takta (*Clock Generator*)
  - SLCK - spori takt (*Slow Clock*) ⇒ frekvencija 32kHz±10kHz
  - MAINCK - glavni takt (*Main Clock*)
  - PLLCK - takt dobiven PLL sintezom (*PLL Clock*)
- *Power Management Controller*, PMC
  - upravlja radom izvora takta
  - generira takt za
    - procesor
    - periferije
    - USB
    - izlazno sklopovlje
  - optimira potrošnju energije
    ⇒ uključuje i isključuje takt periferijama i procesoru

14. Skicirati pojednostavljenu blokovsku shemu *Watch-dog* sklopa. Opisati način njegovog rada. Objasniti razliku između podljeva i pogreške.

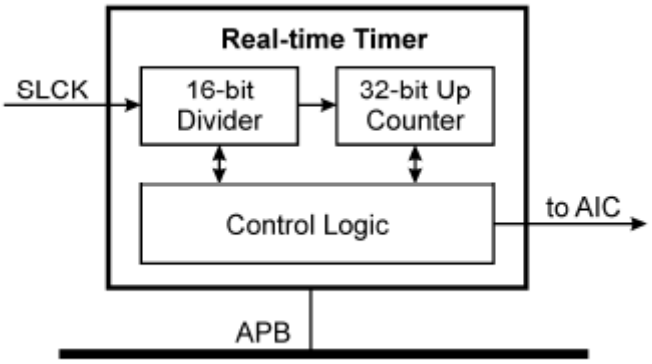


- principa rada



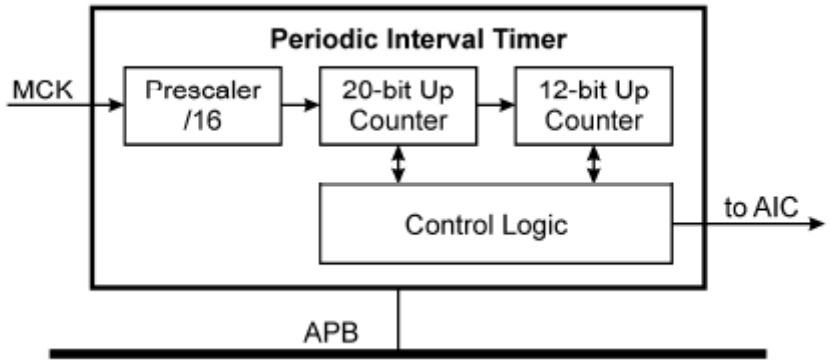
- brojilo broji od vrijednosti WDV (*Watch-dog Value*) prema nuli
- kad brojilo odbroji do 0
  ⇒ podljev (*Underflow*)
- kad se brojilo restarta a sadržaj mu nije manji od vrijednosti WDD (*Watch-dog Delta*)
  ⇒ pogreška (*Error*)
- oba ova slučaja uzrokuju reset mikrokontotolera i/ili prekid ako su oni omogućeni

15. Skicirati pojednostavljenu blokovsku shemu sklopa *Real-time Timer* i opisati način njegovog rada.



- RTT mjeri ukupno proteklo vrijeme
- 16 bitno djelilo (*16-bit Divider*)
  - faktor dijeljenja se može programirati
  - nakon reseta μC, konfigurirano je tako daje impulse perioda 1s
- glavno brojilo (*32-bit Up Counter*)
  - broji do 2^{32} ⇒ ako broji sekunde, broji preko 136 godina
- upravljačko sklopovlje (*Control Logic*)
  - upravlja radom brojila
  - uspoređuje vrijednost glavnog brojila s vrijednošću registru RTT_AR
  - postavlja zastavice u registru RTT_SR i daje zahtjev za prekid
- procesor može čitati izlaz iz glavnog brojila (32-bitnog)
  - problem
    - brojilo broji impulse sinkrone s SLCK
    - čitanje je sinkrono s MCK
    - SCLK i MCK nisu sinkroni
      ⇒može doći do pogrešnog čitanja
  - rješenje
    - vrijednost brojila treba čitati dvaput
      ⇒dva uzastopna čitanja moraju dati istu vrijednost

16. Skicirati pojednostavljenu blokovsku shemu sklopa *Periodic Interval Timer* i opisati način njegovog rada.



- PIT se koristi za generiranje periodičnih zahtjeva za prekid

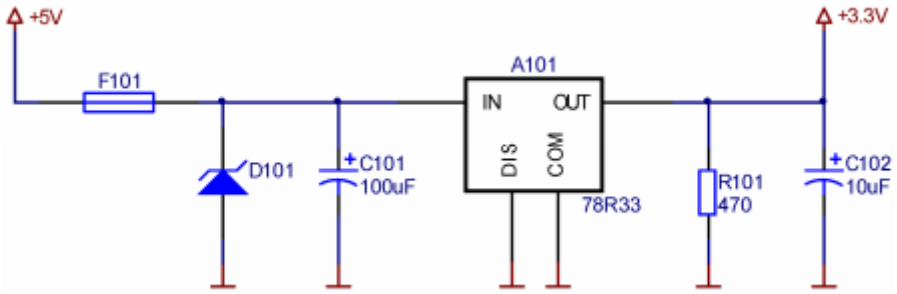
- 20 bitno brojilo (*20-bit Divider*)
    - broji do vrijednosti PIV u registru PIT_MR
    - kad odbroji do PIV
        - ponovo počne brojiti od 0
        - poveća sadržaj 12-bitnog brojila za 1
- upravljačko sklopovlje (*Control Logic*)
    - upravlja radom brojila
    - postavlja zastavice u registru PIT_SR i daje zahtjev za prekid

- uočiti
    - PIT broji impulse iz kristalnog oscilatora (za razliku od RTT)
      ⇒mogu se generirati intervali točnog trajanja

4.8 Pitanja za provjeru znanja

1. Opisati kako je izvedeno napajanje mikrokotrolera familije AT91SAM7X. Nacrtati izvedbu serijskog regulatora koji daje glavno napajanje.

→Sklopovlje µC koristi dva napajanja: 3.3V za ulazno/izlazno sklopovlje i Flash memoriju, te 1.8V za jezgru, sklopovlje za sintezu takta, te ostalo sklopovlje. Unutar µC postoji regulator napona sa 3.3V na 1.8V.



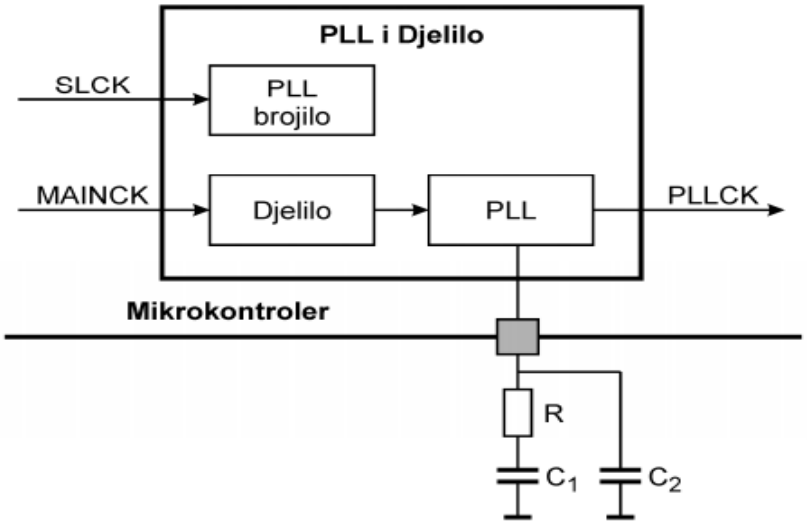
Slika 1.serijski regulator koji daje glavni napon

2. Opisati izvore takta mikrokontrolera AT91SAM7X. Koje vanjsko sklopovlje je potrebno spojiti na izlaze mikrokontrolera za ispravan rad ovih izvora?

→Izvori takta: RC oscilator, Glavni oscilator i PLL.

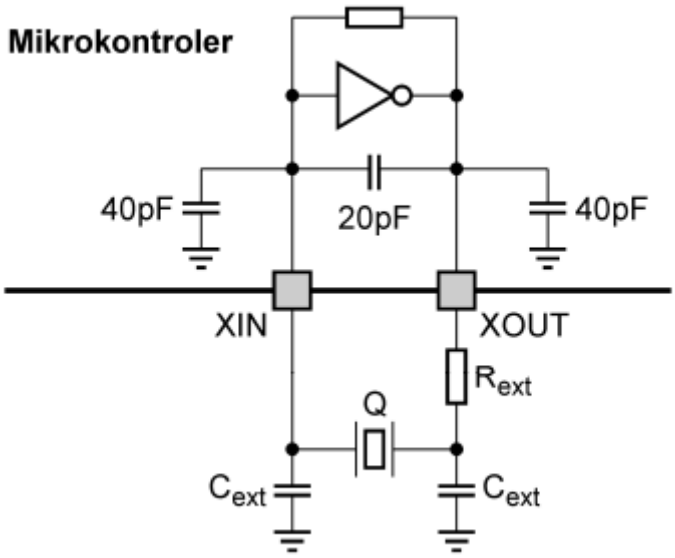
RC oscilator: spori takt, frekvencija 32kHz ± 10kHz, koristi se upočetku dok se ne istitra glavni oscilator

Glavni oscilator: Koristi kristal kvartza za stabilizaciju frekvencije. Koristi paralelnu rezonancu kristala, te mu je frekvencija u području: 3MHz ≤ fQ ≤ 20MHz



Slika 2. PLL izvor takta

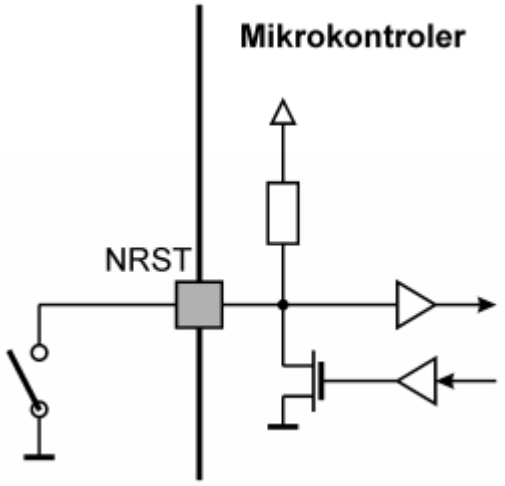
Na ulaz u glavni oscilator potrebno je spojiti kristal kvartza, te po jedan kondenzator na svaki priključak. Time se osigurava povratna veza na fQ. Kod izvedbe µC, korištenog u sklopu predavanja, ne stavlja se otpornik Rext ,koji inače služi za poboljšanje temperaturne karatkeristike oscilatora.



Slika 3. izvedba oscilatora

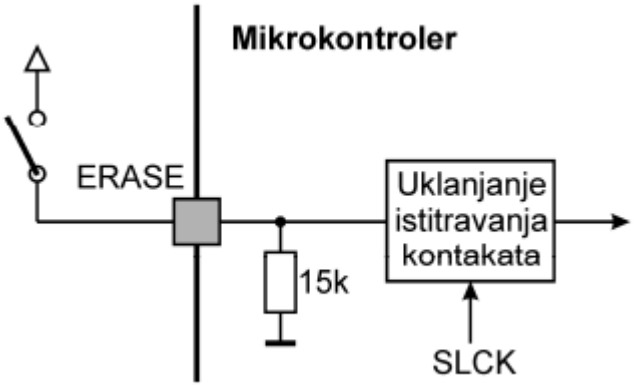
3. Opisati priključke za reset i brisanje memorije i nacrtati spoj vanjskih komponenata koje je potrebno ugraditi. Opisati spoj svjetleće diode na priključak.

→Na priključak za RESET samo je potrebno dodati tipkalo, te se μC sam brine o širini impulsa za reset. Također, postoji funkcija ovog priključka koja služi za generiranje signala za reset prema drugom μC.



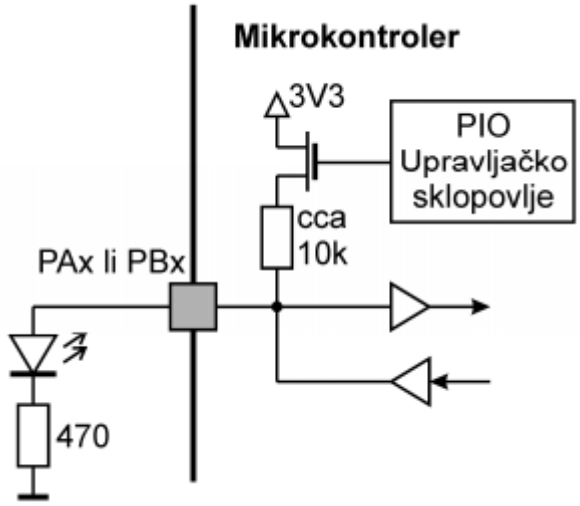
Slika 4. izvedba reset priključka

Dovođenjem pozitivnog impulsa na priključku ERASE briše se sadržaj Flash memorije. Također, postoji sklop za istitravanje (engl. *Debouncing*), te zbog toga tipka mora biti pritisnuta barem 220ms.



Slika 5. izvedba ERASE priključka

Svjetleća dioda kao indikator koji se može upravljati programski. Nakon reset-a pritezni otpornik je uključen, te će zbog toga svjetleća dioda svijetliti slabijim intezitetom.



Slika 6. priključak svjetleće diode

5.3 Pitanja za provjeru znanja

1. **Što su integrirana okruženja za razvoj programske podrške i koje alate oni sadrže.**

- integrirana okruženja za razvoj programske podrške ili kraće IDE je skup alata koji služi za razvoj programske podrške
- npr mi smo koristili Keil uVision IDE (proizvođača Keil Elektronik)
- Keil uVision sadrži:
  - **project manager**
  - **editor (text)**
  - **c/c++ compiler**
  - **macro assembler – prevodilac za asembler**
  - **linker i loader**
  - **debugger i simulator**
  - **hex file generator**
  - **library manager**
  - **c/c++ libraries**

2. **Dati primjer adresiranja registra koji upravljaju radom periferija i upravljačkog sklopovlja i to korištenjem pokazivača i korištenjem pokazivača na strukturu.**

   Dakle, važno:
   svi registri koji upravljaju radom sklopovlja mapirani su u memorijski prostor
   - registrima se pristupa preko pokazivača
   - u uVision okruženju postoje datoteke koje sadrže adrese registara - at91sam7x128.h

1. **Adresiraje preko pokazivača:**

   Primjer

- definicija registara za *Watch-dog Timer* u datoteci at91sam7x128.h

```
typedef volatile unsigned int AT91_REG;

// Registar: WDT_CR (WDT Control Register)
// WatchDogTimerController_WatchDogControlRegister
#define AT91C_WDTC_WDCR ((AT91_REG *) 0xFFFFFD40)

// Registar: WDT_MR (WDT Mode Register)
#define AT91C_WDTC_WDMR ((AT91_REG *) 0xFFFFFD44)

// Registar: WDT_SR (WDT Status Register)
#define AT91C_WDTC_WDSR ((AT91_REG *) 0xFFFFFD48)
```

Primjer

- upis u registrar WDT_CR

31-24	23-1	0
KEY	**-**	**WDRSTT**

```
// Restartanje Watch-dog sklopa
*AT91C_WDTC_WDCR=(0x000000A5<<24)+1;
```

2. **Adresiraje preko pokazivača na strukturu:**

Primjer

- definicija registara za *Watch-dog Timer* u datoteci at91sam7x128.h

```
typedef volatile unsigned int AT91_REG;

typedef struct _AT91S_WDTC {
   AT91_REG   WDTC_WDCR;
   AT91_REG   WDTC_WDMR;
   AT91_REG   WDTC_WDSR;
} AT91S_WDTC, *AT91PS_WDTC;

#define AT91C_BASE_WDTC ((AT91PS_WDTC) 0xFFFFFD40)
```

Primjer

- upis u registrar WDT_CR

31-24	23-1	0
KEY	**-**	**WDRSTT**

```
// Deklaracija
AT91S_WDTC *WDT = AT91C_BASE_WDTC;
// Restartanje Watch-dog sklopa
WDT-> WDTC_WDCR = (0x000000A5<<24)+1;
```

3. **Navesti osnovne tipove registara koji upravljaju radom sklopvlja u mikrokontrolerima familije AT91SAM7X. Po čemu se ovi tipovi razlikuju, obzirom na njihovu funkciju?**

- razlikujemo 3 vrste registara
  - registri čiji sadržaj definira ponašanje sklopovlja
    - *Mode Registers*
      ⇒npr. *Watch Dog Timer Mode Register*, WDT_MR
    - moguće je iz njih čitati ili u njih upisivati
    - nakon reseta imaju predefinirano stanje
  - registri koji služe za unos komande
    - *Control* ili *Command Registers*
      ⇒npr. *Watch Dog Timer Command Register*, WDT_CR
    - u njih je moguće samo upisivati
  - registri koji sadrže status sklopa (zastavice)
    - *Status Registers*
      ⇒npr. *Watch Dog Timer Status Register*, WDT_SR
    - iz njih je moguće samo čitati
- većina registara sadrži skupine bitova (polja) od kojih svaka opisuje jednu funkciju

6.7 Pitanja za provjeru znanja

1. Što obuhvaća elektromehanička provjera sklopa? Što obuhvaća funkcijsko uhodavanje sklopovlja?

→*Elektromehanička provjera* sklopa obuhvaća skup provjera prije i nakon priključivanja napajanja. Dakle, uključuje provjeru mehaničke integracije sklopa, te provjeru električne povezanosti komponenata. Nakon priključenja napajanja potrebno provjeriti temperaturu poluvodičkih komponenata, ispravnost napona napajanja i rad oscilatora, itd.

Funkcijsko uhodavanje sklopovlja obuhvaća uhodavanje razvojnog okruženja, te uhodavanje periferije i vanjskog sklopovlja.

2. Opisati princip učitavanja programske podrške u Flash memoriju mikrokontrolera pomoću namjenskog programatora.

→Prilikom ovakvog načina učitavanja programske podrške u Flash memoriju μC nije zalemljen na štampanu pločicu već je priključen na programator. Komunikacija se obavlja preko FFPI sučelja koje podržava paralelni serijski prijenos podataka. Svojstva ovakvog načina programiranja su da je brzo, pogodno za serijsku proizvodnju, ali je nije praktični u fazi razvoja porgramske podrške.

3. Opisati princip učitavanja programske podrške u Flash memoriju mikrokontrolera pomoću SAM-BA programske podrške.

→Ideja za ovaj način učitavanja programske podrške je da se koristi sam μC. Kako bi se ostvarilo učitavanja pomoću SAM-BA programa, μC mora

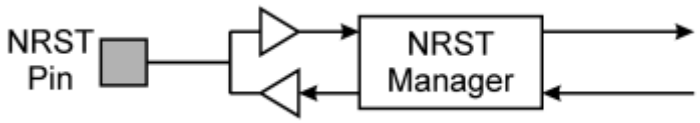
sadržavati program koji inicijalizira sučelja za komunikaciju s PC-om i uspostavlja vezu s PC-om, učitava sadržaj Flash memorije iz osobnog računala, upisuje učitane podatke u Flash memoriju, te postavlja zastavicu GPNVM (mapiranje Flash memorije u prvi MB). Navedeni program nalazi se u ROM memoriji μC. SAM-BA program se počinje izvršavati nakon pritiska tipke ERASE i RESET.

6. Opisati inicijalizaciju osnovnu inicijalizaciju memorijskog kontrolera. Kad je potrebno podesiti broj stanja čekanja a kad to nije potrebno?

→Za inicijalizaciju memorijskog kontrolera potrebno je postaviti odgovarajuće parametre u MC_FMR (*Memory Controller – Flash Mode Register*) registar. Naime, postavlja se parametar FWS (*Flash Wait State*) na osnovu maksimalne frekvencije takta, te se u parametar FMCN (*Flash Microsecond Cycle Number*) postavlja broj perioda takta u jednoj mikrosekundi. Broj stanja čekanja je potrebno postaviti ako je frekvencija takta procesora veća od frekvencije rada Flash memorije (max. 30MHz).

7. Opisati princip rada sklopovlja koje pogoni priključak za vanjski reset. Koje osnovne inicijalizacije je potrebno napraviti ako mikrokontroler ima vanjsku tipku za reset?

→U sklopovlju za upravljanjem resetom (NRSTC) o vanjskom priključku brine sklop NRST
Manager.



NRST Manager upravlja propuštanjem reseta s priključka NRST u μC, te generiranjem reseta koji izlazi na priključak NRST. Ako μC ima vanjsku tipku za reset potrebno je unutar RSTC_MR (*Reset Controller – Mode Register*) registra postaviti bit URSTEN (*User Reset Enable*) u jedinicu.

8. Opisati osnovu inicijalizaciju Watch-dog sklopa u fazi razvoja sklopovlja. Zašto je ona potrebna? Kako treba inicijalizirati Watch-dog sklop u stvarnom radu?

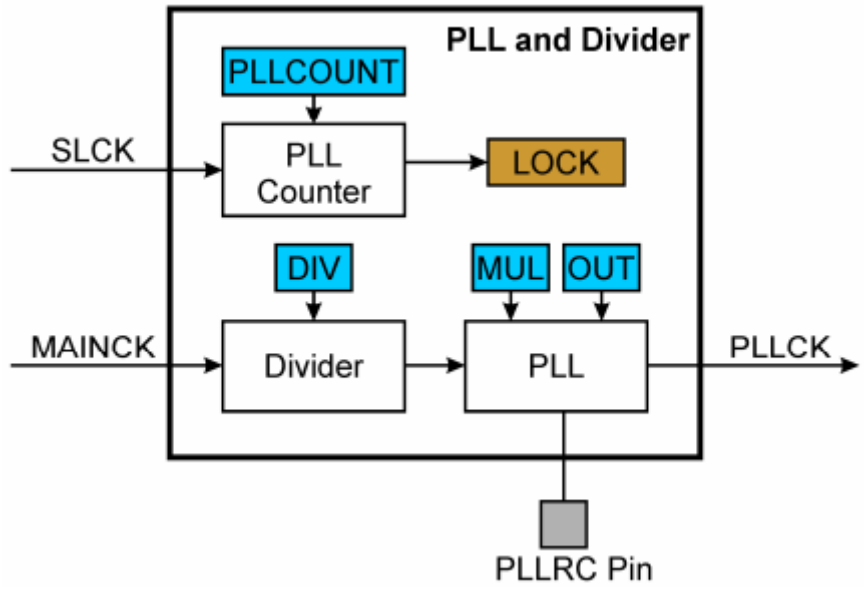
→Kod uhodavanja sklopova rad Watch-dog sklopa je nepoželjan, pa ga je potrebno onemogućiti. Njegovo onemogućavanje se ostvaruje postavljanjem bita WDDIS u jedinicu unutar WDT_MR (*Watch Dog Timer – Mode Register*) registra. U stvarnom radu Watch-dog sklop se omogućuje (postavljanjem bita WDDIS u nulu) kako bi se detektirao pogrešan rad. Također, prilikom inicijalizacije u stvarnom radu potrebno je definirati dvije vrijednosti, WDV i WDD, koje definiraju podljev i pogrešku, retrospektivno.

9. Opisati inicijalizaciju sklopovlja glavnog oscilatora.

→Inicijalizacija glavnog oscilatora vrši se postavljanjem bita za omogućavanje (MOSCEN) i polja vremena utitravanja (OSCOUNT) u registru za upravljanjem radom glavnog oscilatora (CKGR_MOR). Ako se μC ne koristi za low-power aplikaciju, preporučuje se uzeti maksimalnu vrijednost (0xFF) vremena utitravanja.

10. Opisati inicijalizaciju PLL sklopovlja.

→



Potrebno odabrati frekvencijsko područje u kojem će raditi PLL postavljanjem bitova u polju OUT (80 – 160MHz -> OUT = 00b ; 150 – 200MHz -> OUT = 10b). OUT se postavlja u ovistnosti o izlaznoj frekvencije PLL-a. Izlazna frekvencija se određuje
prema formuli:

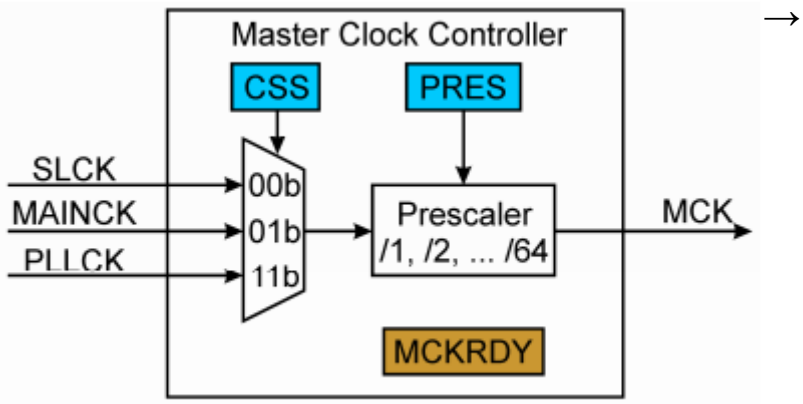
$$f_{PLLCK} = f_{MAINCK} \frac{MUL+1}{DIV}$$

Prije dolaska takta na procesor, PLLCK se dovodi na djelilo, te vrijedi:

$$f_{MCK} = \frac{f_{PLLCK}}{2^{PRES}} = f_{MAINCK} \frac{MUL+1}{DIV \cdot 2^{PRES}}$$

Iz ovih formula se mogu naći parametri MUL, DIV i PRES koje potrebno prilikom inicijalizacije postaviti u *Wizardu.*

11. Opisati sklopovlje za odabir takta na kojem radi procesor. Kako se izvodi njegova inicijalizacija?. Na što je potrebno posebno paziti kod inicijalizacije ovog sklopovlja?



Sklopovlje za odabir takta služi prilikom priključenja napajanja (paljenje μC). Naime, na početku nije moguće koristiti glavni izvor takta jer mu je potrebno neko vrijeme dok se ne ustali (utitravanje/stacionarno stanje). Stoga se upočetku koristi spori takt sa RC oscilatora dok se čeka utitravanje glavnog oscilatora. Kako bi se mogao odabrati izvor takta potrebno u CSS polju unutar PMC_MCKR (*Master Clock Register*) registra postaviti željeni izvor. Pomoću polja PRES koje se također nalazi u upravljačkom registru (PMC_MCKR), određuje se vrijednost djelila. Prilikom inicijalizacije ovog sklopovlja treba paziti na čekanje utitravanja nakon postavljanja CSS.

7 Programska podrška
(nema pitanja za vježbu pa je napisan sažetak)

- razvojno okruženje sadrži
  - ciljni računalni sustav (*target system*)
    - ovaj sustav razvijamo
  - računalo domaćin (*host*)
    - sadrži
      - prevodioc (*cross-compiler*)
      - alate za povezivanje i punjenje (*linker*, *loader*)
      - okolinu za uhodavanje (*debugger*)
      - itd.
- uhodavanje se izvodi
  - isključivo na domaćinu
    ⇒ koristi se simulator
  - isključivo na ciljnom sustavu
    ⇒ program se učita u ciljni sustav i izvodi na sklopovlju
    ⇒ ispitivanje ispravnosti izvodi se preko nekog vanjskog sučelja
      - npr. konzola s kojom se komunicira preko serijskog sučelja
  - djelomično na ciljnom sustavu a djelomično na domaćinu (*semihosting*)
    ⇒ vidi daljni tekst

- **library:**
  - **runtime** - neke operacije i funkcije koje sadrži programski jezik ne mogu se mapirati izravno
    ⇒ prevodilac ih implementira koristeći funkcije iz biblioteke koja se naziva runtime library
  - **standard** – standardna biblioteka C jezika, mat funkcije, operacije sa stringovima, castanja isl...sadrži zaglavlja i rutine
  - **custom** - biblioteke koje sadrže specifične funkcije s aspekta aplikacije ili računalnog sustava

- funkcije koje koriste sklopovlje u ARM bibliotekama implementirane su tako da da podržavaju semihosting
  ⇒ Funkcije čiji rad ovisi o sklopovlju tokom svog izvršavanja daju zahtjev za SWI.
  ⇒ SWI poslužuje agent (*debug agent*)

- uočiti
  - ovo je korisno kod uhodavanja programske podrške
  - posebno je korisno dok sklopovlje još ne postoji (npr. u izradi je)
  - u konačno aplikaciji `fwrite` mora komunicirati sa stvarnim sklopovljem a ne s agentom
    ⇒treba "reimplementirati" funkciju `fputc` tako da radi sa stvarnim sklopovljem
    ⇒to se zove redefinicija (*retargeting*)

- sustav za povezivanje podržava (za ARM prosecore)
  - "*bare metal*" model
    - tipičan za razvoj ugradbenih računalnih sustava
    - izlaz je cjelovit program koji radi izravno sa sklopovljem
    - program može sadržavati i operacijski sustav
  - djelomično povezivanje (*partial linking*)
    - izlaz kod ovakvog povezivanja predstavlja ulaz kod narednog povezivanja

Bare machine –
- program koji ne sadržava main funkciju
- ne inicijalizira se okruženje za korištenje biblioteka

- uočiti
  - biblioteku je potrebno oblikovati da bi odgovarala aplikaciji
    ⇒*library tayloring*
  - oblikovanje je obavezno u slučaju kad funkcije moraju raditi na sklopovlju

7.5 Pisanje programa koji se izvršava na sklopovlju

7.5.1 Oblikovanje funkcija za rad s vanjskim sklopovljem

- ako funkcija radi na stvarnom sklopovlju, potrebno je
  - oblikovati funkcije koje koriste I/O (*tayloring*)
    ⇒ zamjena funkcija iz biblioteke novima (*retargeting*)
  - u kod dodati simbol `__use_no_semihosting_swi`
    - potrebno je dodati samo u JEDNOM cijelom izvornom kodu

    ⇒ izvedba u asembleru
    ```
    IMPORT __use_no_semihosting_swi
    ```
    ⇒ izvedba u jeziku C
    ```
    #pragma import(__use_no_semihosting_swi)
    ```

#pragma import(__use_no_semihosting_swi) // !! UOCITI !!

Primjer

Napisati program koji ispisuje poruku *Pozdrav svijete!*. Program se mora izvršavati na sklopovlju mikrokontrolera AT91SAM7X. Kao izlaznu jedinicu koristiti serijsko sučelje USART0. Za ispis koristiti funkciju `printf()`. Potrebno je koristiti redefiniranje ulazno-izlaznih funckija.

Rješenje

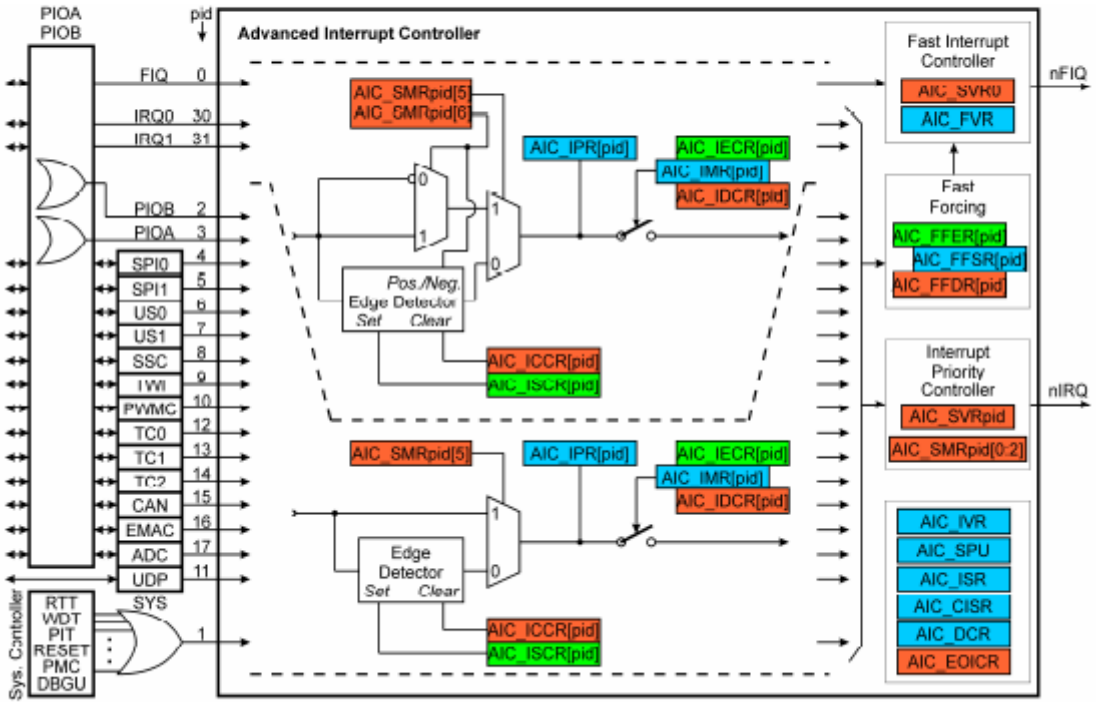
- cjelokupni program sadrži
  - funkciju `main()`
  - redefiniciju funkcija koje koriste *semihost* okruženje
  - funkciju za inicijalizaciju i rad sa serijskim sučeljem

- USART (*Universal Synchronous Asynchronous Receiver Transmitter*)
  - podržava sinkronu i asinkronu komunikaciju
  - podržava serijski prijenos s 5 do 9 bitova u nizu
  - podržava razne protokole kao na primjer
    - RS232
    - RS485
    - IrDA (*Infrared Data Association* protokol)
    - ISO7816 (komunikacija sa *Smart Card* karticama)

- USART sadrži
  - generator takta za prijenos (*baud rate generator*)
  - sklopovlje za slanje i primanje podataka

7.6 Zahtjevi za prekid koje daje vanjsko sklopovlje

- pojednostavljena blokovska shema prekidnog sustava



- posluživanje prekida
  - pojava zahtjev za prekid postavlja bit AIC_IPR[pid]
  - ako je prekid omogućen (AIC_IMR[pid]=1), ide se u daljnju obradu
  - sadržaj AIC_SVRpid prebacuje se u AIC_IVR
  - registar AIC_ISR sadrži pid tekućeg prekida
  - prekid se poslužuje na slijedeći način
    - prekidna funkcija čita AIC_IVR
    - prekidna funkcija obavlja svoj posao
    - prekidna funkcija upisuje bilo što u AIC_EOICR

- neregularni prekidi (*Spurious Interrupt*)
  - neregularni prekid
    - npr. prekid na razinu koji se deaktivirao prije nego je poslužen
  - obrada neregularnog prekida
    - ista kao i za regularan prekid osim što je se prekidni vektor nalazi u AIC_SPU a ne u AIC_IVR
  - uočiti
    - neregularan prekid nema masku

      ⇒potrebno je uvijek napraviti funkciju za obradu neregularnog prekida

- obrada prekida koji zatraži System Controller (SYS)
  - uočiti
    - 6 izvora prekida usmjereno je na jedan zahtjev (RTT, WDT,..)

      ⇒u AIC postoji samo jedan prekidni vektor (AIC_SVR1)

      ⇒ista prekidna funkcija (*Interrupt Handler*) poslužuje sve ove prekide

  ⇒u prekidnoj funkciji potrebno je najprije utvrditi koji je od ovih izvora zatražio prekid

  ⇒potrebno je redom čitati statusne registre u RTT, WDT, ... i vidjeti koji sklop je zatražio prekid

- na sličan način obrađuju se i prekidi PIOA i PIOB

  ⇒u prekidnoj funkciji treba otkriti od kojeg U/I priključka je došao zahtjev za prekid

- uočiti
  - prekid IRQ omogućen je samo u načinu rada User

7.7.3 Inicijalizacija prekida na razini AIC

- programski odsječak

```
#include <at91sam7x128.h>

void irq0_handler (void) __irq;
void spurious_handler (void) __irq;

unsigned int brojac_prekida=0;

int main(void) {

   // ********* Tip i prioriteta prekida IRQ0 **********

   *(AT91C_AIC_SMR+AT91C_ID_IRQ0) = (0x0<<0) | (0x0<<5);
        // *(0xFFFFF000+30) => Adresa registra AIC_SMR30
        // AIC_SMR30 => PRIOR=0, SRCTYPE=0 rastuci brid

   // *************** Prekidni vektori *****************

   *(AT91C_AIC_SVR+AT91C_ID_IRQ0)=
                           (unsigned int)irq0_handler;
        // AIC_SVR30=Adresa_prekidne_funkcije_za_IRQ0
   *AT91C_AIC_SPU = (unsigned int) spurious_handler;
        // AIC_SVR30=Adr._funkcije_za_neregularni_pr.

   // ************** Omogucavanje prekida **************

   *AT91C_AIC_IECR = 1<<AT91C_ID_IRQ0;
        // AIC_IECR => postavljanje bita 30 u 1

   // ***************** Programska petlja ***************

    while(1) {
       // Ovdje dolazi neki koristan kod
    }
}
```

7.7.4 Prekidne funkcije

- prekidne funkcije
  - moraju sačuvati stanja registara koje koriste
    ⇒koristi se atribut `__irq`

- programski odsječak

```
void irq0_handler (void) __irq {
   brojac_prekida++;
   *AT91C_AIC_EOICR = 0;            // UOCITI
}
void spurious_handler (void) __irq {
      *AT91C_AIC_EOICR = 0;
}
```

7.9 Programski prekid

- programski prekid posljedica je izvođenja instrukcije
  - ARM skup instrukcija

    SWI{<cond>} <immed_24>

  - Thumb skup instrukcija

    *SWI <immed_*

- povratak iz *Supervisor* načina rada
  ⇒instrukcija koja učitava registra R15 (PC) iz R14_svc
  - primjer

```
MOVS PC, R14_svc ; S oznacava povrat CPSR
```

⇒ova instrukcija radi slijedeće

```
CPSR = SPSR_svc ; vraća prvobitni sadržaj CPSR
                ; to oznacava slovo S u instrukciji
PC   = R14_svc  ; vraća kontrolu aplikaciji
```

- **razlika u posluživanju sklopovskih i programskih prekida**
  - sklopovski prekidi
    - izaziva ih signal
    - točka u kojoj je prekinuto izvođenje programa je nepoznata
      ⇒odabir posluživanja izvodi se pomoću prekidnog vekora
      ⇒argumenti se mogu prenositi jedino preko statičkih (globalnih) varijabli
      - uočiti: registri (npr. u AIC) su globalne varijable (samim tim su i statičke)
  - programski prekidi
    - prekid je izazvan istrukcijom
    - točka u kojoj je prekinuto izvođenje programa je točno određena i poznata
      ⇒odabir posluživanja sadržan je u samoj instrukciji koja je izazvala prekid
      ⇒argumenti se mogu prenositi preko statičkih varijabli, ali i preko registara

- u nekim arhitekturama mnemonik SWI (*SoftWare Interrupt*) je zamijenjen sa SVC (*SuperVisory Call*)

7.11 Prijenos argumenata u funkciju i iz funkcije

- način prijenosa argumenata u funkciju i iz funkcije
  ⇒*calling convention*
- općenito, argumenti se u funkciju mogu prenositi
  - preko stoga
  - preko registara procesora
  - kombinacijom prijenosa preko registara i preko stoga
- ARM koristi kombinaciju prijenosa
  - prijenos argumenata u funkciju
    - preko registara R0, R1, R2, R3 i preko stoga
  - prijenos rezultata iz funkcije
    - preko registara R0, R1

- uočiti:
  - prijenos argumenata u funkciju
    - `a`, `b`, `c` i `d` se prenose preko registara R0, R1, R2 i R3
    - `e` se prenosi preko stoga
  - povrat rezultata iz funkcije
    ⇒u registru R0
  - registri od R4 do R12 koriste se kao "radni registri"
    ⇒ako ih funkcija koristi, njihov sadržaj sprema se na stog
    ⇒vidi instrukcije STMDB i LDMIA
  - poziv funkcije
    - instrukcija BL (*branch and link*)
      ⇒adresa slijedeće istrukcije ide u R14 (LR)
      ⇒početan adresa potprograma stavlja se u R15 (PC)
  - povrat iz funkcije je instrukcija BX
    ⇒ona po potrebi mijenja skup instrukcija (ARM/Thumb)
  - jednostavne funkcije u pravilu se ne pozivaju
    ⇒linker njihov kod ubacuje u kod funkcija koja ih poziva (slično *inline* funkcijama)
    - gornji primjer napravljen je uz opciju opcija linkera
      `--inline`

- kako se prenose argumenti veći od jedne riječi
  - primjer
    - `int` se vraća preko R0
    - `long` se vraća preko R0 jer sadrži 32 bita
    - `long long` se vraća preko R0 (niža 32 bita) i R1 (viša 32 bita)

7.12 Semihosting

7.12.1 Semihosting

- podsjetimo se
  - ulazno-izlazne funkcije (npr. `printf`) rade sa sklopovljem
  - ove funkcije u biblioteci NE sadrže komunikaciju sa sklopovljem
  - ove funkcije u biblioteci sadrže zahtjev za programski prekid

- ideja
  - tokom uhodavanja koristiti agenta a ne sklopovlje

- agent
  - program ili programski odsječak koji radi na principu zahtjeva za nekom uslugom (kao što to rade agencije, npr. turističke)
    ⇒u našem slučaju
    - posluživanje zahtjeva za programski prekid
    - komunikacija s računalom domaćinom
  - ima određen stupanj autonomije u izvršavanju
    ⇒u našem slučaju
    - simulira sklopovlje neovisno o programu ciljnog sustava
    - obavlja radnje koje može zadati računalo domaćin

- **prijelazak s agenta na sklopovlje izvodi se**
  - reimplementacijom funkcija koje traže semihostong
    ⇒*retarget*
    ⇒ovo smo obradili u poglavlju 7.5
  - pisanjem vlastitog SWI *handlera* koji obrađuje semihost pozive
    ⇒vidi daljnji tekst

7.12.2 Semihosting operacije

- funkcije koje traže posluživanje *semihost* agenta daju zahtjev za SWI
- parametar instrukcije SWI koja traži posluživanje agenta je dogovoren
  ⇒SWI ima dva moguća oblika

```
SWI #0x123456 ; ARM skup instrukcija
SWI #0xAB     ; Thumb skup instrukcija
```

  - ovi parametre ćemo u kodu zvati `SWI_number`

- operacija koja se od agenta traži i njeni parametri, sadržani su u argumentima (koji se nalaze u registrima, poglavlje 7.11)
  - ulazni parametri
    - registar R0

Argument u R0	Namjena
0x00-0x31	Koristi ga ARM (biblioteke)
0x32-0xFF	Rezervirano za buduću upotrebu za ARM
0x100-0x1FF	Rezervirano za korisnika (vidi napomenu)
0x200-0xFFFFFFFF	Neiskorišteno (preporuka da se i ne koristi)

      - napomena: ukoliko se želi koristiti SWI za vlastite funkcije, preporučuje se koristiti drugi `SWI_number`, a ne semihost `SWI_number`
    - registar R1
      - sadrži pokazivač na sve ostale parametre

//NEVEZANO ALI VAŽNO

- programski odsječak `__main` sadrži
  - poziv potprograma za učitavanja (*load*)
  - poziv ljuske koja radi s bibliotekama

7.12.3.3Posluživanje prekida

- posluživanje prekida sadrži
  - određivanje parametra `SWI_number`
    ⇒iz operacijskog koda instrukcije SWI
  - poziv funkcije za obradu
    ⇒ovisno o SWI_number

- C funkcija za posluživanje prekida

```
void C_SWI_Handler (int SWI_number, int *arguments) {
  switch(SWI_number) {
    case(0x123456): PURS_agent(arguments); break;
    case(0xAB):     PURS_agent(arguments); break;
    // case(....) ... Ovdje idu i eventualni drugi SWI pozivi
    default: send_string("\n\rNepoznat SWI_number.\n\r"); break;
  }
}
```

- **neimplementirane funkcije**

- implementirati treba sve funkcije
  - one koje se ne koriste, implementirati kao "dummy"
  - predvidjeti ispis poruka "dummy" funkcija
    ⇒to olakšava uhodavanje programske podrške