

FAKULTET ELEKTROTEHNIKE
I RAČUNARSTVA

Zavod za elektroničke sustave
i obradbu informacija

**PROJEKTIRANJE UGRADBENIH RAČUNALNIH
SUSTAVA**
Upotreba 32-bitnih mikrokontrolera

(Materijali za predavanja)

Mladen Vučić

Zagreb 2009

Verzija: 2009-1.3
Datum: 18.5.2012
Datoteka: arm_pred_v136.doc

Sadržaj

| | |
|--|-----------|
| 1 Uvod..... | 1 |
| 2 Procesori i kontroleri..... | 2 |
| 2.1 Pregled pojmova | 2 |
| 2.2 Procesori s ARM arhitekturom | 5 |
| 2.3 Pitanja za provjeru znanja..... | 8 |
| 3 Mikrokontroleri familije AT91SAM7X | 9 |
| 3.1 Osnovna blokovska shema..... | 9 |
| 3.2 Procesorska jezgra ARM7TDMI | 10 |
| 3.2.1 Arhitektura..... | 10 |
| 3.2.2 Registri..... | 12 |
| 3.2.3 Instrukcije | 15 |
| 3.2.4 Obrada iznimaka | 24 |
| 3.2.5 Reset..... | 25 |
| 3.3 Memorije | 26 |
| 3.3.1 Izvedbe memorija..... | 26 |
| 3.3.2 Sklopolje za upravljanje memorijama | 26 |
| 3.3.3 Memorijska mapa..... | 29 |
| 3.4 Periferno sklopolje..... | 34 |
| 3.5 Upravljačko sklopolje sustava | 35 |
| 3.5.1 Sklopolje za upravljanje resetom..... | 35 |
| 3.5.2 Sklopolje za upravljanje prekidima | 36 |
| 3.5.3 Sinteza i upravljanje taktom | 38 |
| 3.5.4 Kontroler ulazno-izlaznog sklopolja | 39 |
| 3.5.5 <i>Watch-dog</i> | 40 |
| 3.5.6 <i>Real-time Timer</i> | 42 |
| 3.5.7 <i>Periodic Interval Timer</i> | 43 |
| 3.5.8 Sklopolje za uhodavanje mikrokontrolera..... | 44 |
| 3.6 Pitanja za provjeru znanja..... | 45 |
| 3.7 Zadaci za samostalni rad | 45 |
| 4 Električko spajanje mikrokontrolera | 46 |
| 4.1 Vanjski priključci mikrokontrolera | 46 |
| 4.2 Napajanje mikrokontrolera | 46 |
| 4.3 Izvor takta | 48 |
| 4.3.1 RC oscilator | 48 |
| 4.3.2 Glavni oscilator..... | 48 |
| 4.3.3 PLL..... | 50 |
| 4.4 Reset | 51 |
| 4.5 ERASE priključak..... | 51 |
| 4.6 Indikator | 52 |
| 4.7 Primjer sheme spajanja | 53 |
| 4.8 Pitanja za provjeru znanja..... | 54 |
| 4.9 Zadaci za samostalni rad | 54 |

| | |
|---|-----------|
| 5 Alati za razvoj programske podrške | 55 |
| 5.1 Razvojno okruženje | 55 |
| 5.2 Pristup registrima koji upravljaju radom sklopolja | 56 |
| 5.2.1 Pristup registrima | 56 |
| 5.2.2 Adresiranje preko pokazivača | 57 |
| 5.2.3 Adresiranje preko pokazivača na strukturu | 58 |
| 5.2.4 Tipovi registara..... | 59 |
| 5.3 Pitanja za provjeru znanja..... | 60 |
| 5.4 Zadaci za samostalni rad | 60 |
| 6 Puštanje u pogon..... | 61 |
| 6.1 Elektromehanička provjera sklopa | 61 |
| 6.1.1 Provjere prije priključivanja napajanja | 61 |
| 6.1.2 Provjere nakon priključivanja napajanja | 61 |
| 6.2 Funkcijsko uhodavanja sklopolja..... | 62 |
| 6.3 Razvoj programske podrške za uhodavanje sklopolja | 62 |
| 6.4 Učitavanje programske podrške | 63 |
| 6.4.1 Učitavanje pomoću programatora | 63 |
| 6.4.2 Učitavanje pomoću SAM-BA programa..... | 64 |
| 6.4.3 Učitavanje korištenjem sklopolja za uhodavanje | 66 |
| 6.5 Osnovne inicijalizacije..... | 67 |
| 6.5.1 Inicijalizacija prekidnih vektora | 68 |
| 6.5.2 Inicijalizacija memorijskog kontrolera | 70 |
| 6.5.3 Inicijalizacija sklopolja za reset..... | 72 |
| 6.5.4 Inicijalizacija <i>Watch-dog</i> sklopa..... | 73 |
| 6.5.5 Inicijalizacija izvora takta | 74 |
| 6.5.6 Inicijalizacija stoga | 80 |
| 6.5.7 Startup datoteka..... | 82 |
| 6.6 Prvi program | 83 |
| 6.7 Pitanja za provjeru znanja..... | 88 |
| 6.8 Zadaci za samostalni rad | 88 |
| 7 Programska podrška | 89 |
| 7.1 Okolina za razvoj programske podrške..... | 89 |
| 7.2 Biblioteke | 90 |
| 7.2.1 Osnovni pojmovi..... | 90 |
| 7.2.2 Funkcije koje koriste vanjsko sklopolje..... | 91 |
| 7.2.3 Korištenje biblioteka | 93 |
| 7.3 <i>Bare machine</i> | 95 |
| 7.4 Okolina za rad s bibliotekama | 97 |
| 7.5 Pisanje programa koji se izvršava na sklopolju | 104 |
| 7.5.1 Oblikovanje funkcija za rad s vanjskim sklopoljem | 104 |
| 7.5.2 Program "Pozdrav svijete!" | 105 |
| 7.6 Zahtjevi za prekid koje daje vanjsko sklopolje | 113 |
| 7.7 Primjer prekidnog programa | 116 |
| 7.7.1 Čitanje prekidnog vektora | 117 |
| 7.7.2 Omogućavanje prekida na razini procesorske jezgre..... | 118 |
| 7.7.3 Inicijalizacija prekida na razini AIC | 119 |
| 7.7.4 Prekidne funkcije | 120 |
| 7.8 Unutrašnji izvori prekida..... | 121 |
| 7.9 Programski prekid | 122 |
| 7.10 Tipovi podataka | 125 |
| 7.11 Prijenos argumenata u funkciju i iz funkcije | 127 |

| | |
|---|------------|
| 7.12 Semihosting | 130 |
| 7.12.1 <i>Semihosting</i> | 130 |
| 7.12.2 Semihosting operacije | 132 |
| 7.12.3 Primjer posluživanja SWI | 133 |
| 7.13 Zadaci za samostalni rad | 146 |
| 8 Uhodavanje i ispitivanje sustava..... | 147 |
| 8.1 Osnovni pojmovi | 147 |
| 8.2 Razvoj alata za uhodavanje i ispitivanje | 148 |
| 8.3 <i>Boundary scan</i> arhitektura | 150 |
| 8.3.1 Princip rada | 150 |
| 8.3.2 Izvedba sklopolija | 153 |
| 8.3.3 Upravljanje sklopoljem | 155 |
| 8.3.4 Rad sa sklopoljem | 157 |
| 8.4 Ispitivanja i uhodavanje mikrokontrolera familije AT91SAM7X | 158 |
| 8.4.1 <i>Boundary-scan</i> mikrokontrolera | 159 |
| 8.4.2 Uhodavanje sustava pomoću sklopolja za uhodavanje | 160 |
| 8.4.3 Sklopolje za uhodavanje procesorske jezgre | 161 |
| 8.5 Pitanja za provjeru znanja..... | 165 |
| 9 Periferno sklopolje u mikrokontrolerima..... | 166 |
| 9.1 Sklopolje za komunikaciju | 167 |
| 9.1.1 <i>Serial Peripheral Interface</i> | 167 |
| 9.1.2 <i>Two-wire Interface</i> | 171 |
| 9.1.3 <i>Universal Serial Bus</i> | 173 |
| 9.2 Pitanja za provjeru znanja..... | 181 |
| 10 Završne napomene | 182 |
| 11 Literatura | 183 |

1 Uvod

- svrha gradiva
 - upoznati se s 32 bitnim mikrokontrolerima
 - dati principe dizajna sklopolja i programske podrške
 - ovladati alatima za razvoj računalnih sustava s 32 bitnim µC
- gradivo obuhvaća
 - uvodne napomene
 - izvedbu procesora i kontrolera
 - arhitekturu tipičnog 32 bitnog mikrokontrolera
 - razvoj sklopolja
 - razvoj programske podrške
 - alate za testiranje i uhodavanje
 - periferno sklopolje
 - razne primjere
 - itd.
- primjeri će biti dani za
 - familiju mikrokontrolera AT91SAM7X
 - skupa alata za razvoj programske podrške Keil uVision

2 Procesori i kontroleri

2.1 Pregled pojma

- procesori (P), mikroprocesori (μ P), kontroleri (C), mikrokontroleri (μ C),
⇒ vidi predmet *Ugradbeni računalni sustavi*, 2. ciklus
- razlika
 - μ C i C
⇒ pored CPU sadrži i ROM, RAM i periferije
 - μ P i P
⇒ mogu sadržavati cache, a RAM i ROM su najčešće vanjski
- "n bitni procesor" => povezuje se sa širinom podatkovne sabirnice
 - izuzetak: procesori koji su npr. 32 bitni iznutra, a 16 bitni prema van
⇒ multipleksiranje sabirnice
- Što znači μ ? ⇒ postoje različite definicije
 - definicija koja proizlazi iz arhitekture
 - temelji se na širini podatkovne sabirnice
⇒ 8 i 16 bitne komponente zovu se μ C i μ P
⇒ 32 i 64 bitne (i veće) komponente zovu se C i P
 - definicija koja proizlazi iz implementacije
 - temelji se na izvedbi
⇒ komponente koje su izvedene u jednom integriranom sklopu zovu se μ C i μ P
⇒ složene komponente za čiju implementaciju je potrebno više integriranih sklopova nazivaju se C i P
- nazivi koji se sreću u literaturi
 - nazivi procesor i mikroprocesor odnosno kontroler i mikrokontroler se često miješaju
 - mi ćemo koristiti nazive
 - procesori (P), mikroprocesori (μ P)
 - kontroleri (C), mikrokontroleri (μ C)
 - u engleskoj literaturi se koriste nazivi
 - *microprocessor* (*MicroProcessor Unit*, MPU)
 - *microcontroller* (*MicroController Unit*, MCU)

- izvedbe procesora
 - s aspekta implementacijske tehnologije
 - **procesor s mekom jezgrom**
(*Soft Processor* ili *Soft-Core Processor*)
 - napravljen je kao formalni model korištenjem jezika za opis sklopolja (VHDL, Verilog i sl.)
 - ⇒ može biti implementiran na raznim platformma (VLSI, FPGA)
 - ⇒ može se konfigurirati prije same implementacije (npr. može se definirati veličina adresnog prostora i sl.)
 - **fiksno ožičeni odnosno implementirani procesori**
(*Hard-Core Processor* ili *Hard-Wired Processor*)
 - fizička komponenta implementirana u VLSI tehnologiji
 - s aspekta primjene
 - procesori namijenjeni širem rasponu aplikacija (*Application Processors*)
 - koriste u sustavima koji sadrže veći broj različitih aplikacija
 - podržavaju rad sa složenim operacijskim sustavima
 - ugradbeni procesori (*Embedded Processors*)
 - najčešće se koriste u specifičnoj aplikaciji
 - primjene kod kojih se traži rad u stvarnom vremenu (*Real Time*)
 - obično imaju manje mogućnosti
 - često se izvode s malom potrošnjom, malim naponom napajanja i sl.
 - procesori opće namjene (*General Purpose Processors*)
 - koriste u sustavima koji sadrže veći broj različitih aplikacija
 - po složenosti kompromis između gornja dva tipa
 - velik broj procesora s mekom jezgrom na tržištu postoji i kao fiksno ožičen
 - podjela s aspekta primjene nije čvrsta podjela
 - ⇒ češće je to podjela unutar koje proizvođači klasificiraju vlastite procesore

- *System on Chip Design*, SoC
 - danas se teži implementirati što veći dio elektroničkog sustava u jednom integriranom sklopu (*System on Chip Design*, SoC)
 - prednosti
 - ⇒ niža cijena proizvoda
 - ⇒ kraće vrijeme razvoja
 - ⇒ manje problema s miješanjem tehnologija
 - ⇒ npr. izlaz signala na priključak integriranog sklopa traži manju frekvenciju takta, manju brzinu prijenosa i sl.
 - primjer SoC dizajna
 - mikrokontrolери opće namjene
 - kontroleri raznih strojeva (npr. stroj za pranje rublja)
 - razni sustavi implementirani u FPGA tehnologiji
 - SoC često na istom chip-u sadrži digitalne i analogne komponente
 - digitalne komponente
 - procesori, memorije, periferije, i sl.
 - analogne komponente
 - pojačala, analogne sklopke, regulatori napajanja, i sl.
 - miješanje analogno digitalne komponente
 - A/D pretvornici, D/A pretvornici

2.2 Procesori s ARM arhitekturom

- vidi predmet *Arhitektura računala 1*
- ARM procesori
 - ubrajaju se među najpoznatije i najviše korišene procesore s mekom jezgrom
- kompanija *ARM Ltd.*
 - osnovana 1990. kao *Advanced RISC Machines Ltd.* (zajednička investicija kompanija *Acorn Computers*, *Apple Computer* and *VLSI Technology*)
 - najpoznatiji su po razvoju 32 bitnih procesorskih jezgri
 - razvijaju i pripadajuće alate (biblioteke, programsku podršku, sustave za uhodavanje i debugiranje i sl.)
 - proizvodi isključivo sklopovske jezgre (*Intellectual Property Core*, *IP Core*)
 - ⇒ njihov konačan proizvod su datoteke
 - implementaciju procesora izvodi kupac
- ARM procesori ugrađuju se u čitav niz uređaja kao što su
 - mobilni telefone, GPS uređaji, kamere, digitalni televizori, tvrdi diskove, iPod, iPhone, igračke Nintendo Game Boy i Playstation, itd
- ARM procesori temelje se na RISC arhitekturi (*Reduced Instruction Set Computer*)
- potrebno je razlikovati
 - 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

- arhitekture se označavaju
 - inačica
ARMv1, ARM v2,...
 - ekstenzije koja govori da arhitektura podržava ili ne podržava opcije (mnoge ove opcije bit će objašnjene u dalnjem tekstu)
 - T *Thumb* skup instrukcija
 - D *Debugger*
 - M množilo $32 \times 32 \rightarrow 64$
 - I *In-Circuit Emulator*
 - J Jezelle skup instrukcija
 - X slijedeća opcija ne postoji u danoj arhitekturi
itd.
- oznake mekih jezgara sadrže
 - oznaku familije
ARM6, ARM7 i sl.
 - opcije
obično su iste kao u gornjem slučaju
- dosad je su razvijene arhitekture ARMv1 do ARMv7
 - dana više nisu podržane (zastarjele su) arhitekture
 - razne inačice arhitektura ARMv1, ARMv2, ARMv3
⇒ koristile su 26-bitnu arhitekturu
 - ARMv4xM, ARMv4TxM
 - ARMv5, ARMv5xM, and ARMv5TxM
 - danas su podržane arhitekture
 - ARMv4, ARMv4T
 - ARMv5T, ARMv5TE, ARMv5TEJ
 - ARMv6
 - ARMv7

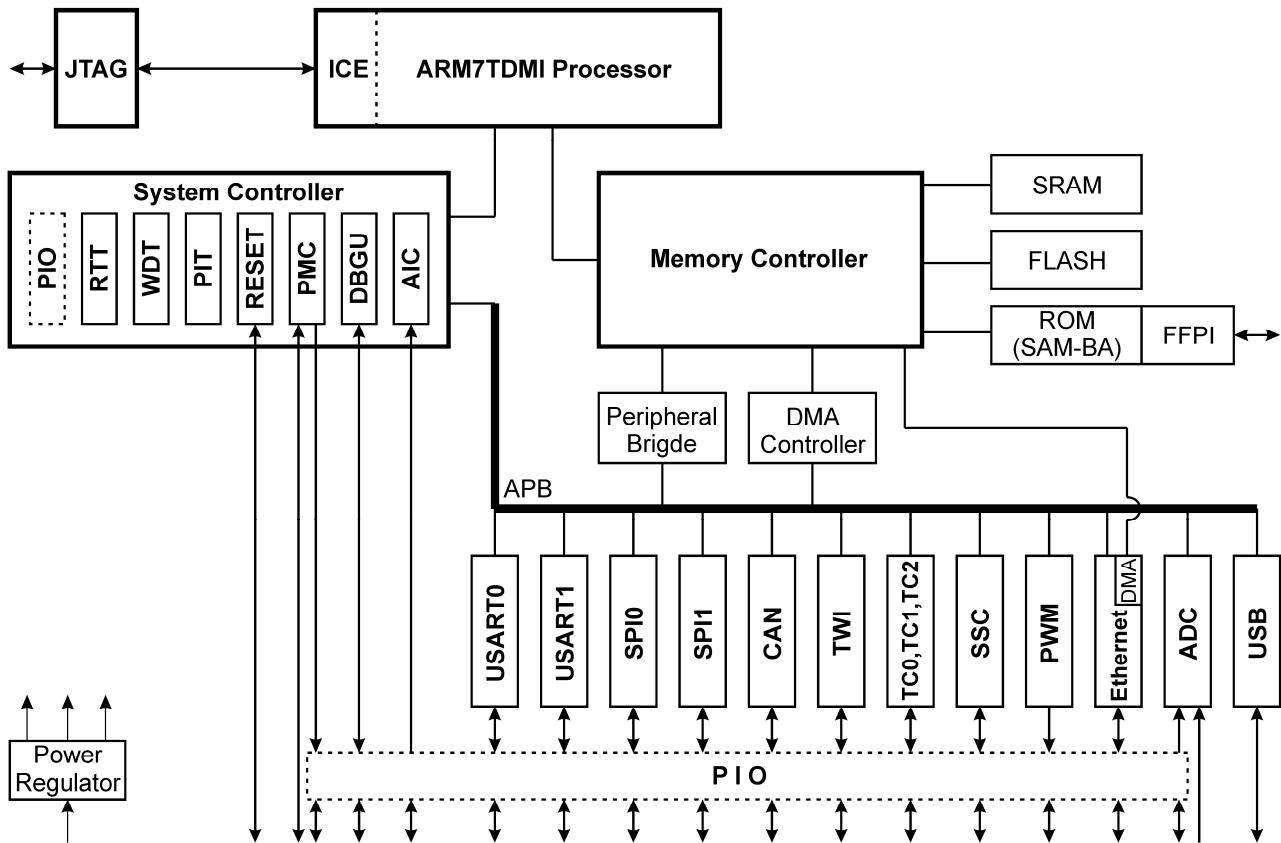
- u slijedećim poglavljima bit će detaljno obrađeni mikrokontroleri familije AT91SAM7X, proizvođača Atmel Corporation
 - mikrokontroleri familije AT91SAM7X koriste procesorsku jezgru ARM7TDMI
 - procesorska jezgra ARM7TDMI je implementacija ARMv4T arhitekture
- literatura koja pokriva ovo područje obuhvaća
 - datasheet familije mikrokontrolera AT91 ARM Thumb-based Microcontrollers, Datasheets, Atmel 2007
 - manual procesorske jezgre ARM7TDMI Technical Reference Manual, Revision r4p1, ARM 2004
 - manual koji opisuje arhitekturu ARM Architecture Reference Manual, ARM 2005

2.3 Pitanja za provjeru znanja

1. Objasniti razliku između procesora odnosno mikroprocesora i kontrolera odnosno mikrokontrolera?
2. Što se podrazumijeva pod pojmom mikro u nazivu mikroprocesor odnosno mikrokontroler?
3. Koje izvedbe procesora postoje obzirom na implementacijske tehnologije? Kakve vrste procesora razlikujemo obzirom na područje primjene?
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?
5. Na primjeru ARM tehnologije, objasniti razliku između pojmove: arhitektura, procesorska jezgra i (μ)procesor odnosno (μ)kontroler.
6. Opisati način označavanja ARM arhitektura.

3 Mikrokontroleri familije AT91SAM7X

3.1 Osnovna blokovska shema



- komponente mikrokontrolera
 - procesor
 - sklopovlje za upravljanje memorijom (*Memory Controller*)
 - sklopovlje za upravljanje radom sustava (*System Controller*)
 - periferno sklopovlje (*Peripherals*)
 - unutrašnji izvori napajanja (*Power Regulator*)

3.2 Procesorska jezgra ARM7TDMI

- TDMI=Thumb Instruction Set, Debugger, Multiplier, In Circuit Emulator

3.2.1 Arhitektura

- von Neumann arhitektura
- 32 bitni procesor
- radi s podacima širine
 - 32 bita, riječ (*Word*)
 - 16 bitova, poluriječ (*Halfword*)
 - 8 bitova, bajt (*Byte*)
- podaci moraju biti poravnati (*Alignment*)
 - npr. 32 bitni podatak zauzima 4 bajta počevši od adrese koja je višekratnik broja 4 (0, 4, 8, 16,)

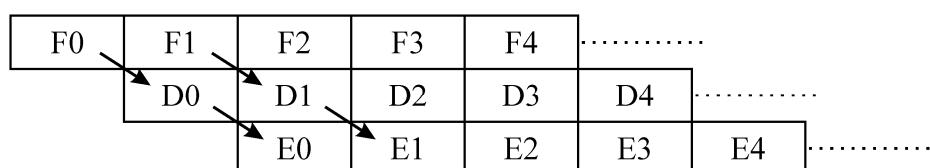
- procesorska jezgra podržava oba zapisa podataka u memoriji
 - little endian (češće se koristi)

| 31-24 | 23-16 | 15-8 | 7-0 |
|--------------------|--------------------|--------------------|------------------|
| bajt na adresi A+3 | bajt na adresi A+2 | bajt na adresi A+1 | bajt na adresi A |

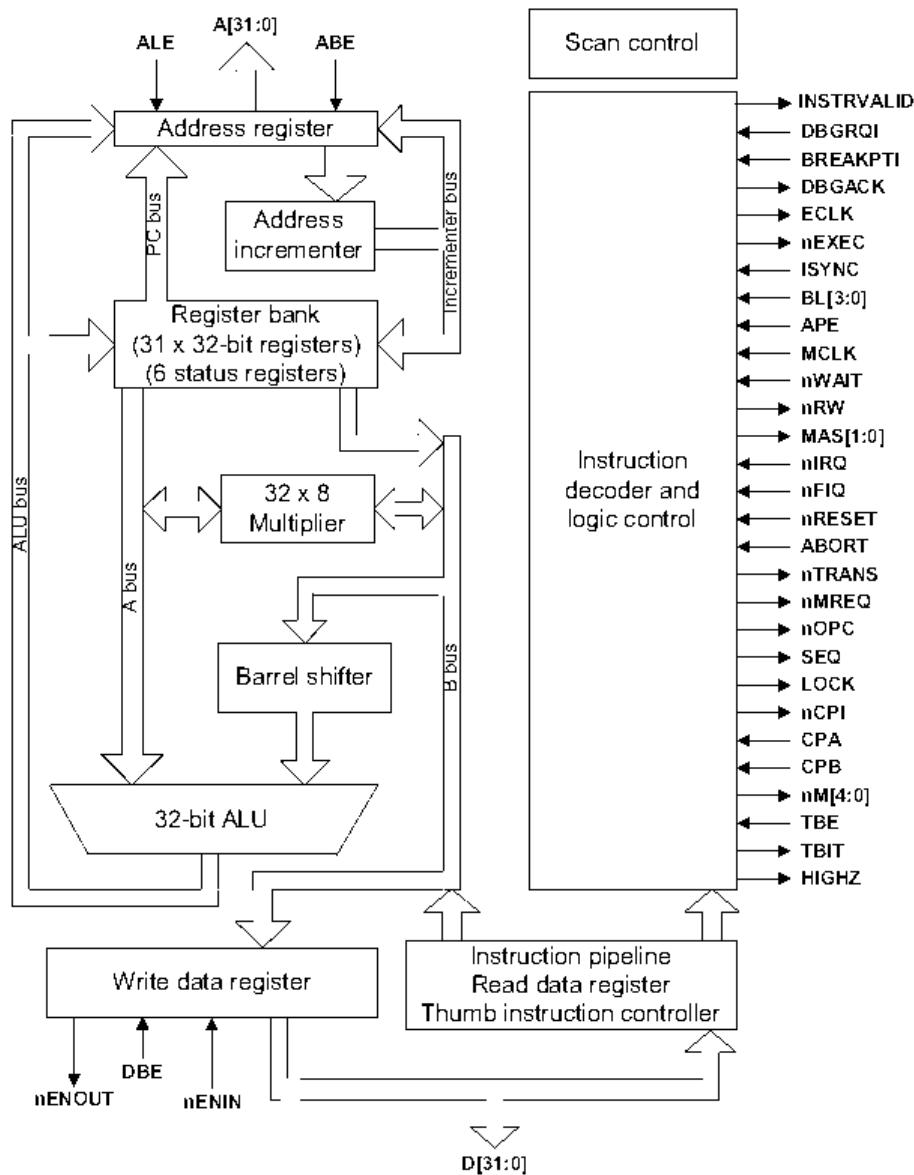
- big endian

| 31-24 | 23-16 | 15-8 | 7-0 |
|------------------|--------------------|--------------------|--------------------|
| bajt na adresi A | bajt na adresi A+1 | bajt na adresi A+2 | bajt na adresi A+3 |

- endian se na jezgri ARM7TDMI odabire stanjem signala BIGEND
- neki mikrokontroleri koji imaju jezgru ARM7TDMI **ne podržavaju** odabir endiana (npr. AT91SAM7X)
- 3-stupanjska protočna arhitektora



- blokovska shema procesorske jezgre



- načini rada procesora (*Operating Modes*) - ukupno 7 načina rada
 - neprivilegirani način rada
 - *User*
 - privilegirani način rada
 - iznimke (*Exception modes*)
 - *Fast Interrupt Mode, FIQ*
 - *Interrupt Mode, IRQ*
 - *Supervisor*
 - *Abort*
 - *Undefined*
 - regularni rad (nije iznimka)
 - *System*

- **neprivilegirani način rada**
 - program koji se izvršava NE može pristupiti svim resursima sustava
 - iz ovog načina rada može se preći u privilegirani jedino u slučaju iznimke
- **privilegirani način rada**
 - program koji se izvršava može pristupiti svim resursima sustava
 - npr. jedina razlika između *User* i *System* načina rada je u tome što program u *System* načinu rada može pristupiti svim resursima

3.2.2 Registri

- ukupno 37 registara
 - 31 registar opće namjene
 - 6 statusnih registara
- svi registri imaju širinu od 32 bita

3.2.2.1 Registri opće namjene

- organizacija registara opće namjene

| Neprivileg. način rada | Privilegirani način rada | | | | | | |
|---------------------------|-----------------------------|------------|----------|-----------|-----------|-------------------|----------|
| | Iznimke | | | | | | |
| User | System | Supervisor | Abort | Undefined | Interrupt | Fast Interrupt | |
| R0 | R0 | R0 | R0 | R0 | R0 | R0 | |
| R1 | R1 | R1 | R1 | R1 | R1 | R1 | |
| R2 | R2 | R2 | R2 | R2 | R2 | R2 | |
| R3 | R3 | R3 | R3 | R3 | R3 | R3 | |
| R4 | R4 | R4 | R4 | R4 | R4 | R4 | |
| R5 | R5 | R5 | R5 | R5 | R5 | R5 | |
| R6 | R6 | R6 | R6 | R6 | R6 | R6 | |
| R7 | R7 | R7 | R7 | R7 | R7 | R7 | |
| R8 | R8 | R8 | R8 | R8 | R8 | R8 | R8_fiq |
| R9 | R9 | R9 | R9 | R9 | R9 | R9 | R9_fiq |
| R10 | R10 | R10 | R10 | R10 | R10 | R10 | R10_fiq |
| R11 | R11 | R11 | R11 | R11 | R11 | R11 | R11_fiq |
| R12 | R12 | R12 | R12 | R12 | R12 | R12 | R12_fiq |
| R13 | R13 | R13_svc | R13_abt | R13_und | R13_irq | R13_fiq | |
| R14 | R14 | R14_svc | R14_abt | R14_und | R14_irq | R14_fiq | |
| PC (R15) | PC (R15) | PC (R15) | PC (R15) | PC (R15) | PC (R15) | PC (R15) | PC (R15) |

- u praksi se koriste i nazivi
 - niži registri (*Low Registers*) \Rightarrow R0-R7
 - viši registri (*High Registers*) \Rightarrow R8-R15
- uočiti
 - niži registri su zajednički za sve načine rada
 - žuto označeni registri se razlikuju za pojedine načine rada (postoji banka paralelnih registara)
- posebne funkcije registara
 - R13 koristi se kao pokazivač stoga (*Stack Pointer*)
 - R14 sadrži povratnu adresu potprograma ili programa koji obrađuje iznimku (*Link Register*, LR)
 - R15 je PC

3.2.2.2 Statusni registri

- organizacija statusnih registara

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

- registrovani trenutnih stanja, **CPSR** (*Current Program Status Register*)
 - vidljiv je u svim načinima rada
 - sadrži zastavice i upravljačke bitove
- registrovani 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] |

- zastavice
 - N** Negative
 - Z** Zero
 - C** Carry
 - V** oVerflow
- upravljački bitovi
 - I** IRQ disable
 - F** FIQ disable
 - T** skup instrukcija: 1=Thumb, 0=ARM (vidi daljnji tekst)
 - M[0:4]** način rada
- rezervirani bitovi
 - Reserved** neiskorišteni ali rezervirani bitovi
⇒ u njih korisnik ne smije upisivati
- uočiti
 - registrovani SPSR vidljiv je jedino u načinima rada koji obuhvaćaju iznimke

3.2.3 Instrukcije

- ARM7TDMI jezgra
 - koristi ARMv4T arhitekturu
 - RISC

3.2.3.1 ARM i Thumb skup instrukcija

- ARM7TDMI jezgra podržava dva skupa instrukcija
 - **ARM skup instrukcija**
 - kodna riječ ima širinu 32 bita
⇒ veće mogućnosti
 - **Thumb skup instrukcija**
 - kodna riječ ima širinu 16 bita
⇒ manje mesta u programskom prostoru
- unutar procesora su oba skupa instrukcija implementirana na istoj, 32-bitnoj arhitekturi
 - skupovi instrukcija su kompatibilni
 - zbog kraće kodne riječi, neki se registri ne mogu dohvatiti izravno (postoji način kako se ipak mogu iskoristiti)
 - mapiranje registara

| Thumb skup instrukcija | | ARM skup instrukcija |
|--------------------------------------|---|--------------------------------------|
| R0 | → | R0 |
| R1 | → | R1 |
| R2 | → | R2 |
| R3 | → | R3 |
| R4 | → | R4 |
| R5 | → | R5 |
| R6 | → | R6 |
| R7 | → | R7 |
| | | R8 |
| | | R9 |
| | | R10 |
| | | R11 |
| | | R12 |
| R13 (<i>Stack Pointer, SP</i>) | → | R13 (<i>Stack Pointer, SP</i>) |
| R14 (<i>Link Register, LR</i>) | → | R14 (<i>Link Register, LR</i>) |
| (R15) (<i>Program Counter, PC</i>) | → | (R15) (<i>Program Counter, PC</i>) |
| CPSR | → | CPSR |
| SPSR | → | SPSR |

- kodne riječi Thumb i ARM skupa instrukcija se razlikuju
- koji skup instrukcija se koristi
 - određeno je bitom T u CPSR
 - alati za razvoj programske podrške imaju opciju za generiranje ARM odnosno Thumb coda
- u dalnjem tekstu je dan pregled instrukcije bez uloženja u detalje s ciljem da se dobije pregled nad cijelim skupom instrukcija

3.2.3.2 Uvjeti

- većina instrukcija može biti izvršena uvjetno
 - zastavice u CPSR: N, Z, C, i V
- tablica uvjeta

| <cond> | Značenje | Stanje zastavica |
|---------------------|-----------------------------------|-------------------------|
| EQ | Equal | Z=1 |
| NE | Not equal | Z=0 |
| CS/HS | Carry set/unsigned higher or same | C=1 |
| CC/LO | Carry clear/unsigned lower | C=0 |
| MI | Minus/negative | N=1 |
| PL | Plus/positive or zero | N=0 |
| VS | Overflow | V=1 |
| VC | No overflow | V=0 |
| HI | Unsigned higher | C=1, Z=0 |
| LS | Unsigned lower or same | C=0, Z=1 |
| GE | Signed greater than or equal | N==V |
| LT | Signed less than | N!=V |
| GT | Signed greater than | Z==0, N==V |
| LE | Signed less than or equal | Z==0, N!=V |
| AL | Always (unconditional) | - |

3.2.3.3 Prijenos podataka

- *Load and Store Instructions*

- učitavanje podataka u registre

| | | |
|-------|--|--|
| LDR | LDR{<cond>} <Rd>, <addr_mode> | <i>Load Register</i> , Učitava riječ iz memorije |
| LDRSH | LDR{<cond>}SH <Rd>, <addr_mode> | <i>Load Register Signed Halfword</i> ; Učitava poluriječi iz memorije i predznačno ju proširuje na riječ |
| LDRSB | LDR{<cond>}SB <Rd>, <addr_mode> | <i>Load Register Signed Byte</i> , Učitava bajta iz memorije i predznačno ga proširuje na riječ |
| LDRH | LDR{<cond>}H <Rd>, <addr_mode> | <i>Load Register Halfword</i> , Učitava poluriječi iz memorije i proširuje su s nulama na riječ |
| LDRB | LDR{<cond>}B <Rd>, <addr_mode> | <i>Load Register Byte</i> , Učitava bajt iz memorije i proširuje ju s nulama na riječ |
| LDRBT | LDR{<cond>}BT <Rd>, <post_indexed_addr_mode> | <i>Load Register Byte with Translation</i> , Učitava bajt iz memorije i proširuje ju s nulama na riječ |
| LDRT | LDR{<cond>}T <Rd>, <post_indexed_addr_mode> | <i>Load Register with Translation</i> , Učitava riječ iz memorije |
| LDM | LDM{<cond>}<addr_mode> <Rn>{!}, <registers> | <i>Load Multiple</i> , Učitava podatke u više registara. |

- primjeri instrukcija za učitavanje podatka u jedan registar

```

LDR R1, [R0]      ; Učitava R1 s adrese u R0
LDR R12, [R13, #-4] ; Učitava R12 s adrese R13-4.

LDRB R3, [R8, -R4] ; Učitava bajt u R3 s adrese R8-R4.
                     ; Viša 3 bajta su 0.

LDR R0, [PC, #40]  ; Load R0 from PC + 0x40
                     ; (adresa instrukcije LDR)+8+0x40
    
```

- primjeri instrukcija za učitavanje podatka u više registara

```

LDM R0, {R5-R8} ; Učitava R5,R6,R7 i R8 s adresom
                  ; koje počinju od R0.
    
```

- korištenje uvjeta

- uvjet zamjenjuje <cond> u mnemoniku instrukcije

```

LDR R1, [R0]      ; Učitava R1 s adrese u R0.
LDRCS R1, [R0]     ; Učitava R1 s adrese u R0
                   ; pod uvjetom da je C=1 (Carry Set).
    
```

- načini adresiranja (*Addressing Modes*)
 - adresiranje s pomakom (*offset addressing*)
 - adresa se dobiva zbrajanjem ili oduzimanjem pomaka od vrijednosti registra
 - adresiranje s prethodnim računanjem adrese (*pre-indexed addressing*)
 - adresa se dobiva zbrajanjem ili oduzimanjem pomaka od vrijednosti registra
 - dobivena adresa se spremi u registar
 - adresiranje s naknadnim računanjem adrese (*post-indexed addressing*)
 - adresa se uzima iz registra
 - pomak se zbraja ili oduzima od vrijednosti registra i dobivena vrijednost se spremi u registar
- pohranjivanje podataka u memoriju

| | | |
|-------|--|--|
| STR | STR{<cond>} <Rd>, <addr_mode> | <i>Store Register</i> , Pohranjuje riječ iz registra u memoriju |
| STRH | STR{<cond>}H <Rd>, <addr_mode> | <i>Store Register Halfword</i> , Pohranjuje poluriječ iz registra (niža 2 bajta) u memoriju |
| STRB | STR{<cond>}B <Rd>, <addr_mode> | <i>Store Register Byte</i> , Pohranjuje najniži bajt iz registra u memoriju |
| STRBT | STR{<cond>}BT <Rd>, <post_indexed_addressing_mode> | <i>Store Register Byte with Translation</i> , Pohranjuje poluriječ iz registra (niža 2 bajta) u memoriju |
| STRT | STR{<cond>}T <Rd>, <post_indexed_addressing_mode> | <i>Store Register with Translation</i> , Pohranjuje riječ iz registra u memoriju |
| STM | STM{<cond>}<addressing_mode> <Rn>{!}, <registers> | <i>Store Multiple</i> , Pohranjuje podatke iz više registara u memoriju. |

- zamjena podataka između registra i memorije

| | | |
|------|---------------------------------|---|
| SWP | SWP{<cond>} <Rd>, <Rm>, [<Rn>] | Swap; Zamjenjuje riječ između registra i memorije |
| SWPB | SWP{<cond>}B <Rd>, <Rm>, [<Rn>] | Swap; Zamjenjuje bajta između registra i memorije |

- primjer

```
SWP R12, R10, [R9] ; Učitava R12 sa adresu R9
; Sprema R10 na adresu R9
```

3.2.3.4 Obrada podataka

- *Data Processing Instructions*
- zbrajanje i oduzimanje

| | | |
|-----|-------------------------------------|---|
| ADD | ADD{<cond>} {S} <Rd>, <Rn>, <sh_op> | Zbraja dvije vrijednosti. Prva je iz registra, a druga je iz registra ili je konstanta. Druga se može posmagnuti prije zbrajanja. |
| ADC | ADC{<cond>} {S} <Rd>, <Rn>, <sh_op> | Zbraja dvije vrijednosti i <i>Carry flag</i> . Prva je vrijednost iz registra, a druga je iz registra ili je konstanta. Druga se može posmagnuti prije zbrajanja. |
| SUB | SUB{<cond>} {S} <Rd>, <Rn>, <sh_op> | Oduzima dvije vrijednosti. Prva je iz registra, a druga je iz registra ili je konstanta. Druga se može posmagnuti prije oduzimanja. |
| SBC | SBC{<cond>} {S} <Rd>, <Rn>, <sh_op> | Oduzima dvije vrijednosti i komplement <i>Carry flag-a</i> . Prva je vrijednost iz registra, a druga je iz registra ili je konstanta. Druga se može posmagnuti prije oduzimanja. |
| RSB | RSB{<cond>} {S} <Rd>, <Rn>, <sh_op> | Reverse Subtract; Oduzima prvu vrijednost od druge. Druga vrijednost je sadržaj registra ili konstanta, a može se posmagnuti prije oduzimanja. |
| RSC | RSC{<cond>} {S} <Rd>, <Rn>, <sh_op> | Reverse Subtract with Carry; Oduzima prvu vrijednost od druge i uzima u obzir <i>Carry</i> . Druga vrijednost je sadržaj registra ili konstanta, a može se posmagnuti prije oduzimanja. |

- primjeri

```

ADD R3, R4, #2      ; R3=R4+2
ADD R4, R3, R2      ; R4=R3+R2

ADDS R4, R3, R2      ; R4=R3+R2, postavljuju se zastavice

ADD R9, R5, R5, LSL #3    ; R9=R5+R5*8 ili R9=R5*9

```

- podržane su slijedeći posmaci, <sh_op> ...
 - ASR *Arithmetic shift right*
 - LSL *Logical shift left*
 - LSR *Logical shift right*
 - ROR *Rotate right*
 - RRX *Rotate right with extend (Carry flag)*

```

SUB R1, R2, #3      ; R1=R2-3
RSB R1, R2, #3      ; R1=3-R2

• uočiti
RSB R1, R2, #0 ; R1=-R2

```

- logičke operacije

| | | |
|-----|-------------------------------------|---|
| AND | AND{<cond>} {S} <Rd>, <Rn>, <sh_op> | Izvodi bit po bit I operaciju između dvije vrijednosti. Prva je iz registra, a druga je iz registra ili je konstanta. Druga se može posmknuti. |
| ORR | ORR{<cond>} {S} <Rd>, <Rn>, <sh_op> | Izvodi bit po bit ILI operaciju između dvije vrijednosti. Prva je iz registra, a druga je iz registra ili je konstanta. Druga se može posmknuti. |
| EOR | EOR{<cond>} {S} <Rd>, <Rn>, <sh_op> | Izvodi bit po bit EX-ILI operaciju između dvije vrijednosti. Prva je iz registra, a druga je iz registra ili je konstanta. Druga se može posmknuti. |
| BIC | BIC{<cond>} {S} <Rd>, <Rn>, <sh_op> | <i>Bit Clear</i> , Izvodi bit po bit I operaciju između dvije vrijednosti. Prva je iz registra, a druga je komplement vrijednosti iz registra ili komplement konstante. Druga se vrijednost može posmknuti. |

- upis vrijednosti u register

| | | |
|-----|-------------------------------|--|
| MOV | MOV{<cond>} {S} <Rd>, <sh_op> | Move; Upisuje vrijednost u register. Vrijednost je sadržaj registra ili konstanta. Vrijednost se može posmknuti prije upisa. |
| MVN | MVN{<cond>} {S} <Rd>, <sh_op> | Move Not; Upisuje jedinični komplement vrijednosti u register. Vrijednost je sadržaj registra ili konstanta. Vrijednost se može posmknuti prije upisa. |

- usporedbe

| | | |
|-----|---------------------------|---|
| CMP | CMP{<cond>} <Rn>, <sh_op> | Compare Positive, Uspoređuje dvije vrijednosti tako da ih oduzima. Prva je iz registra, a druga je iz registra ili je konstanta. Druga se može posmknuti. |
| CMN | CMN{<cond>} <Rn>, <sh_op> | Compare Negative, Uspoređuje vrijednost i komplement druge tako da ih oduzima. Prva vrijednost je iz registra, a druga je iz registra ili je konstanta. Druga vrijednost se može posmknuti. |
| TST | TST{<cond>} <Rn>, <sh_op> | Uspoređuje dvije vrijednosti tako da radi logički I. Prva vrijednost je iz registra, a druga je iz registra ili je konstanta. Druga vrijednost se može posmknuti. |
| TEQ | TEQ{<cond>} <Rn>, <sh_op> | Uspoređuje dvije vrijednosti tako da radi logički XOR. Prva vrijednost je iz registra, a druga je iz registra ili je konstanta. Druga vrijednost se može posmknuti. |

- množenja

| | | |
|-------|---|---|
| MUL | MUL{<cond>}{S} <Rd>, <Rm>, <Rs> | <i>Multiply</i> ; Množi dva 32 bitna broja (s predznakom ili bez). Rezultat ima 32 niža bita. |
| MLA | MLA{<cond>}{S} <Rd>, <Rm>, <Rs>, <Rn> | <i>Multiply Accumulate</i> ; Množi dva 32 bitna broja (s predznakom ili bez) i pribraja im treći. Rezultat ima 32 niža bita. |
| UMULL | UMULL{<cond>}{S} <RdLo>, <RdHi>, <Rm>, <Rs> | <i>Unsigned Multiply Long</i> ; Množi dva 32-bitna broja bez predznaka. Rezultat ima 64 bita. |
| UMLAL | UMLAL{<cond>}{S} <RdLo>, <RdHi>, <Rm>, <Rs> | <i>Unsigned Multiply Accumulate Long</i> ; Množi 32-bitna dva broja bez predznaka i 64-bitnom rezultatu pribraja 64-bitni broj. Krajnji rezultat ima 64 bita. |
| SMULL | SMULL{<cond>}{S} <RdLo>, <RdHi>, <Rm>, <Rs> | <i>Signed Multiply Long</i> ; Množi dva 32-bitna broja s predznakom. Rezultat ima 64 bita. |
| SMLAL | SMLAL{<cond>}{S} <RdLo>, <RdHi>, <Rm>, <Rs> | <i>Signed Multiply Accumulate Long</i> ; Množi 32-bitna dva broja s predznakom i 64-bitnom rezultatu pribraja 64-bitni broj. Krajnji rezultat ima 64 bita. |

- primjer

```

MUL R4, R2, R1      ; R4=R2*R1
MULS R4, R2, R1     ; R4=R2*R1
                     ; Postavlja zastavice N i Z
MLA R7, R8, R9, R3  ; R7=R8*R9+R3
SMULL R4, R8, R2, R3; R4= bitovi 0 do 31 od R2*R3
                     ; R8= bitovi 32 do 63 od R2*R3

```

- uočiti

- MLA i UMLAL su pogodne za implementaciju konvolucije

3.2.3.5 Grananja

- *Branch Instructions*

| | | |
|----|-----------------------------|--|
| B | B{<cond>} <target_address> | <i>Branch, Skok na adresu</i> |
| BL | BL{<cond>} <target_address> | <i>Branch and Link, Skok na adresu. Sprema povratnu adresu u R14 (LR)</i> |
| BX | BX{<cond>} <Rm> | <i>Branch and Exchange, Skače na adresu Rm. Mjenja instrukcijski set, T=Rm[0]. (0 je ARM, 1 je Thumb set instrukcija).</i> |

- primjer

```

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

```

3.2.3.6 Rad sa statusnim registrima

- *Status Register Transfer Instructions*

| | | |
|-----|--|--|
| MSR | MSR{<cond>} CPSR_<fields>, #<immediate> MSR{<cond>} CPSR_<fields>, <Rm> MSR{<cond>} SPSR_<fields>, #<immediate> MSR{<cond>} SPSR_<fields>, <Rm> | <i>Move to Status Register;</i> Upisuje u statusni registar sadržaj regista ili konstantu |
| MRS | MRS{<cond>} <Rd>, CPSR MRS{<cond>} <Rd>, SPSR | <i>Move PSR to general-purpose register;</i> Upisuje vrijednost staturnosg registra u registar opće namjene. |

3.2.3.7 Generiranje iznimke

- *Exception-Generating Instructions*

| | | |
|-----|------------------------|---|
| SWI | SWI{<cond>} <immed_24> | <i>Software Interrupt;</i> Uzrokuje programski prekid |
|-----|------------------------|---|

- napomena

- parametar immed_24 se ignorira
 \Rightarrow SWI uzrokuje skok na adresu 0x0000 0008
(vidi poglavlje o iznimkama)
- immed_24 može poslužiti da se naknadno otkrije koji dio koda je zatražio prekid

3.2.3.8 Rad s koprocesorom

- *Coprocessor Instructions*
- postoje 5 instrukcija za rad s koprocesorom
LDC, CDP, MCR, MRC, CTC
- ove instrukcije služe za
 - inicijalizaciju rada koprocesora
 - prijenos podataka u koprocesor
 - prijenos rezultata iz koprocesora
- ovdje nećemo davati detaljan opis ovih instrukcija

3.2.3.9 DSP i JBC instrukcije

- uočiti
 - opisana procesorska jezgra NE podržava
 - instrukcije karakteristične za digitalnu obradu signala (*Digital Signal Processing*, DSP)
 - instrukcije za izvođenje Java bajt-koda (*Jazelle DBX*, *Jazelle Direct Bytecode eXecution*)
- neke ARM procesorske jezgre podržavaju instrukcije karakteristične za digitalnu obradu signala
 - primjer - aritmetika sa zasićenjem
 - "obično" zbrajanje (dvojni komplement)
$$\begin{array}{r} 0110_{(2)} \rightarrow 6_{(10)} \\ +0100_{(2)} \rightarrow 4_{(10)} \\ \hline 1010_{(2)} \rightarrow -6_{(10)} \end{array}$$
 - zbrajanje sa zasićenjem (dvojni komplement)
$$\begin{array}{r} 0110_{(2)} \rightarrow 6_{(10)} \\ +0100_{(2)} \rightarrow 4_{(10)} \\ \hline 0111_{(2)} \rightarrow 7_{(10)} \end{array}$$

3.2.4 Obrada iznimaka

- iznimka (*Exception*)
 - nastaje kad se normalan tok programa mora privremeno prekinuti
 - primjer
 - reset
 - posluživanje zahtjeva za prekid
 - obrada nepostojeće instrukcije
- nakon ulaska u iznimku uvijek se koristi ARM skup instrukcija
- iznimke (ARM7TDMI jezgra)

| Iznimka | Vektor | Način rada | Izvor |
|--|-------------|----------------|---|
| Reset | 0x0000 0000 | Supervisor | Aktivno stanje signala na vanjskom priključku za reset |
| Nedefinirana instrukcija (<i>Undefined instruction</i>) | 0x0000 0004 | Undefined | Pojava instrukcije s nedefiniranim operacijskim kodom |
| Programski prekid (<i>Software interrupt</i>) | 0x0000 0008 | Supervisor | Instrukcija SWI (<i>Software Interrupt</i>) |
| Nedefinirana adresa instrukcije (<i>Prefetch abort</i>) | 0x0000 000C | Abort | Dohvat instrukcije s nepostojeće ili neporavnate adrese (<i>Memory Controller</i>) |
| Nedefinirana adresa podatka (<i>Data abort</i>) | 0x0000 0010 | Abort | Dohvat podatka s nepostojeće ili neporavnate adrese (<i>Memory Controller</i>) |
| Rezervirano | 0x0000 0014 | - | - |
| IRQ | 0x0000 0018 | Interrupt | Vanjski zahtjev za prekid |
| FIQ | 0x0000 001C | Fast Interrupt | Vanjski zahtjev za prekid |

- uočiti
 - IRQ i FIQ se mogu onemogućiti postavljanjem I=1 odnosno F=1

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

- prioriteti prekida
 - Reset najviši prioritet
 - Data Abort
 - FIQ
 - IRQ
 - Prefetch Abort
 - Nedefinirana instrukcija ili SWI najniži prioritet
- uočiti
 - nedefinirana instrukcija i SWI imaju isti prioritet
⇒ ako je došlo do SWI, instrukcija je sigurno regularna !
 - procesor ima samo 2 linije za vanjski prekid (IRQ i FIQ)
⇒ za posluživanje više od 2 jedinice koje mogu tražiti prekid treba dodatno sklopovlje
⇒ vidi "System Controller / Advanced Interrupt Controller"

3.2.5 Reset

- stanje procesora nakon reseta
 - PC=0x0000 0000
 - M[4:0]=0b10011 ⇒ *Supervisor* način rada
 - I=1 ⇒ Onemogućen IRQ
 - F=1 ⇒ Onemogućen FIQ

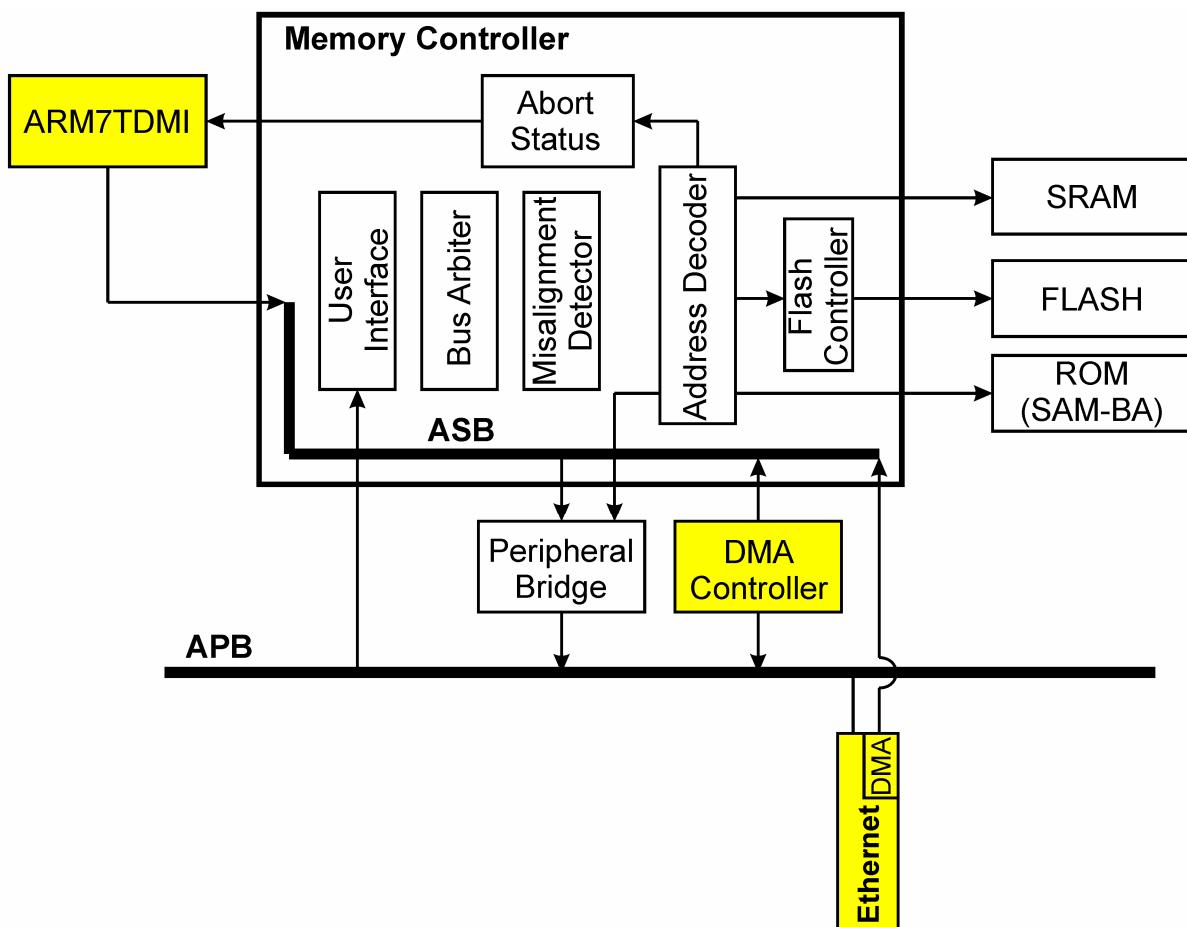
3.3 Memorije

3.3.1 Izvedbe memorija

- postoje 3 inačice mikrokontrolera koje se razlikuju po količini memorije
 - AT91SAM7X128 ⇒ Flash: 128 KB, SRAM: 32 KB
 - AT91SAM7X256 ⇒ Flash: 256 KB, SRAM: 64 KB
 - AT91SAM7X512 ⇒ Flash: 512 KB, SRAM: 128 KB
- Flash memorija se briše impulsom na priključku ERASE

3.3.2 Sklopoljje za upravljanje memorijama

- pojednostavljena blokovska shema



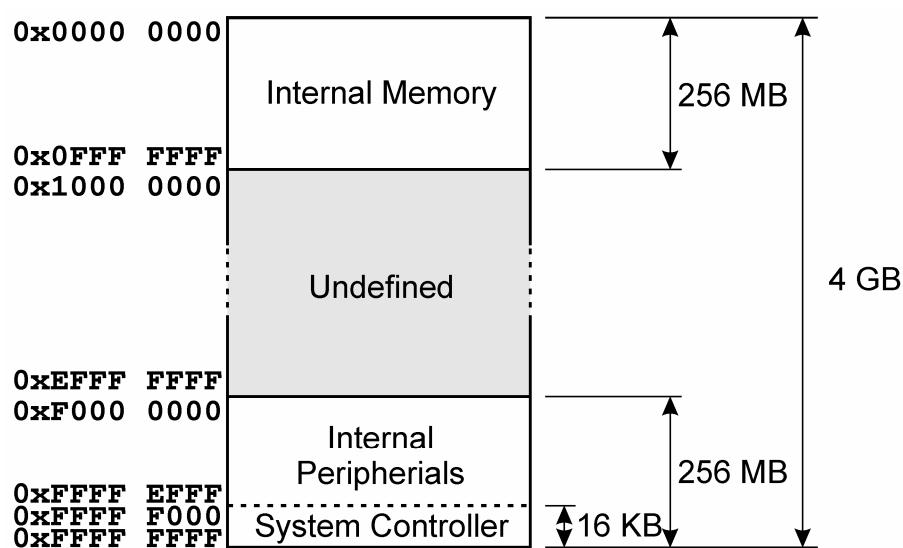
- žuto su označeni upravljači (*Bus Master*)
- strelice označavaju tok podataka od upravljača prema izvršiocima

- uočiti
 - korisničko sučelje (*User Interface*) čine registri za konfiguraciju
⇒ ima ukupno 9 registara
 - ovim registrima pristupa se kao i registrima periferija
- memorijski kontroler (*Memory Controller*) izvodi
 - dekodiranje adrese (*Address Decoder*, AD)
⇒ npr. 32 KB SRAM-a vidi se počevši od adrese 0x0020 0000
 - mapiranje neke od memorija u boot memorijski prostor
⇒ boot memorija počinje od adrese 0x0000 0000
 - arbitražu između 3 sklopa koji mogu zahtijevati pristup memoriji (*Programmable Bus Arbiter*, PBA)
 - procesor
 - DMA kontroler
 - Ethernet MAC kontroler
 - rad s Flash memorijom (*Embedded Flash Controller*)
 - brisanje, programiranje
 - čitanje podatka širine 8, 16 ili 32 bita
 - pisanje podatka širine 32 bita
 - zaključavanje blokova
⇒ neke blokove je moguće zaštititi od upisa, npr. one koji sadrže programski kod
 - sigurnosni bit (*Security Bit*)
⇒ sprečavanje čitanja podataka iz Flash memorije
 - detekciju pogrešnog poravnavanja riječi u memoriji (*Misalignment Detector*)
 - pamćenje statusa pogreške koja je uzrokovala prekid rada procesora (*Abort Status Registers*)
 - upravljanje nekim operacijama koje se koriste kod uhodavanja programa
 - podržava samo *little endian*

- sabirnice
 - ASB (*Advanced System Bus*)
 - APB (*Advanced Peripheral Bus*)
⇒ ovaj mikrokontroler koristi AMBA sabirnice
- AMBA - *Advanced Microcontroller Bus Architecture*
 - neslužbeni ali široko prihvaćeni standard sabirnica za SOC koji koriste ARM procesore
 - razvijene u kompaniji ARM, 1996
 - postoji više generacija ovih sabirnica
 - ASB, APB (najstarije)
 - AHB - AMBA *High-performance Bus*, (AMBA 2.0)
 - AXI - *Advanced eXtensible Interface*,
 - ATB - *Advanced Trace Bus*,(AMBA 3.0)
 - vidi predmet *Arhitektura računala*

3.3.3 Memorijska mapa

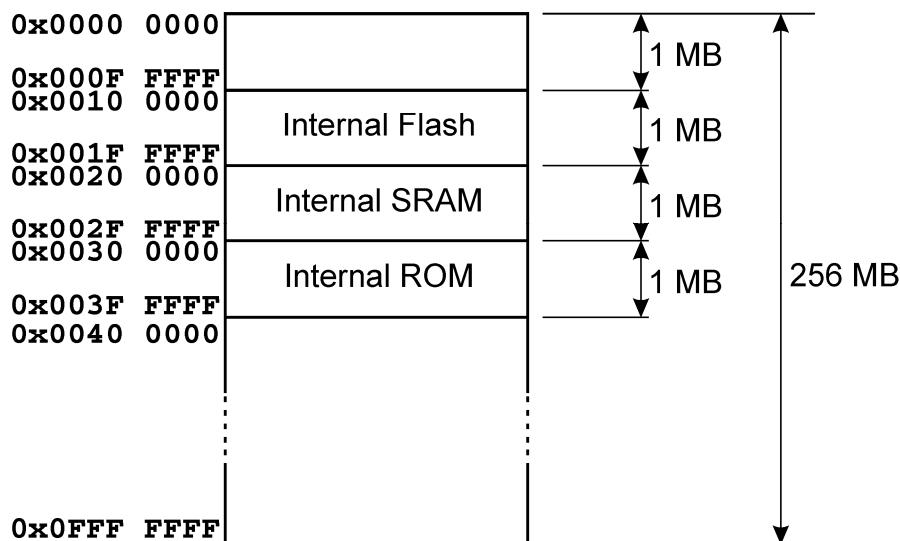
- adresna sabirnica
 - širina 32 bita
 - ⇒ adrese 0x0000 0000 do 0xFFFF FFFF
 - ⇒ ukupno 4 294 967 296 lokacija (4GBytes)
 - iskorišten je početak i kraj memorijskog prostora
- adresni memorijski prostor



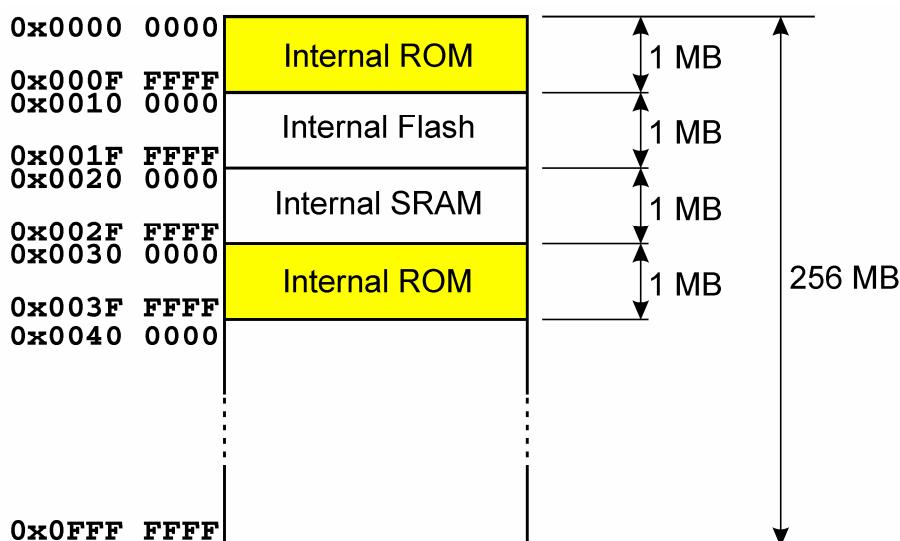
- uočiti
 - periferije se vide u memorijskom prostoru
- pristup neiskorištenom (nedefiniranom, *Undefined*) području uzrokuje prekid rada procesora (*Abort*)

3.3.3.1 Unutrašnja memorija

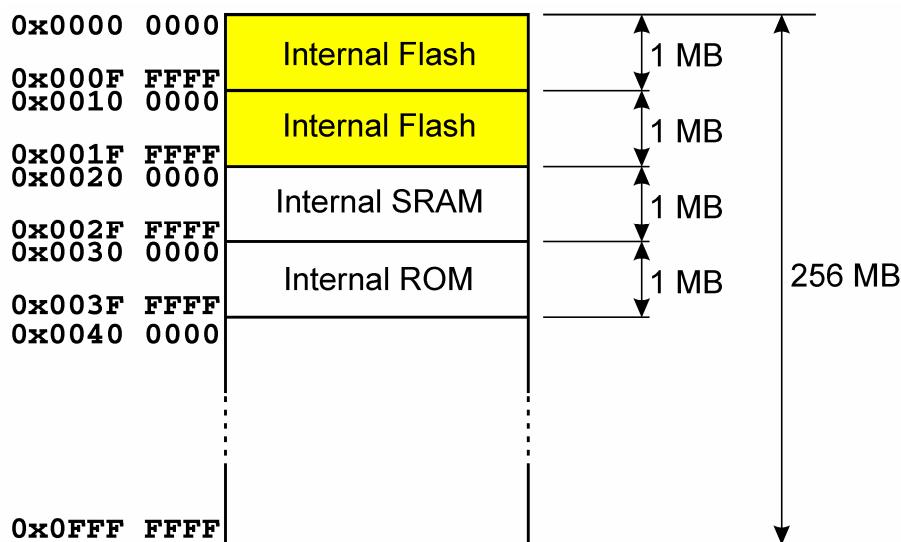
- *Internal Memory*
- SRAM, Flash i ROM se uvijek vide na fiksnim adresama



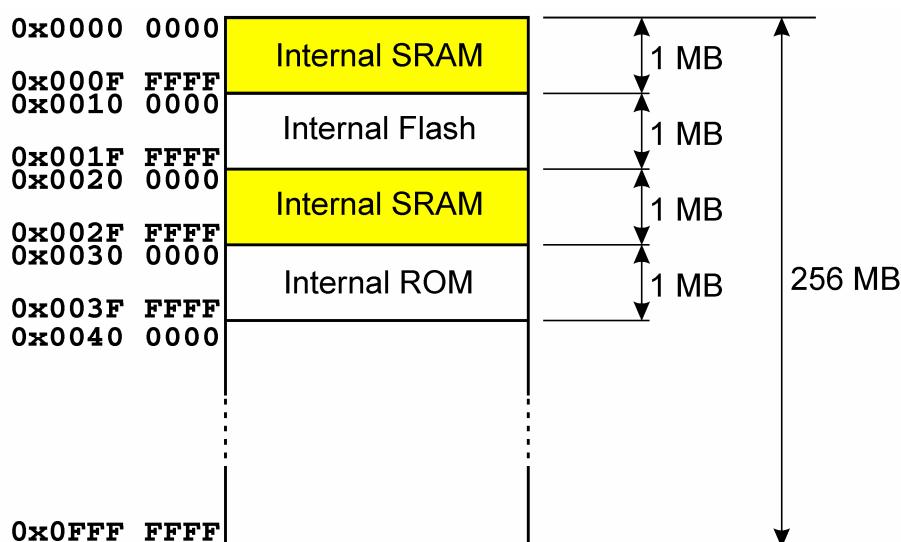
- prvi MB sadrži mapiranu jednu od memorija, što ovisi o
 - stanju zastavice GPNVM (*General Purpose Non-Volatile Memory*)
⇒ nalazi se u registru MC_FSR
 - komandi Remap koja je dana memorijskom kontroleru
- koja memorija se vidi počevši od lokacije 0 ovisi o tome je li
 - GPNVM, bit 2 = 0, (ovo je slučaj nakon brisanja Flash-a)



- GPNVM, bit 2 = 1



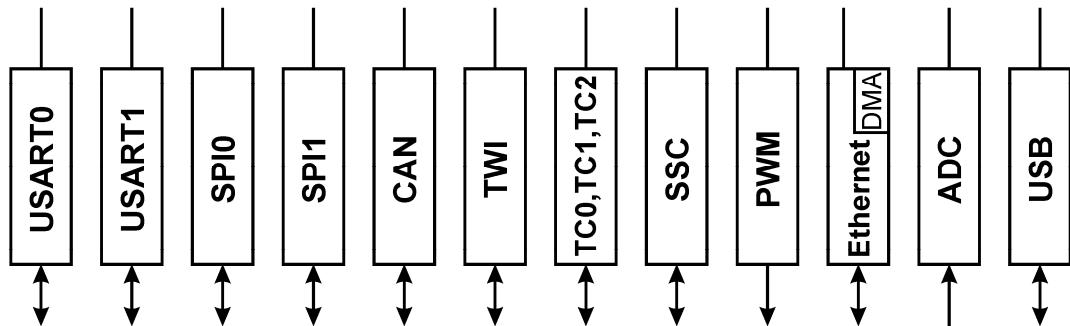
- memorijski kontroler izvršio komandu Remap



- zastavica GPNVM je "non-volatile"
 - ⇒ nakon reset-a u prvom MB može biti ili ROM ili FLASH
- Remap se izvodi programski
 - ⇒ nakon reseta u prvom MB uvijek se vidi ROM ili FLASH
- uočiti
 - μC nakon reseta počinje izvršavanje instrukcija od 0x0000 0000
 - ⇒ boot memorija može biti ROM, Flash ili SRAM
 - nakon brisanja Flash-a (ERASE) boot memorija je ROM

3.3.3.2 Periferijski memorijski prostor

- periferije



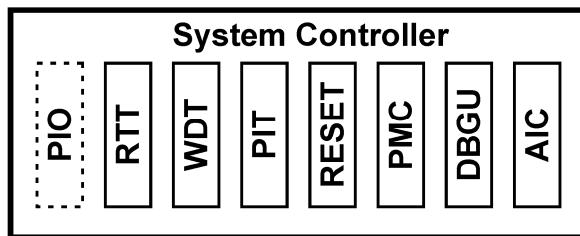
- registri koji odgovaraju pojedinim periferijama vide se u vanjskom memorijskom prostoru

| | | | |
|---------|------|---------------|-------|
| 0xF000 | 0000 | | |
| 0xFFFF9 | FFFF | | |
| 0xFFFFA | 0000 | TC0, TC1, TC2 | 16 KB |
| 0xFFFFA | 3FFF | | |
| 0xFFFFA | 4000 | | |
| 0xFFFFA | FFFF | | |
| 0xFFFFB | 0000 | UDP | 16 KB |
| 0xFFFFB | 3FFF | | |
| 0xFFFFB | 4000 | | |
| 0xFFFFB | 7FFF | | |
| 0xFFFFB | 8000 | TWI | 16 KB |
| 0xFFFFB | BFFF | | |
| 0xFFFFB | C000 | | |
| 0xFFFFB | FFFF | | |
| 0xFFFFC | 0000 | USART0 | 16 KB |
| 0xFFFFC | 3FFF | | |
| 0xFFFFC | 4000 | USART1 | 16 KB |
| 0xFFFFC | 7FFF | | |
| 0xFFFFC | 8000 | | |
| 0xFFFFC | BFFF | | |
| 0xFFFFC | C000 | PWMC | 16 KB |
| 0xFFFFC | FFFF | | |
| 0xFFFFD | 0000 | CAN | 16 KB |
| 0xFFFFD | 3FFF | | |
| 0xFFFFD | 4000 | SSC | 16 KB |
| 0xFFFFD | 7FFF | | |
| 0xFFFFD | 8000 | ADC | 16 KB |
| 0xFFFFD | BFFF | | |
| 0xFFFFD | C000 | EMAC | 16 KB |
| 0xFFFFD | FFFF | | |
| 0xFFFFE | 0000 | SPI0 | 16 KB |
| 0xFFFFE | 3FFF | | |
| 0xFFFFE | 4000 | SPI1 | 16 KB |
| 0xFFFFE | 7FFF | | |
| 0xFFFFE | 8000 | | |
| 0xFFFFF | EFFF | | |
| 0xFFFFF | F000 | | |
| 0xFFFFF | FFFF | SYSC | |

- 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 sklopolja (*SYSC, System Controller*)

3.3.3.3 Memorijski prostor upravljačkog sklopoljja

- upravljačko sklopolje

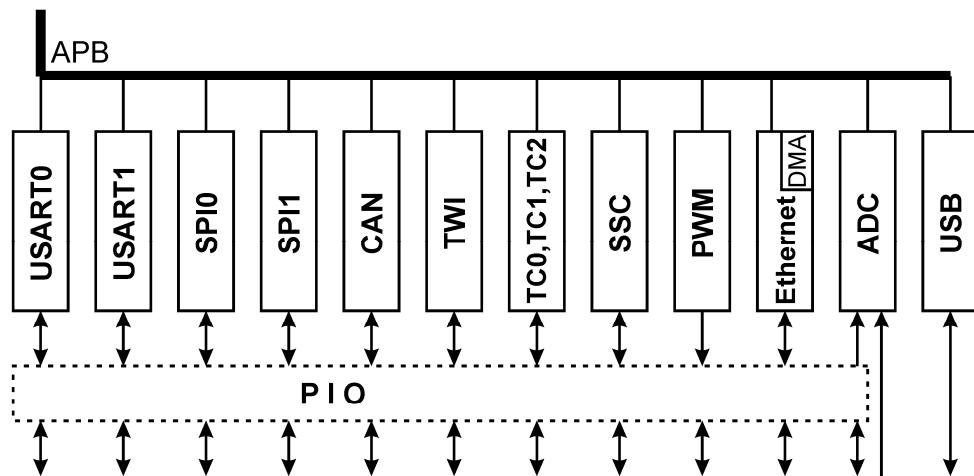


- registri koji odgovaraju pojedinim sklopovalima vide se u vanjskom memorijskom prostoru

| | | |
|-------------|-------------------|-------|
| 0xFFFF F000 | AIC | 512 B |
| 0xFFFF F1FF | DBGU | 512 B |
| 0xFFFF F200 | PIOA | 512 B |
| 0xFFFF F3FF | PIOB | 512 B |
| 0xFFFF F400 | | |
| 0xFFFF F5FF | | |
| 0xFFFF F600 | | |
| 0xFFFF F7FF | | |
| 0xFFFF F800 | | |
| 0xFFFF FBFF | | |
| 0xFFFF FC00 | PMC | 256 B |
| 0xFFFF FCFF | RSTC | 16 B |
| 0xFFFF FD00 | | |
| 0xFFFF FD0F | | |
| 0xFFFF FD10 | | |
| 0xFFFF FD1F | | |
| 0xFFFF FD20 | RTT | 16 B |
| 0xFFFF FD2F | | |
| 0xFFFF FD30 | PIT | 16 B |
| 0xFFFF FD3F | | |
| 0xFFFF FD40 | WDT | 16 B |
| 0xFFFF FD4F | | |
| 0xFFFF FD50 | | |
| 0xFFFF FD5F | | |
| 0xFFFF FD60 | VREG | 4 B |
| 0xFFFF FD6F | | |
| 0xFFFF FD70 | | |
| 0xFFFF FEFF | | |
| 0xFFFF FF00 | | |
| 0xFFFF FFFF | Memory Controller | 256 B |

- uočiti
 - ovdje se nalaze i registri koji pripadaju sklopolju za upravljanje memorijom

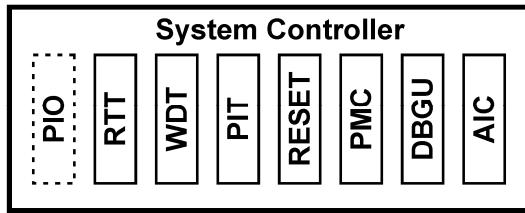
3.4 Periferno sklo povlje



- periferno sklo povlje spojeno je na sabirnicu APB
- komunicira s okolinom preko priključaka mikrokontrolera
- priključci mikrokontrolera su multipleksirani
⇒ priključci mogu imati od 2 do 3 funkcije
 - ulazno-izlazni priključak
 - jedna od dvije moguće periferne funkcije
 - primjer priključka s 2 funkcije
 - PA0 Port A - ulazni ili izlazni priključak
 - RXD0 - prijemna linija serijskog međusklopa USART0
 - primjer priključka s 3 funkcije
 - PB27 Port B - ulazni ili izlazni priključak
 - TIOA2 - priključak brojila 2 (TC2)
 - PWM0 - izlaz PWM modulatora
- periferno sklo povlje i rad s njim će biti detaljnije obrađeni u narednim poglavljima

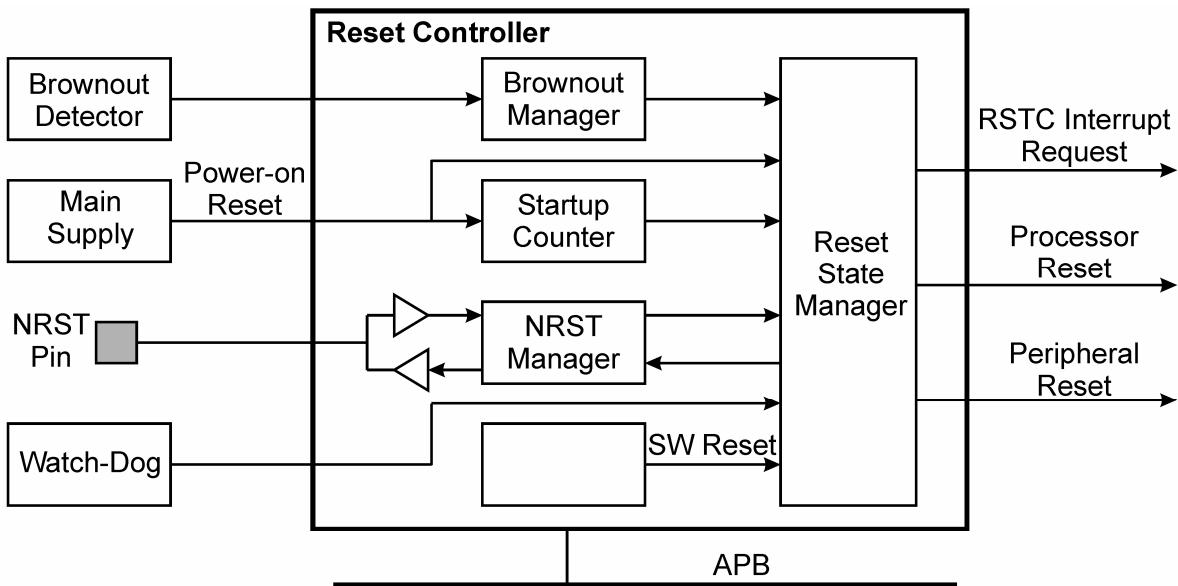
3.5 Upravljačko sklopovlje sustava

- System Controller



3.5.1 Sklopovlje za upravljanje resetom

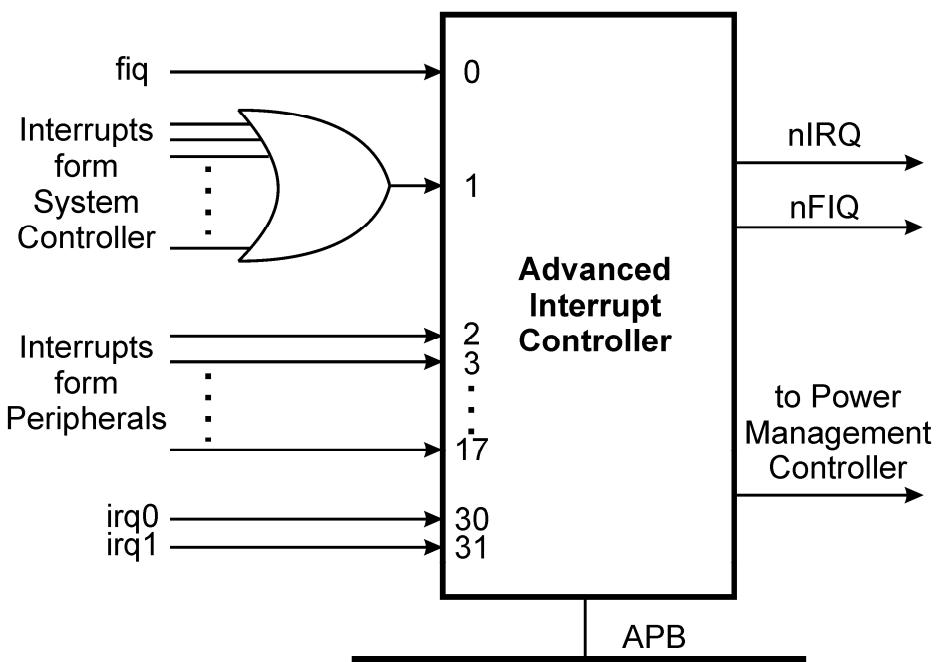
- Reset Controller, RSTC



- upravljački registri: RSTC_MR, RSTC_SR, RSTC_CR
(M=mode, S=status, C=control)
- 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

3.5.2 Sklopovlje 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, PWM, UDP, TC0, TC1, TC2, CAN, EMAC, ADC(=17)
 - 1 izvor prekida od upravljačkog sustava (*System Controller*) (=1)
 - ⇒ PIT, RTT, WDT, DBGU, PMC, RSTC, EFC

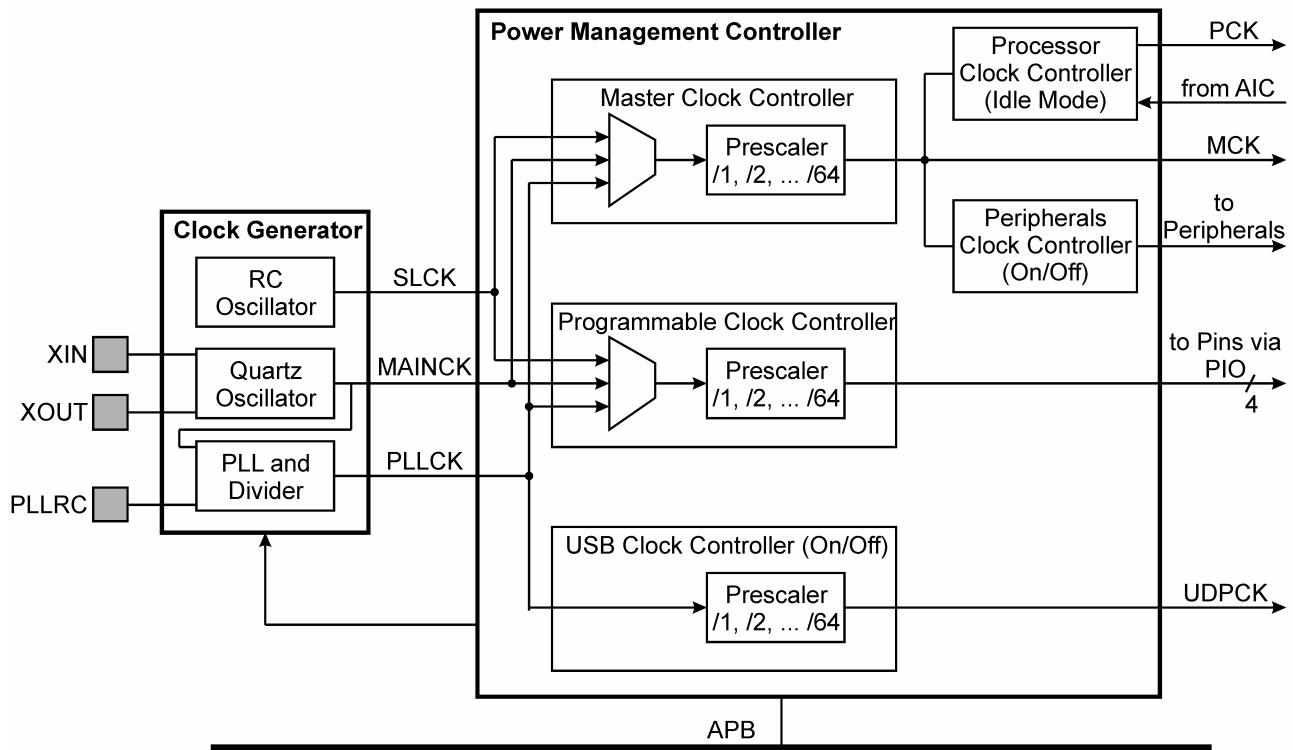


- uočiti
 - nisu iskorišteni sva 32 ulaza u AIC

- upravljački registri (ima ukupno 22 različita tipa registara) AIC sklopoljia omogućuju
 - maskiranje pojedinih prekida
 - definiranje prioriteta prekida (postoji 8 razina)
 - preusmjeravanje svakog izvora prekida na nIRQ i nFIQ
 - odabir tipa prekida
 - odabir prekida na brid ili na razinu
⇒ vidi predmet *Ugradbeni računalni sustavi*, 2. ciklus
 - pohranu prekidnih vektora za pojedine prekide
- AIC generira signal za prijelaz procesora iz *Stand-by* u regularan način rada
⇒ podsjetiti se predmeta *Ugradbeni računalni sustavi*, 2. ciklus
⇒ Načini rada u režimu smanjene potrošnje
- registri za konfiguriranje AIC i registri koji sadrže prekidne vektore
⇒ vide se u vanjskom memorijskom prostoru (APB sabirnica)

3.5.3 Sinteza i upravljanje taktom

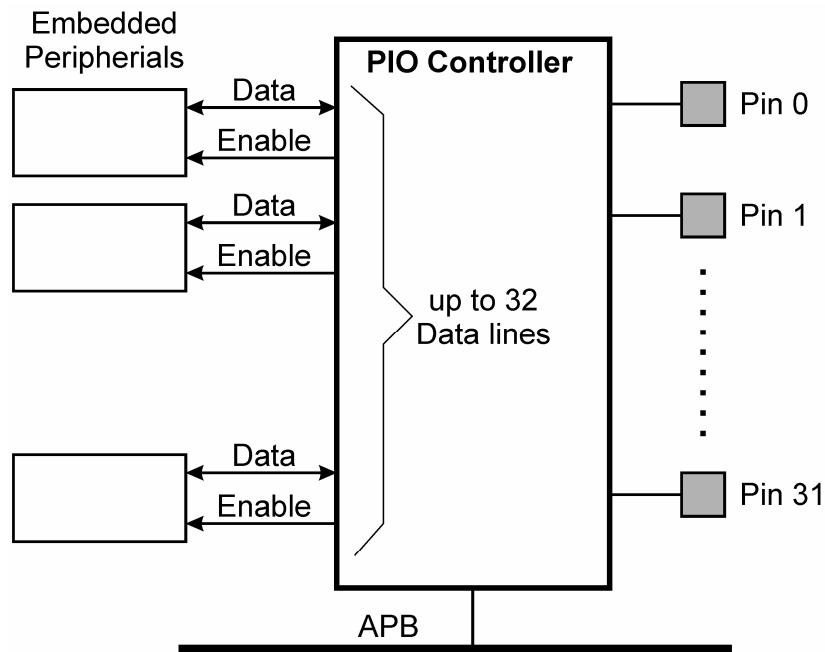
- blokovska shema sustava za sintezu i razvođenje takta



- mikrokontroler ima 3 izvora takta (*Clock Generator*)
 - SLCK - spori takt (*Slow Clock*) \Rightarrow frekvencija $32\text{kHz} \pm 10\text{kHz}$
 - 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 sklopoljje
 - optimira potrošnju energije
 \Rightarrow uključuje i isključuje takt periferijama i procesoru
- sklopoljje se konfigurira pomoću 16 upravljačkih registara
- uočiti
 - PMC zaustavlja takt kod *Idle* načina rada procesora
 - AIC daje signal za prijelaz u normalan način rada
 \Rightarrow posljedica zahtjeva za prekid

3.5.4 Kontroler ulazno-izlaznog sklopolja

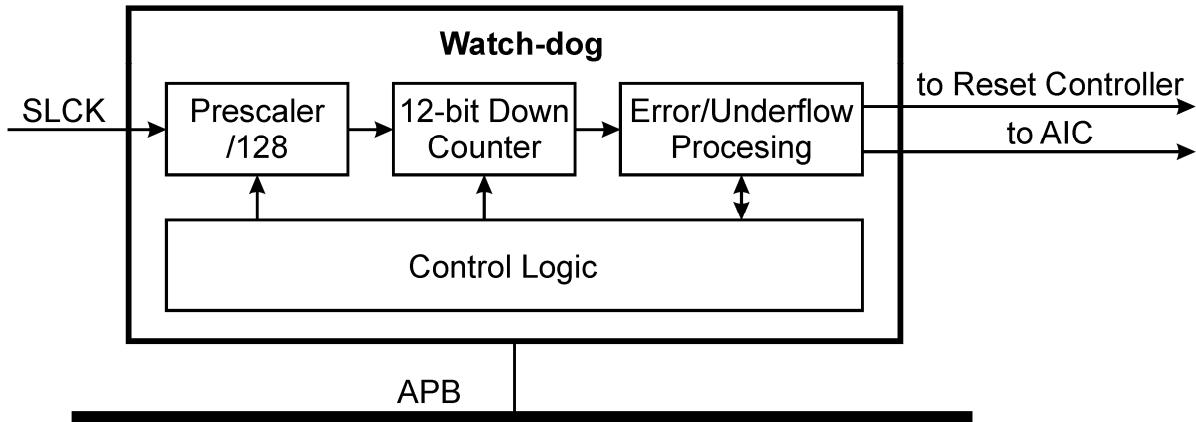
- *Parallel Input / Output Controller, PIO*



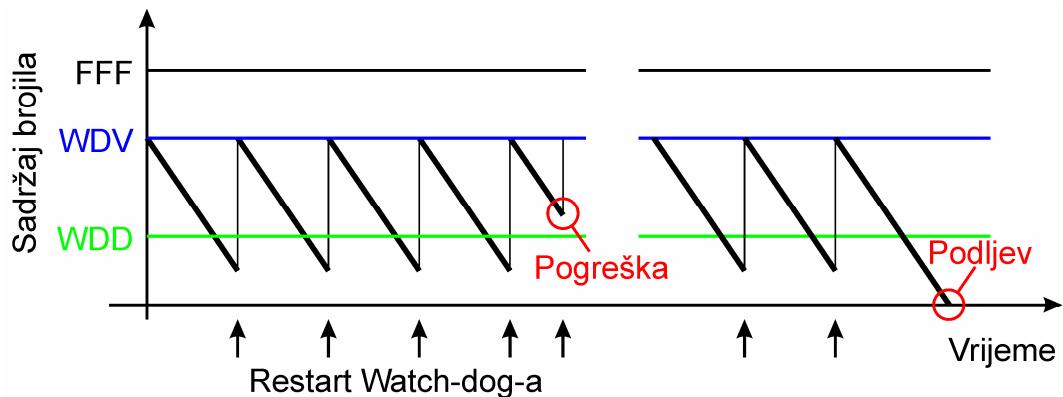
- mikrokontroleri familije SAM7 sadrže 2 kontrolera
⇒PIOA i PIOB
- svaki kontroler može upravljati s do 32 U/I linije
- upravljački i statusni registri (ima ih ukupno 29)
 - omogućuju ili onemogućuju I/O ili sekundarne funkcije
 - konfiguriraju parametre priključaka (pritezni otpornici, glitch filtri i sl.)
 - upravljaju zahtjevima za prekid koje daju priključci
 - itd.
- paziti
 - u PIO kontroleru registri za konfiguraciju imaju 32 bita, ali
 - u mikrokontrolerima SAM7 familije **postoji samo 31 linija na integriranom sklopu** (linije idu od 0:30)
 - ovakav PIO kontroler koristi se u mnogim mikrokontrolerima (ne samo u SAM7)
 - ⇒ u njima (neki put) postoji i linija 32
- uočiti
 - na ulazno-izlazne priključke preko PIO upravljačkog sklopa dolaze
 - signali iz perifernog sklopolja
 - neki signali koje daje System controller
 - ⇒ npr. iz sklopolja koje upravlja taktom

3.5.5 Watch-dog

- sklop za detekciju pogrešnog rada (Watch-dog, WDT)
 - vidi predmet *Ugradbeni računalni sustavi, 2. ciklus*
- pojednostavljena blokovska shema



- principa rada



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

- upravljački registri *Watch-dog* sklopa

WDT_MR (Watch-dog Timer Mode Register)

| | | | | | | | | |
|----|------------------|-----------------|------------|--------------|----------------|----------------|---------------|------------|
| 31 | 30 | 29 | 27-16 | 15 | 14 | 13 | 12 | 11-0 |
| - | WDIDLEHLT | WDDBGHLT | WDD | WDDIS | WDRPROC | WDRSTEN | WDFIEN | WDV |

- u ovaj register nakon reseta procesora može se upisati samo jednom !!!
⇒ time se onemogućuje naknadno neželjeno reprogramiranje

WDT_CR (Watch-dog Timer Control Register)

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

- upis WDRSTT=1 restara *Watch-dog* brojilo i preskaler
- upis se izvodi sa zaporkom !!!
⇒ da bi upis u WDRSTT uspio, mora biti istovremeno upisano KEY=0xA5

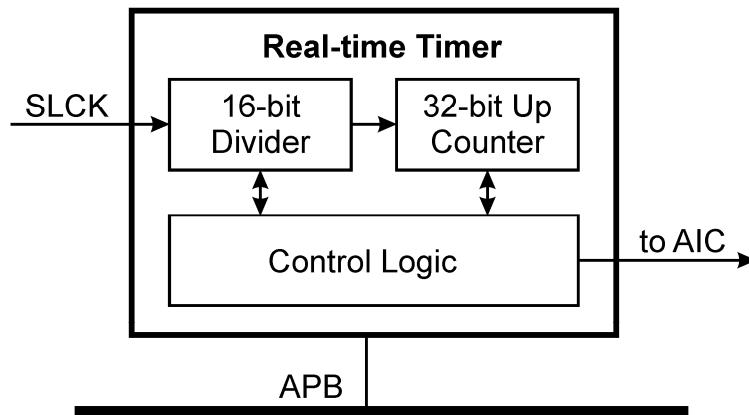
WDT_SR (Watch-dog Timer Status Register)

| | | |
|------|--------------|--------------|
| 31-2 | 1 | 0 |
| - | WDERR | WDUNF |

- Watch-dog* pogreška postavlja WDERR=1
- Watch-dog* podljev postavlja WDUNF=1
- Watch-dog* se onemogućava postavljanjem
WDDDIS=1
- nakon reseta
 - Watch-dog* je omogućen
 - WDV=0xFFFF
⇒ najveći mogući period
- uočiti
 - Watch-dog* koristi spori takt iz RC oscilatora
⇒ najveći period (uz WDV=0xFFFF) iznosi oko 16 s

3.5.6 Real-time Timer

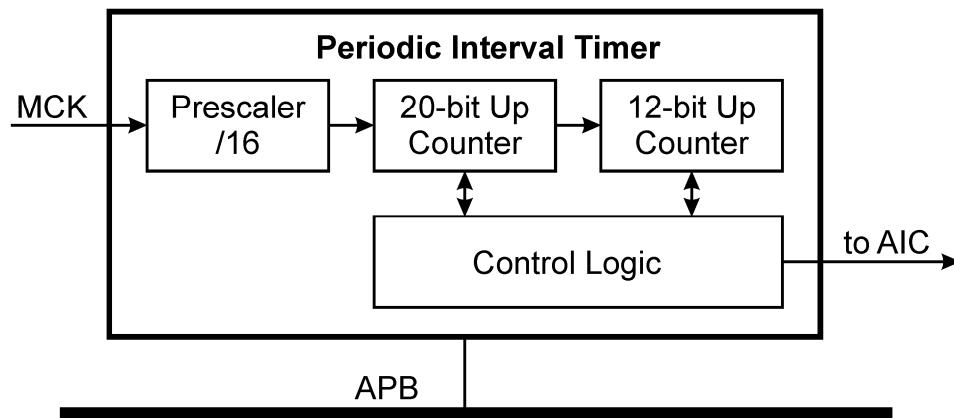
- *Real-time Timer, RTT*



- upravljački registri: RTT_MR, RTT_AR, RTT_VR, RTT_SR
- RTT mjeri ukupno proteklo vrijeme
- 16 bitno djelilo (*16-bit Divider*)
 - faktor dijeljenja se može programirati
 - nakon reseta μ C, konfiguirano je tako daje impulse perioda 1s
- glavno brojilo (*32-bit Up Counter*)
 - broji do 2^{32} ⇒ ako broji sekunde, broji **preko 136 godina**
- upravljačko sklopolje (*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
- uočiti
 - SLCK daje RC oscilator
⇒ frekvencija iznosi 32kHz ± 10kHz !!!

3.5.7 Periodic Interval Timer

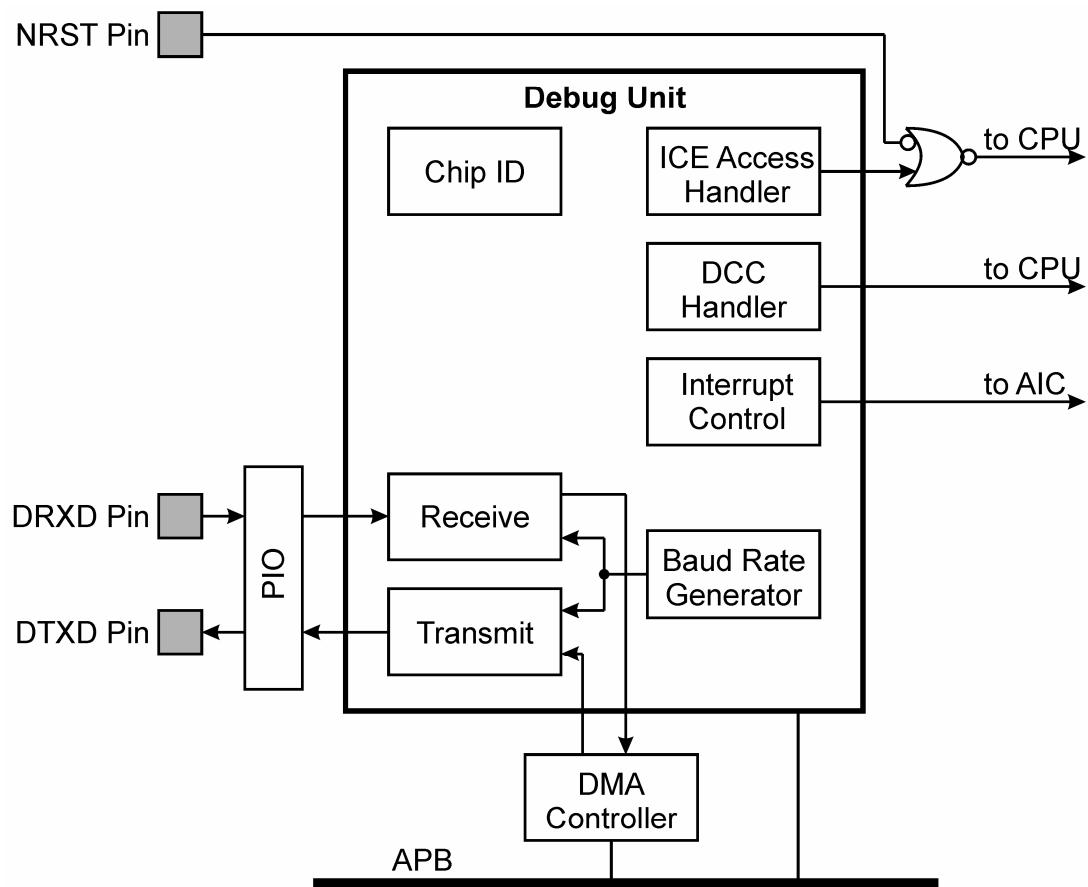
- *Periodic Interval Timer, PIT*



- upravljački registri: PIT_MR, PIT_SR, PIT_PIVR, PIT_PIIR
- 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 sklopolje (*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

3.5.8 Sklopovlje za uhodavanje mikrokontrolera

- *Debug Unit, DBGU*



- sklopovlje za uhodavanje omogućava
 - komunikaciju s osobnim računalom preko RS232 (DRXD i DTXD)
 - komunikaciju s ICE (*In-Circuit Emulator*) preko DCC (*Debug Communication Channel*)
 - identifikaciju integriranog sklopa i njegove revizije
- više o spomenutim sklopovima bit će govora u narednim poglavljima
- tipična primjena DBGU sklopovlja
 - u ROM-u se nalazi program SAM-BA (*SAM Boot Assistant*)
 - nakon reseta mikrokontrolera, SAM-BA
 - inicijalizira DBGU
 - inicijalizira memorijski kontroler (*Memory Controller*)
 - preko DTXD i DRXD uspostavlja komunikaciju s osobnim računalom
 - učitava s osobnog računala program za μ C
 - sprema program u Flash memoriju
 - određuje "boot" memoriju (postavlja GPNVM bit)

3.6 Pitanja za provjeru znanja

1. Nacrtati osnovnu blokovsku shemu mikrokontrolera familije AT97SAM7X i ukratko opisati funkciju pojedinih sklopova.
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?
3. Opisati registre opće namjene procesora ARM7TDMI. Navesti "bankirane" registre u pojedinim načinima rada. Koja je uloga registara R13, R14 i R15?
4. Opisati statusne registre procesora ARM7TDMI.
5. Objasniti razliku između ARM i Thumb skupova instrukcija. Čime je određeno koji skup instrukcija se trenutno koristi?
6. Opisati iznimke kod procesora ARM7TDMI.
7. Nacrtati pojednostavljenu blokovsku shemu memorijskog kontrolera mikrokontrolera familije AT91SAM7. Ukratko opisati njegovu funkciju. Navesti potencijalne upravljače u sustavu.
8. Nacrtati memorijsku mapu mikrokontrolera AT91SAM7X i opisati mapiranje unutrašnjih memorija.
9. Opisati način adresiranja periferija. Gdje se nalazi memorijski prostor u kojem se vide periferije i upravljačko sklopolje?
10. Opisati princip spajanja periferija s vanjskim priključcima mikrokontrolera.
11. Nacrtati pojednostavljenu blokovsku shemu sklopolja za upravljanjem resetom i navesti funkciju pojedinih dijelova.
12. Skicirati i opisati pojednostavljenu blokovsku shemu sklopolja za upravljanje prekidima.
13. Skicirati i opisati pojednostavljenu blokovsku shemu sklopolja za upravljanje taktom.
14. Skicirati pojednostavljenu blokovsku shemu *Watch-dog* sklopa. Opisati način njegovog rada. Objasniti razliku između podjeva i pogreške.
15. Skicirati pojednostavljenu blokovsku shemu sklopa *Real-time Timer* i opisati način njegovog rada.
16. Skicirati pojednostavljenu blokovsku shemu sklopa *Periodic Interval Timer* i opisati način njegovog rada.

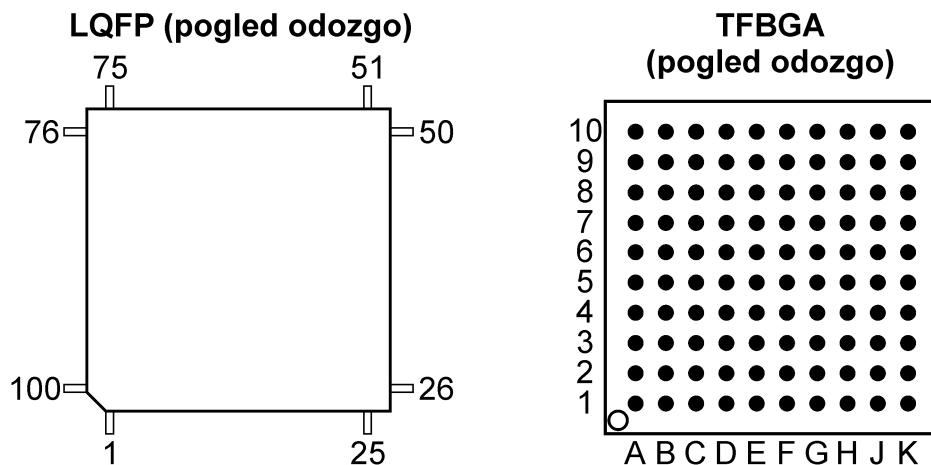
3.7 Zadaci za samostalni rad

1. Potrebno je na Internetu potražiti dokument *AT91 ARM Thumb-based Microcontrollers, Datasheets, Atmel 2007* Pregledati strukturu dokumenta s ciljem da se ovlada njegovim sadržajem, tj. da se u njemu brzo može pronaći tražene podatke. Proučiti dijelove koji su obrađivani u okviru predavanja.

4 Električko spajanje mikrokontrolera

4.1 Vanjski priključci mikrokontrolera

- kućište mikrokontrolera
 - LQFP (*Low-Profile Quad Flat Package*) sa 100 izvoda
 - TFBGA (*Thin and Fine-Pitch Ball Grid Array*) sa 100 izvoda

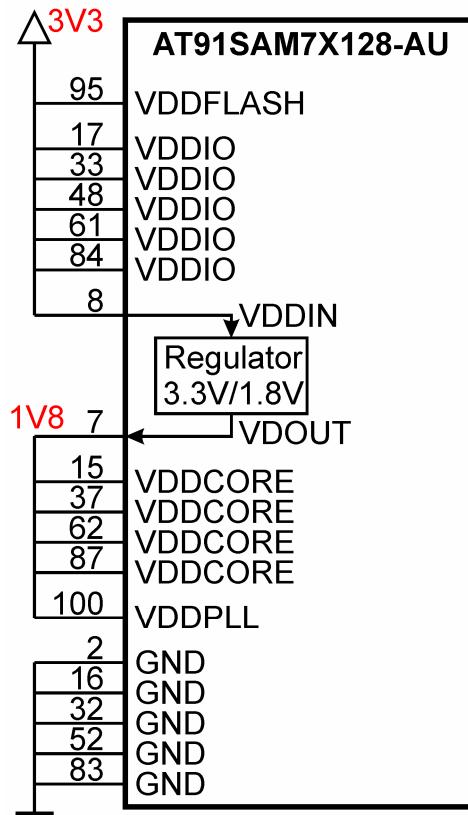


- sadrži priključke za
 - napajanje i masu
 - vanjske komponente izvora takta
 - reset
 - brisanje Flash memorije
 - JTAG sučelje
 - ulaze i izlaze odnosno spajanje periferija

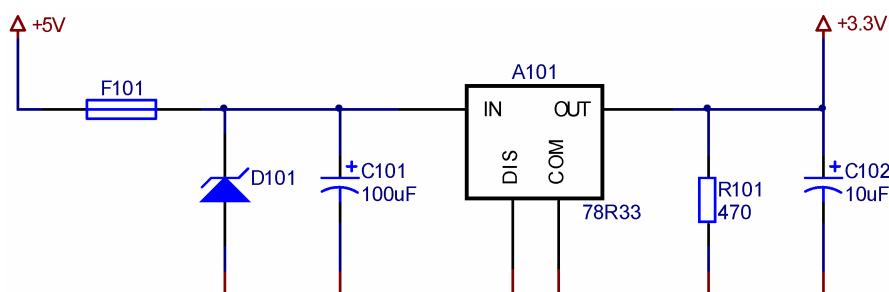
4.2 Napajanje mikrokontrolera

- sklopolje mikrokontrolera koristi dva napajanja
 - 3.3V
 - VDDIO - ulazno izlazno sklopolje
 - VDDFLASH - Flash memorija
 - 1.8V
 - VDDCORE - jezgra i ostalo sklopolje
 - VDDPLL - sklopolje za sintezu raka
- unutar mikrokontrolera postoji regulator 3.3V na 1.8V

- spoj napajanja (primjer LQFP podnožja)

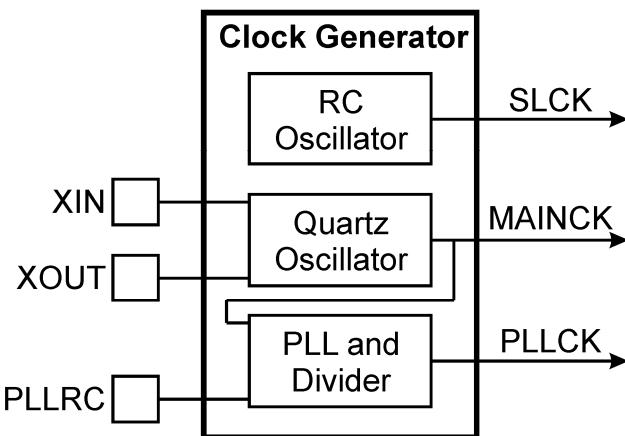


- na svaki priključak potrebno je još staviti kondenzator oznosa 100nF , malog vlastitog induktiviteta
 - ⇒ vidi predmet *Konstrukcija elektroničkih uređaja*
 - ⇒ vidi predmet *Ugradbeni računalni sustavi*, 1. nastavna cijelina
- često se mikrokontroler napaja iz izvora napona 5V
 - vidi URS
 - primjer regulatora



- osigurač i Zener dioda služe za zaštitu
- ulazni napon može biti od 4.5V do 35V
 - ⇒ npr. iz USB priključka

4.3 Izvor takta

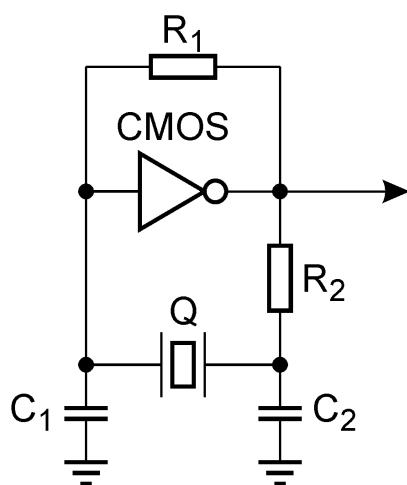


4.3.1 RC oscilator

- frekvencija $32\text{kHz} \pm 10\text{kHz}$
- sve komponente nalaze se na integriranom sklopu

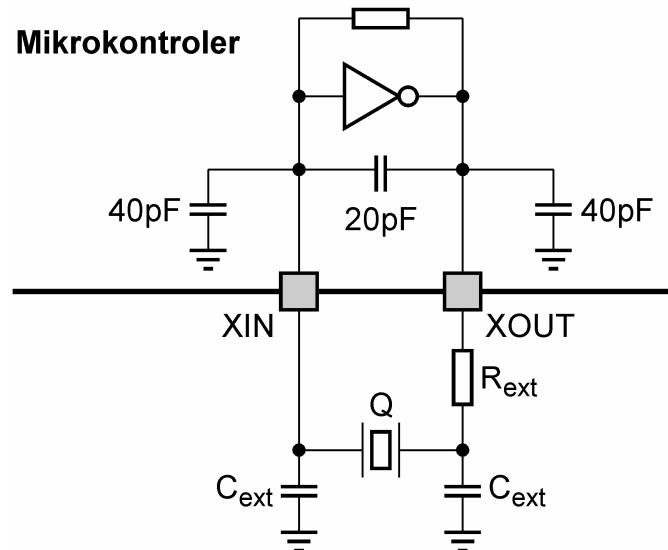
4.3.2 Glavni oscilator

- za stabilizaciju frekvencije koristi kristal kvartza
- koristi paralelnu rezonanciju kristala
- oscilator koji koristi paralelnu rezonanciju kristala
⇒ vidi predmet *Ugradbeni računalni sustavi*, 2. nastavna cijelina
 - osnovna shema spajanja



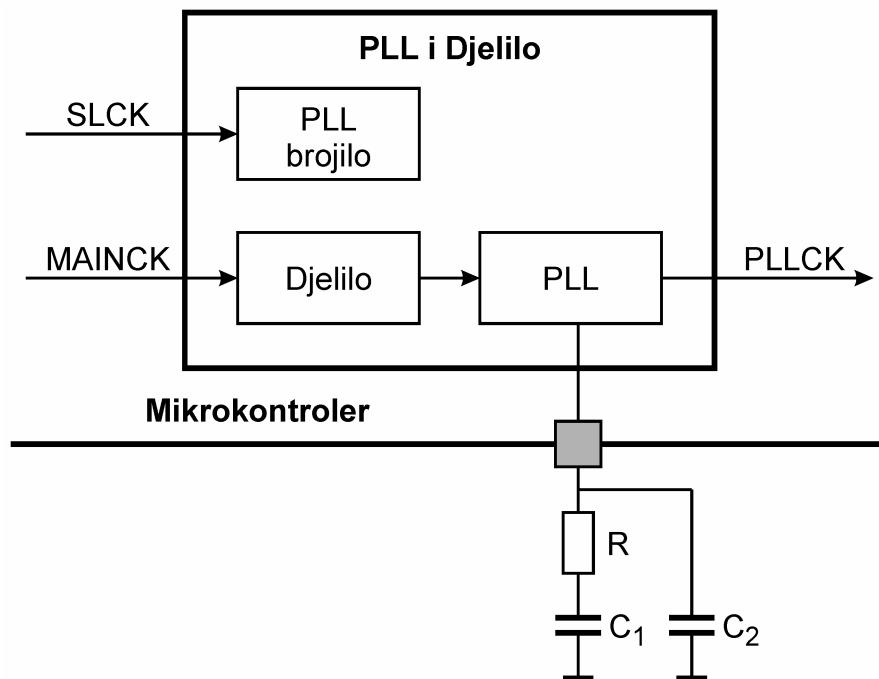
- R_1 osigurava statičku radnu točku digitalnog sklopa
- Q , C_1 i C_2 osiguravaju povratnu vezu na f_Q
- R_2 se ponekad stavlja zbog bolje temperaturne karakteristike oscilatora

- izvedba oscilatora kod ovog mikrokontrolera



- vrijednosti vanjskih komponenata
 $3\text{MHz} \leq f_Q \leq 20\text{MHz}$ (osnovni harmonik)
 $C_{ext} \leq 10\text{pF}$
 $R_{ext} = 0$

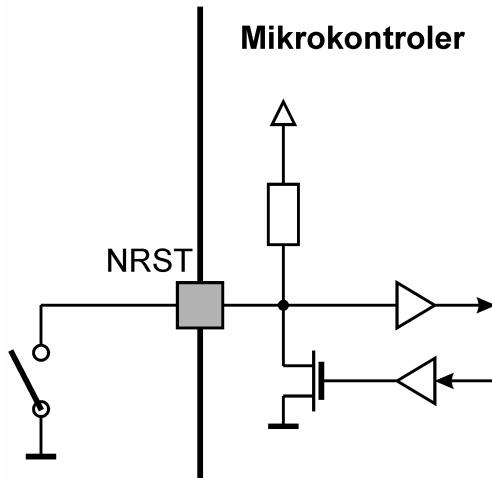
4.3.3 PLL



- sinteza frekvencije
 - djelilo \Rightarrow dijeli s faktorom DIV koji je između 1 i 255
 - PLL \Rightarrow množi s faktorom MUL+1, MUL je između 1 i 2047
 - frekvencija signala PLLCK
$$f_{\text{PLLCK}} = f_{\text{MAINCK}} \frac{\text{MUL} + 1}{\text{DIV}}$$
- PLL brojilo
 - broji vremenski interval potreban da se PLL utitra
 \Rightarrow postavlja zastavicu LOCK kad je PLL utiran
- izračun vrijednosti R, C₁ i C₂ prelazi okvire ovog predmeta
 - proizvođač daje preporuke za izračun

4.4 Reset

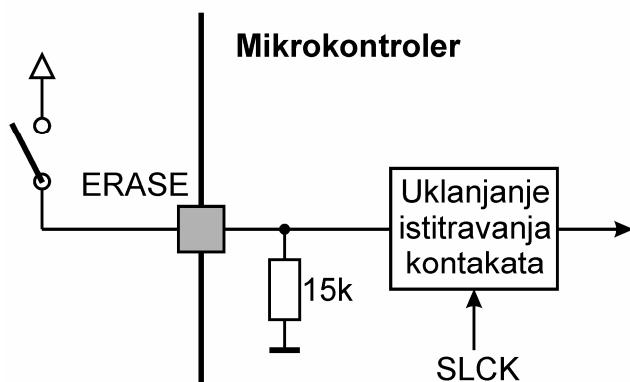
- izvedba reset priključka



- mikrokontroler brine o širini impulsa za reset
⇒ izvana je potrebno samo dodati tipku

4.5 ERASE priključak

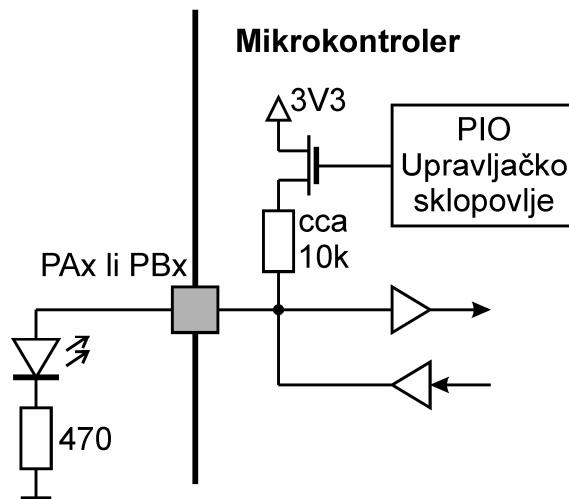
- pozitivan impuls na priključku ERASE briše sadržaj Flash memorije
- izvedba priključka ERASE



- postoji sklop za istitravanje kontakata
⇒ tipka mora biti pritisnuta barem 220ms

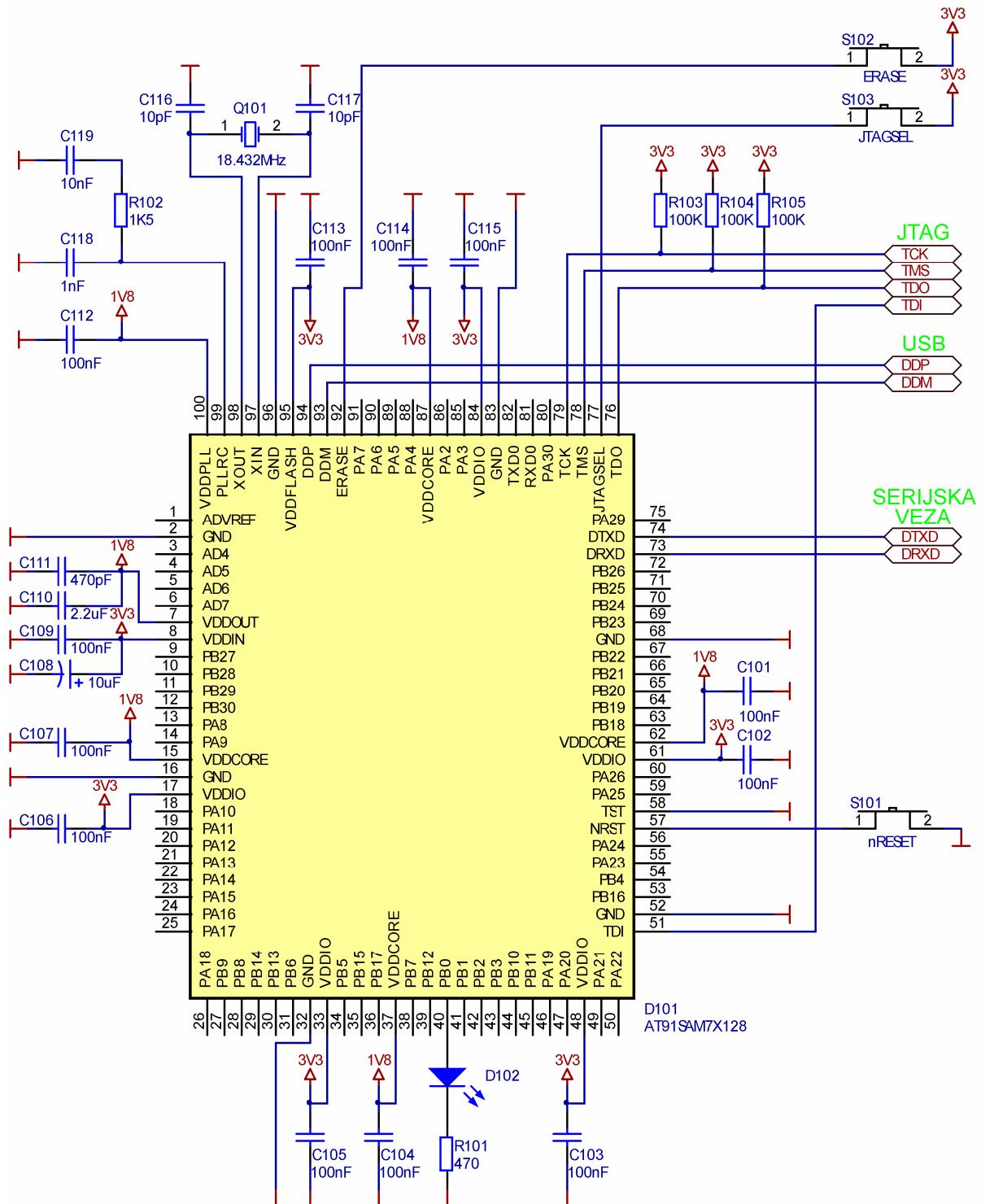
4.6 Indikator

- u računalnom sustavu uvijek je potrebno predvidjeti neki indikator koji se može upravljati programski
⇒ to je korisno kod uhodavanja sustava
- kod mikrokontrolera
⇒ staviti svijetleću diodu (LED) na jedan izlazni priključak



- uočiti
 - pritezni otpornik je nakon reseta uključen
⇒ dioda će svijetliti nakon reseta (slabim intenzitetom)
- paziti kod priključivanja komponenata na U/I linije
 - priključci mogu dati konačnu struju
 - PA0-PA3 ⇒ 16mA
 - PB27-PB30 ⇒ 2 mA
 - svi ostali priključci ⇒ 8 mA
 - ukupna struja (suma struja) svih priključaka ⇒ 200mA
 - priključci su TTL kompatiblini
 - mogu se spajati komponente koje daju napon od 0V do 5V
 - pritezni otpornik treba isključiti da ne troši struju

4.7 Primjer sheme spajanja



- ako se ne koristi AD, treba spojiti na masu ADVREF, AD4, AD5, AD6 and AD7 da se smanji potrošnja

4.8 Pitanja za provjeru znanja

1. Opisati kako je izvedeno napajanje mikrokotrolera familije AT91SAM7X. Nacrtati izvedbu serijskog regulatora koji daje glavno napajanje.
2. Opisati izvore takta mikrokontrolera AT91SAM7X. Koje vanjsko sklopovlje je potrebno spojiti na izlazne mikrokontrolera za ispravan rad ovih izvora?
3. Opisati priključke za reset i brisanje memorije i nacrtati spoj vanjskih komponenata koje je potrebno ugraditi. Opisati spoj svijetleće diode na priključak.

4.9 Zadaci za samostalni rad

1. Podsjetiti se principa na kojima rade oscilatori izvedeni pomoću digitalnih sklopova i to oscilatori s kristalom kvartza i RC oscilatori. (Poslužiti se udžbenikom M. Vučić, Upotreba mikrokontrolera u ugrađenim računalnim sustavima, FER, 2007.)

5 Alati za razvoj programske podrške

5.1 Razvojno okruženje

- razvoj programske podrške izvodi se pomoću skupa programskih alata
⇒ *Integrated Development Environment*, IDE
- jedna od poznatijih alata
 - Keil uVision IDE, proizvođač Keil Elektronik GmbH (dio ARM Ltd.)
- Keil uVision IDE sadrži
 - *Project Manager*
⇒ organizacija datoteka i vođenje cijelog projekta
 - *Text Editor*
⇒ pisanje programa
 - *C/C++ compiler*
⇒ prevodioč za jezik C/C++
 - *Macro Assembler*
⇒ prevodioč za asembler
 - *Linker/Loader*
⇒ program za povezivanje relokabilnog koda (linker) te njegovo smještanje u memoriju (loader)
 - *Debugger/Simulator*
⇒ skup alata za uhodavanje programske podrške i simulaciju rada mikrokontrolera
 - *HEX file generator*
⇒ program za kreiranje takozvanog Intel-HEX formata programa koji se razvija
 - *Library Manager*
⇒ služi za objedinjavanje korisnikovih .obj modula u korisnikove biblioteke
 - *C/C++ Libraries*
- uVision IDE podržava 2 načina rada
 - *Build Mode*
⇒ služi za razvoj programa podrške i generiranje izvršnog koda
 - *Debug Mode*
⇒ služi za uhodavanje programske podrške
 - omogućava izvršavanje razvijenog programa
 - u simulatoru
 - na stvarnom sklopolju

5.2 Pristup registrima koji upravljaju radom sklopovlja

5.2.1 Pristup registrima

- podsjetimo se
 - osnovne adrese (*Base Addresses*) pojedinih sklopova

| | | | |
|--------------|---------------|-------------------|-------|
| 0xF000 0000 | | | |
| 0xFFFF9 FFFF | | | |
| 0xFFFFA 0000 | | | |
| 0xFFFFA 3FFF | TC0, TC1, TC2 | | |
| 0xFFFFA 4000 | | | |
| 0xFFFFA FFFF | | | |
| 0xFFFFB 0000 | | | |
| 0xFFFFF 3FFF | UDP | | |
| 0xFFFFF 4000 | | | |
| 0xFFFFB 7FFF | | | |
| 0xFFFFB 8000 | TWI | | |
| 0xFFFFB BFFF | | | |
| 0xFFFFB C000 | | | |
| 0xFFFFB FFFF | | | |
| 0xFFFFC 0000 | USART0 | | |
| 0xFFFFC 3FFF | | | |
| 0xFFFFC 4000 | USART1 | | |
| 0xFFFFC 7FFF | | | |
| 0xFFFFC 8000 | | | |
| 0xFFFFC BFFF | | | |
| 0xFFFFC C000 | PWMC | | |
| 0xFFFFC FFFF | | | |
| 0xFFFFD 0000 | CAN | | |
| 0xFFFFD 3FFF | | | |
| 0xFFFFD 4000 | SSC | | |
| 0xFFFFD 7FFF | | | |
| 0xFFFFD 8000 | ADC | | |
| 0xFFFFD BFFF | | | |
| 0xFFFFD C000 | EMAC | | |
| 0xFFFFD FFFF | | | |
| 0xFFFFE 0000 | SPI0 | | |
| 0xFFFFE 3FFF | | | |
| 0xFFFFE 4000 | SPI1 | | |
| 0xFFFFE 7FFF | | | |
| 0xFFFFE 8000 | | | |
| 0xFFFFF EFFF | | | |
| 0xFFFFF F000 | SYSC | | |
| 0xFFFFF FFFF | | | |
| | | AIC | 512 B |
| | | DBGU | 512 B |
| | | PIOA | 512 B |
| | | PIOB | 512 B |
| | | PMC | 256 B |
| | | RSTC | 16 B |
| | | RTT | 16 B |
| | | PIT | 16 B |
| | | WDT | 16 B |
| | | VREG | 4 B |
| | | Memory Controller | 256 B |

⇒ svi registri koji upravljaju radom sklopovlja mapirani su u memorijski prostor

⇒ registrima se pristupa preko pokazivača

- ima nešto manje od 500 adresa na koje su mapirani registri
- u uVision okruženju postoje datoteke koje sadrže adrese registara
 - primjer: **at91sam7x128.h**
- datoteka at91sam7x128.h sadrži dva različita načina na kojima možemo pristupiti registrima mikrokontrolera
 - pristup izravno preko pokazivača
 - pristup preko pokazivača na strukturu
- uočiti
 - registrima se uvijek pristupa u na isti način, tj. preko pokazivača
⇒ spomenuti načini razlikuju se jedino po zapisu u C jeziku

5.2.2 Adresiranje 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 registr WDT_CR

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

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

- uočiti
 - registri su deklarirani kao volatile
 - potrebito je prevodiocu dati do znanja da su to memorijске lokacije čiji sadržaj ne mijenja samo program već i sklopovlje
 - primjer
 - 2 čitanja iz memorijске lokacije u koju se u međuvremenu ne upisuje, daje isti rezultat
⇒ prevodioc bi mogao u fazi optimizacije koda izbaciti jedno čitanje
 - 2 čitanja iz regista koji odgovara periferiji **ne** daje uvijek isti rezultat
⇒ tu treba koristiti deklaraciju tipa volatile
- napomena
 - definicije u datoteci at91sam7x128.h izvedene su nešto složenije nego što je pokazano u gornjem primjeru
 - tako je izvedeno zbog univerzalnosti i prenosivosti koda

5.2.3 Adresiranje 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;
```

5.2.4 Tipovi registara

- 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

Primjer

- register koji upravlja radom *Watch-dog* sklopa

WDT_MR

| | | | | | | | | |
|----|-----------|----------|-------|-------|---------|---------|--------|------|
| 31 | 30 | 29 | 27-16 | 15 | 14 | 13 | 12 | 11-0 |
| - | WDIDLEHLT | WDDBGHLT | WDD | WDDIS | WDRPROC | WDRSTEN | WDFIEN | WDV |

- značenje polja (vidi poglavlje 3.5.5 *Watch-dog*)
 - WDV - *Watch Dog Value*
 - WDFIEN - *Watch Dog Fault Interrupt Enable*
 - itd.
- da bi se olakšao pristup pojedinim poljima u registrima, u datoteci at91sam7x128.h definirane su maske za pojedina polja u registrima

Primjer

- definicija pojedinih polja u registrima za *Watch-dog Timer*

```
#define AT91C_WDTC_WDV    (0xFFFF<<0)// Watch Dog Value
#define AT91C_WDTC_WDFIEN (0x1<<12)
                                         // Watchdog Fault Interrupt Enable
// itd.
```

5.3 Pitanja za provjeru znanja

1. Što su integrirana okruženja za razvoj programske podrške i koje alate oni sadrže.
2. Dati primjer adresiranja registra koji upravljaju radom periferija i upravljačkog sklopovlja i to korištenjem pokazivača i korištenjem pokazivača na strukturu.
3. Navesti osnovne tipove registara koji upravljaju radom sklopovlja u mikrokontrolerima familije AT91SAM7X. Po čemu se ovi tipovi razlikuju, obzirom na njihovu funkciju?

5.4 Zadaci za samostalni rad

1. Potrebno je na Internetu pronaći, učitati i instalirati školsku inačicu programskog paketa za razvoj programske podrške za mikrokontrolere familije ARM. Paket se zove uVision, proizvodi ga Keil Elektronik koji dio kompanije ARM, a može se pronaći na web stranici www.keil.com. Ovaj program bit će korišten na laboratorijskim vježbama, a neki aspekti njegove primjene bit će obrađeni tokom slijedećih predavanja.
2. Nakon instalacije programskog paketa uVision, proučiti strukturu direktorija i uočiti što se u njima nalazi
3. Proučiti sadržaj datoteke *at91sam7x128.h*.

6 Puštanje u pogon

6.1 Elektromehanička provjera sklopa

- elektromehanička provjera sklopa nije provjera funkcionalnosti sklopa
- obuhvaća
 - skup provjera koje se provode **prije** priključivanja napajanja
 - skup provjera koje se provode **nakon** priključivanja napajanja

6.1.1 Provjere prije priključivanja napajanja

- prije prvog priključivanja napajanja potrebno je
 - vizualno provjeriti ispravnost mehaničke integracije sklopa
 - pregledati lemlja mesta
 - provjeriti čvrstoću montaže velikih komponentata
 - provjeriti kvalitetu montaže hladnjaka
 - provjeriti električku povezanost komponenata
 - provjeriti (ohmmetrom) sve važnije spojeve
 - ⇒ najvažnije je provjeriti spoj vodova za napajanje
 - provjeriti kratke spojeve između susjednih priključaka
 - provjeriti spoj energetskih komponenata
 - ⇒ npr. imaju li greškom spoj s hladnjakom
- **napajanje mora biti ispravno spojeno**
 - pogrešan spoj napajanja obično uzrokuje uništenje komponenata
 - pogrešan spoj signalnih linija u pravilu nije katastrofalan

6.1.2 Provjere nakon priključivanja napajanja

- neposredno nakon prvog priključivanja napajanja potrebno je
 - provjeriti temperaturu poluvodičkih komponenata
 - ⇒ do 60°C se prst može trajno držati na komponenti
- nakon prvog priključivanja napajanja potrebno je provjeriti
 - ispravnost napona napajanja na **svim** priključcima za napajanje na poluvodičkim sklopovima
 - rad oscilatora
 - paziti: na nekim μC oscilator ne radi dok se programski ne omogući
 - ispravnost upravljačkih signala i referentnih signala
 - ⇒ reset, referentne napone AD pretvarača i sl.
 - **itd.**

- uočiti
 - nepravilnost u spajanju (npr. kratak spoj) kod prve štampane pločice može kao posljedicu imati dugačko vrijeme uhodavanja
⇒ lociranje greške je otežano zbog (pogrešne) pretpostavke da je sklopolje ispravno spojeno
 - nepravilnost u spajanju kod serijske proizvodnje već uhodanog sklopa je lakše
⇒ poznato je da je sklop uhodan i da bi trebao raditi
 - izuzetak su greške koje se mogu uočiti tek u serijskoj proizvodnji
⇒ npr. greške izazvane tolerancijama komponenata

6.2 Funkcijsko uhodavanja sklopolja

- prije nego se pristupi razvoju "prave" programske podrške, potrebno je
 - uhodati razvojno okruženje
 - PC
 - skup alata za razvoj programske podrške
 - Keil uVision
 - veza PC i sustava s mikrokontrolerom
 - sustav za učitavanja programa u mikrokontroler
 - uhodati pojedine dijelove sustava s mikrokontrolerom
 - periferije
 - vanjsko sklopolje
- za uhodavanje periferija i vanjskog sklopolja koristi se programska podrška

6.3 Razvoj programske podrške za uhodavanje sklopolja

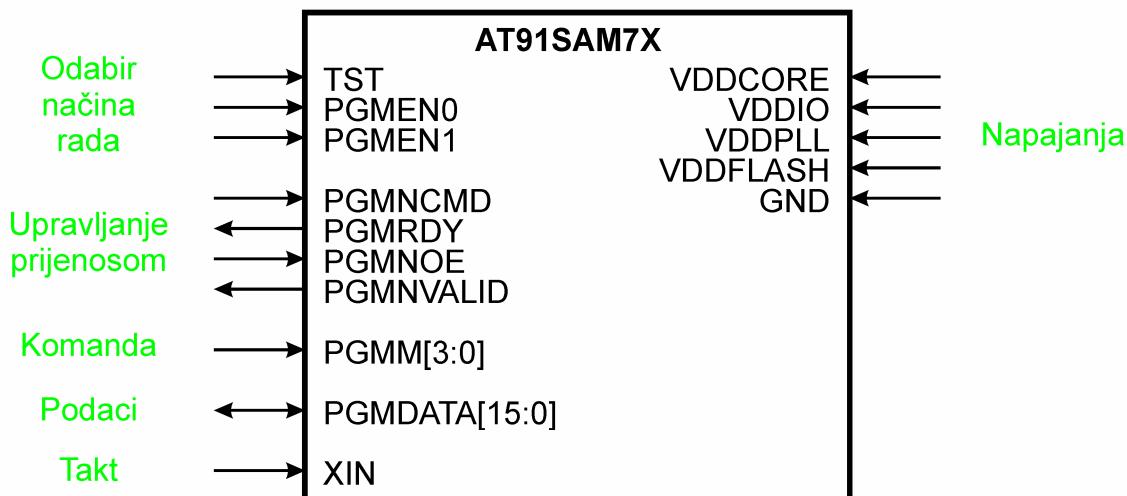
- programska podrška za uhodavanje
 - mora biti jednostavna
 - uhodava se jedna po jedna periferija ili vanjski sklop
 - ⇒ napiše se programska podrška za pojedini sklop
 - ⇒ uhoda se u simulatoru
 - ⇒ učita se u mikrokontroler
 - ⇒ provjeri se ispravnost rada
 - ⇒ po potrebi se promijeni program i/ili sklopolje
 - ⇒ tek onda se piše podrška za sljedeći sklop

6.4 Učitavanje programske podrške

- razvijena programska podrška se u Flash memoriju μ C može upisati na 3 načina
 - programiranjem pomoću posebnog programatora
 - korištenjem unutrašnjeg programa za inicijalizaciju (*Boot Loader*)
 - preko sučelja za uhodavanje (*In-Circuit Emulation, ICE*)

6.4.1 Učitavanje pomoću programatora

- postoje standardni programatori koji služe za programiranje raznih programabilnih sklopovima (μ C, EPROM, EEPROM)
 - ⇒ vidi predmet Ugradbeni računalni sustavi, 1. ciklus, lab. vježbe
- mikrokontroler NIJE zalemljen na štampanu pločicu već je priključen na programator
 - ⇒ μ C komunicira s programatorom preko FFPI sučelja (*Fast Flash Programming Interface*)
- FFPI sučelje podržava
 - paralelni prijenos podataka (*Parallel Programming*)
 - serijski prijenos podataka (*Serial Programming*)
- primjer - paralelno programiranje
 - koriste se samo slijedeći priključci

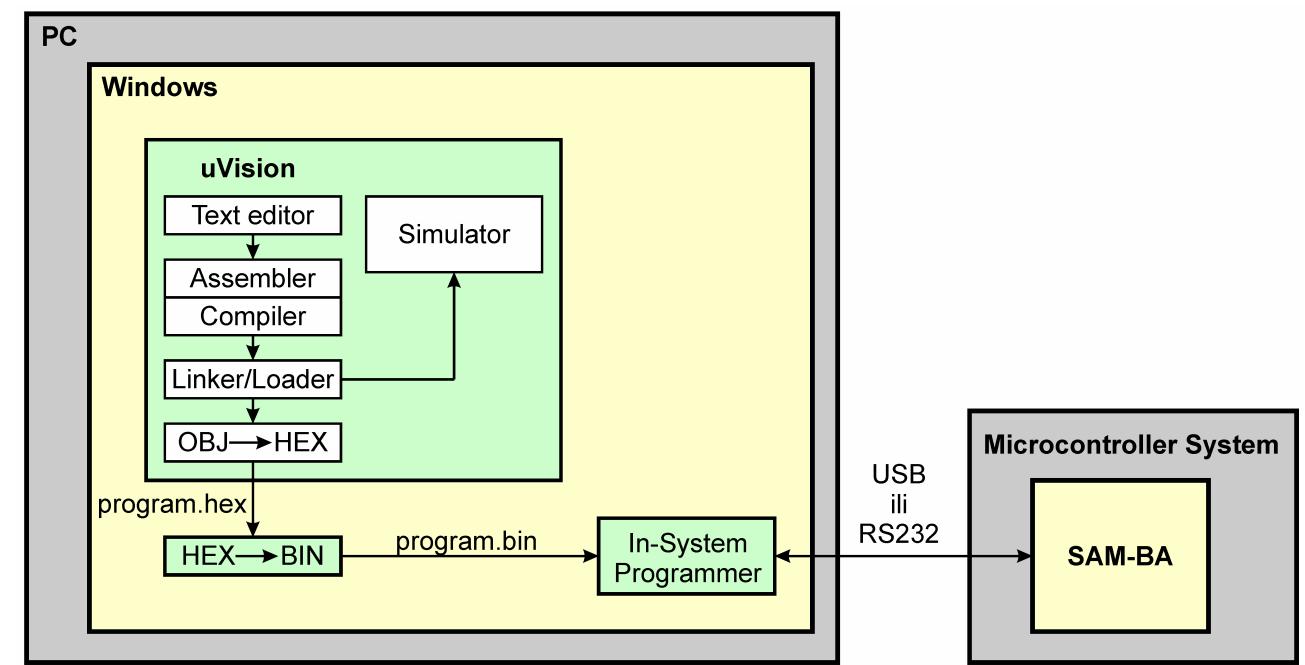


- ostali priključci moraju biti odspojeni
- signale i protokol za programiranje daje proizvođač
 - ⇒ za više informacija vidi *Datasheets* mikrokontrolera

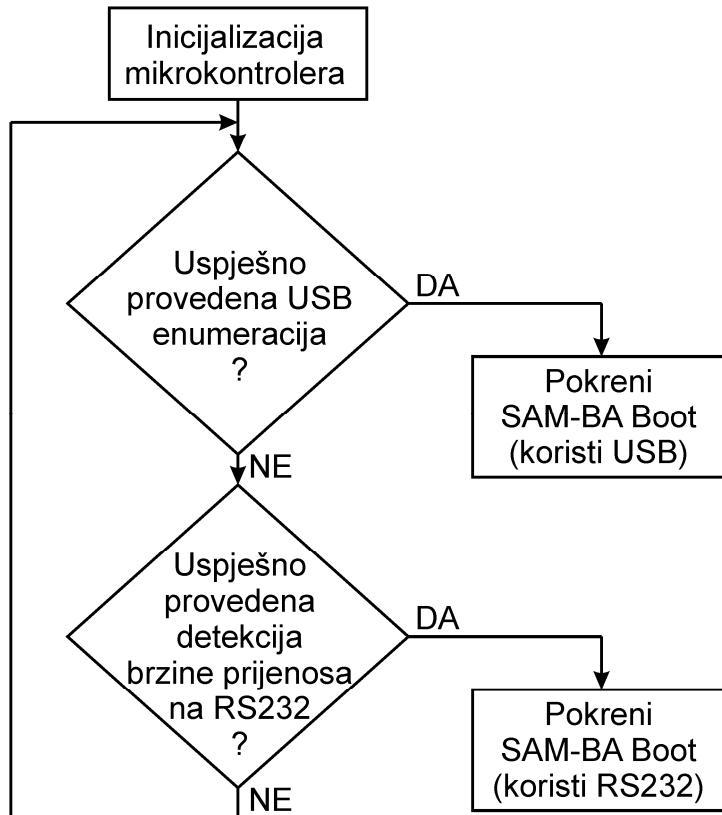
- uočiti
 - FFPI priključci koriste se samo za vrijeme programiranja
⇒ u normalnom radu mikrokontrolera, ovi priključci pripadaju ulazno-izlaznim sklopovima PIOA i PIOB
- svojstva ovakvog načina programiranja
 - brz
 - pogodan za serijsku proizvodnju
 - nije pogodan u fazi razvoja programske podrške

6.4.2 Učitavanje pomoću SAM-BA programa

- ideja
 - Kako učitati u Flash memoriju pomoću samog mikrokontrolera
 - rješenje
 - μC mora sadržavati program koji
 - inicijalizira sučelja za komunikaciju s osobnim računalom
 - uspostavlja vezu s osobnim računalom
 - učitava sadržaj Flash memorije iz osobnog računala
 - upisuje učitane podatke u Flash memoriju
 - postavlja zastavicu GPNVM
⇒ mapiranje Flash Memorije u prvi MB
- ⇒ ovaj program nalazi se u ROM memoriji mikrokontrolera
(zove se *SAM Boot Assistant*, SAM-BA)



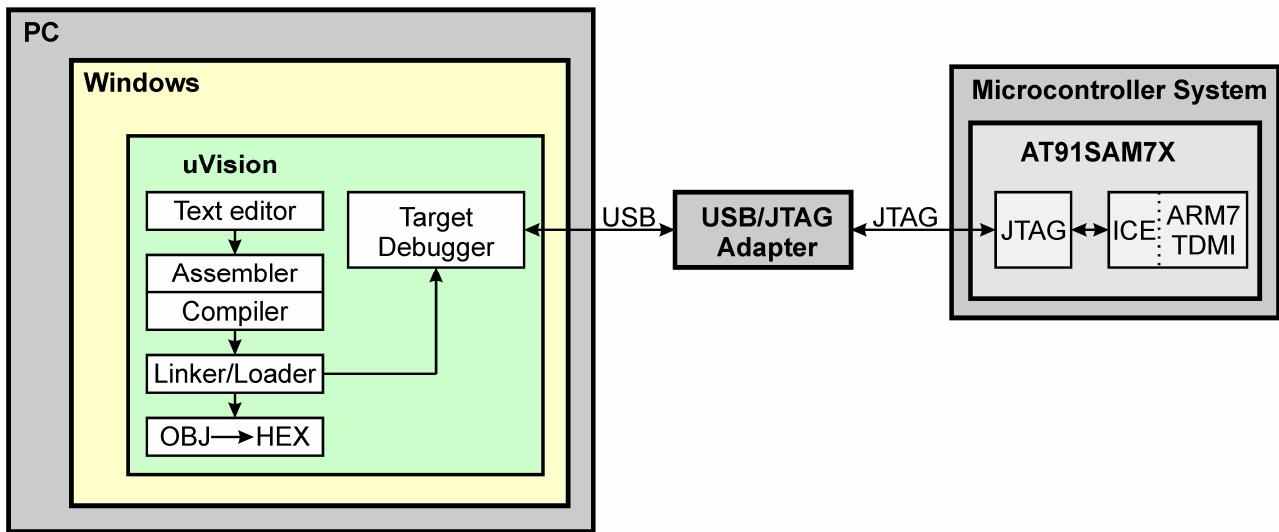
- nakon pritiska tipke ERASE
 - briše se sadržaj Flash memorije
 - u prvi MB se mapira ROM
- nakon reseta kojem prethodi ERASE
 ⇒ mikrokontroler počinje izvršavati program SAM-BA



- SAM-BA Boot koji koristi RS232 komunikaciju
 - koristi se DBGU sklopovlje
 ⇒ linije DRXD i DTXD
 - radi s brzinom prijenosa 115200 bitova/s
 - ISP šalje preddefiniranu poruku (0x80, 0x80, ...)
 - SAM-BA će sam podešiti parametre svog baud-rate generatora
 ⇒ frekvencija kristala u glavnem oscilatoru mikrokontrolera može biti bilo koja
- SAM-BA Boot koji koristi USB komunikaciju
 - koristi se USB sklopovlje
 ⇒ PC detektira µC kao standardnu USB komponentu (*Device*)
 - frekvencija kristala u glavnem oscilatoru mikrokontrolera mora biti 18.432 MHz

6.4.3 Učitavanje korištenjem sklopovlja za uhodavanje

- učitavanje se može izvesti preko sučelja za uhodavanja (*In Circuit Debugger, ICE*)



- ICE omogućuje i druge funkcije osim učitavanja programa
 - pokretanje programa
 - postavljanje točaka prekida
 - itd

6.5 Osnovne inicijalizacije

- programska podrška za µC piše se u jeziku C ili C++
- ipak, neke inicijalizacije praktičnije (preglednije) je napisati u asembleru
⇒ ove inicijalizacije obično se grupiraju u takozvanu **startup** datoteku
- za ispravan rad mikrokontrolera potrebno je
 - inicijalizirati prekidne vektore
 - inicijalizirati sklopolje
 - memoriski kontroler (*Memory Controller*)
⇒ podesiti broj stanja čekanja za pristup Flash memoriji
 - upravljačko sklopolje (*System Controller*)
⇒ konfigurirati sklopolje za reset
⇒ ugasiti *Watch-dog*
⇒ konfigurirati izvore takta
 - inicijalizirati početne adrese stogova

6.5.1 Inicijalizacija prekidnih vektora

- kod inicijalizacije je potrebno uzeti u obzir slijedeće
 - procesor nakon reseta učitava instrukciju na adresi 0
⇒ **kod mora biti smješten na adresu 0**
 - osnovni vektori iznimaka nalaze se na fiksnim adresama, 0x0000 0004, 0x0000 0008, 0x0000 000C, itd. (vidi poglavlje 3.2.4)
⇒ poželjno je napraviti rutine koje poslužuju prekide
⇒ **smjestiti pozive rutina na adrese na koje pokazuju vektori**
- programski odsječak koji predstavlja početak startup koda

```

; definicija kodnog odsjecka s imenom RESET
        AREA RESET, CODE, READONLY
        ARM          ; koristit će ARM skup instrukcija

; Vektori iznimaka

Vectors    LDR  PC,Reset_Addr
            LDR  PC,Undef_Addr
            LDR  PC,SWI_Addr
            LDR  PC,PAbt_Addr
            LDR  PC,DAbt_Addr
            NOP           ; Reservirani vektor
            LDR  PC,IRQ_Addr
            LDR  PC,FIQ_Addr

Reset_Addr DCD  Reset_Handler ; DCD alocira rijec u mem.
Undef_Addr DCD  Undef_Handler
SWI_Addr   DCD  SWI_Handler
PAbt_Addr  DCD  PAbt_Handler
DAbt_Addr  DCD  DAbt_Handler
            DCD  0           ; Rezervirani vektor
IRQ_Addr   DCD  IRQ_Handler
FIQ_Addr   DCD  FIQ_Handler

; Rutine za posluživanje iznimaka => beskonacne petlje

Undef_Handler B  Undef_Handler
SWI_Handler   B  SWI_Handler
PAbt_Handler  B  PAbt_Handler
DAbt_Handler  B  DAbt_Handler
IRQ_Handler   B  IRQ_Handler
FIQ_Handler   B  FIQ_Handler

; Rutina koja se izvrsava nakon reseta (Reset Handler)

        EXPORT Reset_Handler
Reset_Handler
;
;
; Ovdje idu instrukcije koje se izvršavaju nakon reseta.
;
```

- prekidni vektori moraju biti na odgovarajućim adresama
 - ⇒ ovaj odsječak mora početi od adrese 0
 - ⇒ to treba dati do znanja linker-u
- način povezivanja opisan je u datoteci s nastavkom **.sdc** (*Scatter-Loading Description File*, SDC)
- primjer SDC datoteke

```
LR_IROM1 0x00000000 0x00020000
{
    ER_IROM1 0x00000000 0x00020000
    {
        ; UOCITI: Kodni odsjecak s imenom RESET ide prvi,
        ; tj. nalazi se na na adresi 0
        *.o (RESET, +First)
        *(InRoot$$Sections)
        .ANY (+RO)
    }
    RW_IRAM1 0x00200000 0x00008000
    {
        .ANY (+RW +ZI)
    }
}
```

- ukoliko se koristi predefinirana startup datoteka i povezivanje izvodi automatski, ova datoteka će se automatski generirati
 - ovo je dobro "za početak"
 - pisanje "ozbiljne" programske podrške traži od korisnika da vlastitom direktivama pojedinih programskih alata
 - ⇒ opcije poziva alata za prevođenje i povezivanje definira korisnik (ručno)
- uočiti
 - u gornjem primjeru prekidne rutine ne rade ne rade ništa (*dummy*)
 - u dalnjim poglavljima pokazat ćemo kako se obrađuju prekidi koje na IRQ i FIQ postavlja AIC

6.5.2 Inicijalizacija memorijskog kontrolera

- proizvođač mikrokontrolera specificira (u poglavlju *AC Characteristics*)
 - najveću frekvenciju takta procesora
 - brzinu rada memorije
- primjer - kod našeg mikrokontrolera
 - najveća frekvencija takta procesora (*Maximum Master Clock Frequency*)

$$f_{\text{Max}} = \left(\frac{1}{t_{\text{CPMCK}}} \right)_{\text{Max}} = 55 \text{MHz}$$

- brzina rada Flash memorije
 - čitanje

⇒ dana je najveća frekvencija takta za zadani broj stanja čekanja (*Wait States*)

| Broj stanja čekanja | Trajanje čitanja | Maksimalna frekvencija takta |
|---------------------|------------------|------------------------------|
| 0 | 1 period takta | 30 MHz |
| 1 | 2 period takta | 55 MHz |
| 2 | 3 period takta | 55 MHz |
| 3 | 4 period takta | 55 MHz |

- pisanje

⇒ vrijeme programiranja (*Program Cycle Time*)
maksimalno 6ms

⇒ vrijeme potrebno za brisanje (*Erase Time*)
minimalno 15ms
- registri koji upravljaju radom Flash kontrolera
 - MC_FMR (*Memory Controller - Flash Mode Register*)
 - MC_FCR (*Memory Controller - Flash Command Register*)
 - MC_FSR (*Memory Controller - Flash Status Register*)

- broj stanja čekanja upisuje se u registar MC_FMR

| | | | | | | | | | |
|-------|-------------|------|------------|-------------|-----|--------------|--------------|---|-------------|
| 31-24 | 23-16 | 15-9 | 9-8 | 7 | 6-4 | 3 | 2 | 1 | 0 |
| - | FMCN | - | FWS | NEBP | - | PROGE | LOCKE | - | FRDY |

- FRDY *Flash Ready Interrupt Enable*
- LOCKE *Lock Error Interrupt Enable*
- PROGE *Programming Error Interrupt Enable*
- NEBP *No (page) Erase Before Programming*
- FWS *Flash Wait State*
- FMCN *Flash Microsecond Cycle Number*
po preporuci proizvođača iznosi
broj perioda takta u jednoj mikrosekundi

Primjer

- prepostavimo da procesor radi s taktom frekvencije

$$f_{\text{Max}} = 48 \text{MHz}$$

⇒ parametri memorije bit će (vidi tablicu)

$$\text{FWS}=1$$

$$\text{FMCN}=48=0x30$$

- programski odsječak

```

EFC_BASE      EQU      0xFFFFFFF00 ; EFC Base Address
EFC0_FMR      EQU      0x60       ; EFC0_FMR Offset
EFC0_FMR_Val  EQU      0x00300100 ; FWS=1, FMCN=0x30

LDR    R0, =EFC_BASE
LDR    R1, =EFC0_FMR_Val
STR   R1, [R0, #EFC0_FMR]

```

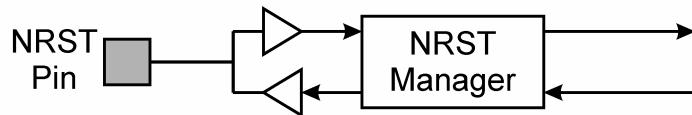
- uočiti

- *Embedded Flash Controller 0* (EFC0)

⇒ postoje mikrokontroleri koji imaju više od jednog kontrolera

6.5.3 Inicijalizacija sklopolja za reset

- podsjetimo se
 - u sklopolju za upravljanjem resetom (NRSTC) o vanjskom priključku brine sklop NRST Manager



- NRST Manager upravlja
 - propuštanjem reseta s priključka NRST u mikrokontroler
 - generiranju reseta koji izlazi na priključak NRST
- radom NRST Manager-a upravljaju registri

RSTC_MR (Reset Controller Mode Register)

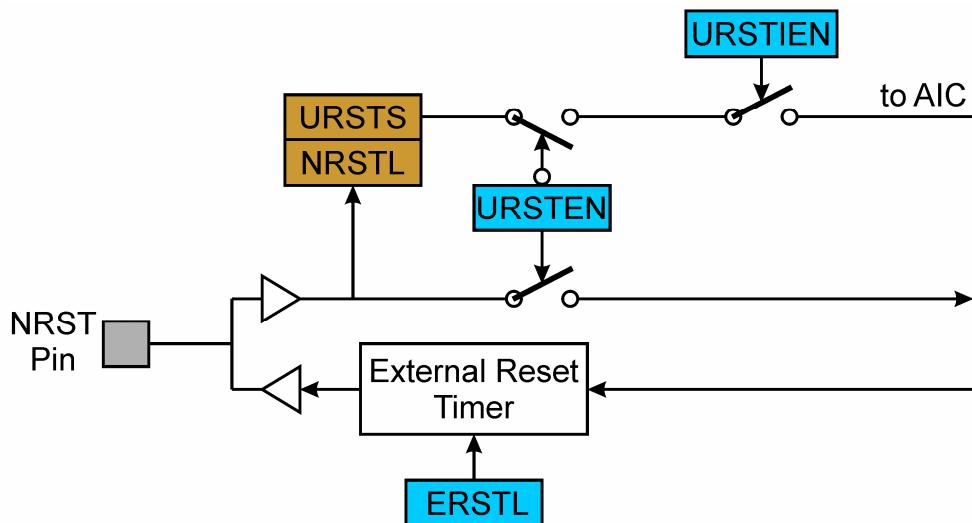
| | | | | | | | | |
|-------|-------|---------------|-------|--------------|-----|----------------|-----|---------------|
| 31-24 | 23-17 | 16 | 15-12 | 11-8 | 7-5 | 4 | 3-1 | 0 |
| KEY | - | BODIEN | - | ERSTL | - | URSTIEN | - | URSTEN |

RSTC_SR (Reset Controller Status Register)

| | | | | | | | |
|-------|--------------|--------------|-------|---------------|-----|---------------|--------------|
| 31-18 | 17 | 16 | 15-11 | 10-8 | 7-2 | 1 | 0 |
| - | SRCMP | NRSTL | - | RSTTYP | - | BODSTS | URSTS |

RSTC_CR (Reset Controller Control Register)

| | | | | | |
|-------|------|---------------|---------------|---|----------------|
| 31-24 | 23-4 | 3 | 2 | 1 | 0 |
| KEY | - | EXTRST | PERRST | - | PROCRST |



- u našem primjeru, konfiguracija ima oblik
 - želimo da vanjski reset uzrokuje reset procesora, a ne prekid
⇒ URSTEN=1 (*User ReSeT ENable*)
⇒ URSTIEN nije bitan (*User ReSeT Interrupt ENable*)
 - μC neće generirati reset signal
⇒ ERSTL nije bitan (*External ReSeT Length*)
- programski odsječak

```

RSTC_BASE      EQU      0xFFFFFD00 ; RSTC Base Address
RSTC_MR        EQU      0x08      ; RSTC_MR Offset
RSTC_MR_Val    EQU      0xA5000101 ; URSTEN=1, KEY=0xA5

        LDR      R0, =RSTC_BASE
        LDR      R1, =RSTC_MR_Val
        STR      R1, [R0, #RSTC_MR]

```

- uočiti
 - kao i kod *Watch-dog* sklopa, upis se izvodi pomoću zaporce (KEY)

6.5.4 Inicijalizacija *Watch-dog* sklopa

- kod uhodavanja sklopova rad *Watch-dog* sklopa je nepoželjan
⇒ potrebno je onemogućiti njegov rad
- o *Watch-dog* sklopu bilo je govora u poglavljju 3.5.5 pa to ovdje nećemo ponavljati
- onemogućavanje rada *Watch-dog* sklopa
⇒ WDDIS=1

WDT_MR (*Watch-dog Timer Mode Register*)

| | | | | | | | | |
|----|-----------|---------|-------|-------|---------|---------|--------|------|
| 31 | 30 | 29 | 27-16 | 15 | 14 | 13 | 12 | 11-0 |
| - | WDIDLEHLT | WDBGHLT | WDD | WDDIS | WDRPROC | WDRSTEN | WDFIEN | WDV |

- programski odsječak

```

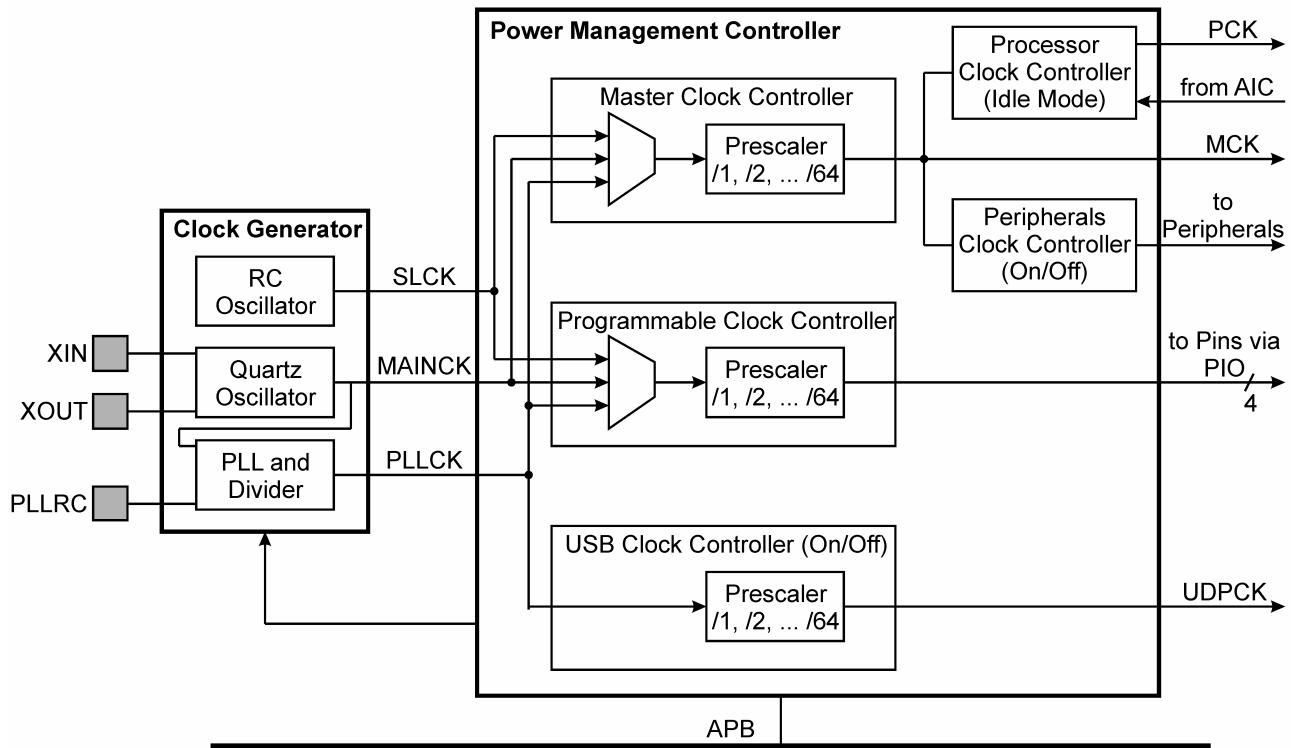
WDT_BASE      EQU      0xFFFFFD40 ; WDT Base Address
WDT_MR        EQU      0x04      ; WDT_MR Offset
WDT_MR_Val    EQU      0x00008000 ; WDDDIS=1

        LDR      R0, =WDT_BASE
        LDR      R1, =WDT_MR_Val
        STR      R1, [R0, #WDT_MR]

```

6.5.5 Inicijalizacija izvora takta

- napomena
 - predefinirane vrijednosti upravljačkih registara takta i memorijskog kontrolera su takve da mikrokontroler radi regularno (ali sporo)
 - memorijski kontroler treba inicijalizirati prije izvora takta
- podsetimo se



- nakon uključenja napajanja procesor radi na taktu od 32 kHz (SLCK)**
- osnovna frekvencija glavnog takta iznosi manje od 20MHz
- najveća brzina dobiva se korištenjem PLL sklopa
- da bi procesor počeo raditi na taktu PLLCK, potrebno je **programski**
 - omogućiti rad kristalnog oscilatora**
 - pričekati da se oscilator utitra**
 - podesiti parametre PLL sklopa i omogućiti njegov rad**
 - pričekati da se PLL utitra**
 - odabrati odgovarajući faktor dijeljenja**
 - pričekati da MCK utitra**
 - odabrati izvor takta**
 - pričekati da MCK utitra**

6.5.5.1 Inicijalizacija glavnog oscilatora

- registar za upravljanje radom glavnog oscilatora

CKGR_MOR (*Clock Generator Main Oscillator Register*)

| | | | | |
|-------|----------------|-----|------------------|---------------|
| 31-16 | 15-8 | 7-2 | 1 | 0 |
| - | OSCOUNT | - | OSCBYPASS | MOSCEN |

- **MOSCEN** (*Main Oscillator Enable*)
- **OSCBYPASS** (*Oscillator Bypass*)
- **OSCOUNT** (*Main Oscillator Start-up Time*)
 - uključivanje oscilatora
MOSCEN=1
 - podešavanje vremena utitravanja
 - u OSCOUNT treba upisati vrijeme potrebno za utitravanje oscilatora
⇒ izražava se kao **(broj perioda SLCK)/8**
 - vrijeme utitravanja specificira proizvođač mikrokontrolera
 - u našem slučaju

$$f_{\text{Quartz}} = 18.432\text{MHz} \Rightarrow \text{frekvencija glavnog oscilatora}$$

$$f_{\text{Quartz}} \approx 16\text{MHz} \Rightarrow \text{uzeti iz datasheet-a } t_{\text{ST}} = 1.4\text{ms}$$

$$t_{\text{ST}} = 1.4\text{ms} \Rightarrow \text{najdulje vrijeme utitravanja}$$

$$t_{\text{SCLK}} = 1/32\text{kHz} \Rightarrow \text{period RC oscilatora}$$

$$\text{OSCOUNT} = \left\lceil \frac{1}{8} \frac{t_{\text{ST}}}{t_{\text{SCLK}}} \right\rceil = 6$$
 - preporučuje se uzeti faktor sigurnosti ili ako se ne radi o low-power aplikaciji, čak uzeti maksimalnu vrijednost od 0xFF
 - programski odsječak za pokretanje glavnog oscilatora

```

PMC_BASE      EQU      0xFFFFFC00 ; PMC Base Address
PMC_MOR       EQU      0x20        ; PMC_MOR Offset
PMC_MOR_Val   EQU      0x0000FF01 ; MOSCEN=1, OSCOUNT=0xFF

LDR R0, =PMC_BASE

; "Ukljucivanje" oscilatora
LDR R1, =PMC_MOR_Val
STR R1, [R0, #PMC_MOR]

; Cekanje da se oscilator utitra
MOSCS_Loop   LDR R2, [R0, #PMC_SR]
              ANDS R2, R2, #PMC_MOSCS
              BEQ MOSCS_Loop

```

6.5.5.2 Inicijalizacija PLL sklopa i djelila

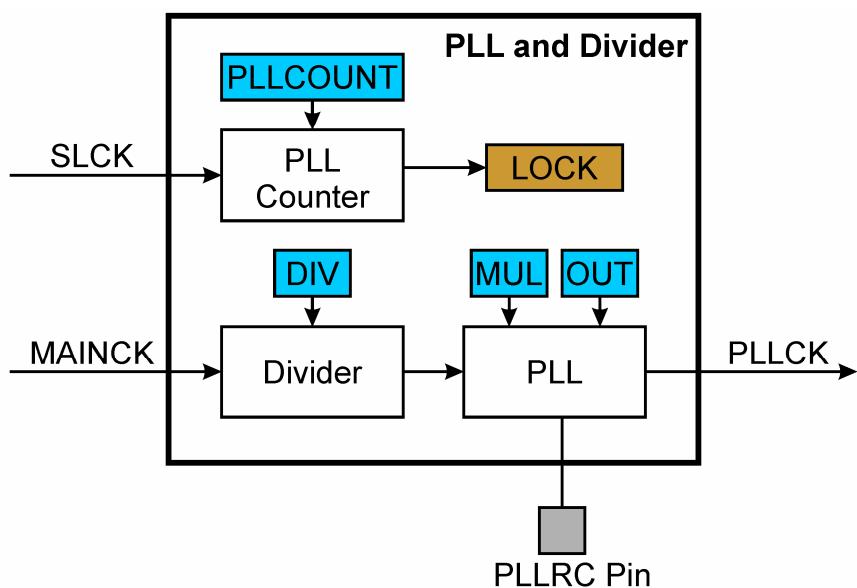
- registri za upravljanje radom PLL sklopa i djelila

CKGR_PLLR (Clock Generator PLL Register)

| 31 | 30 | 29-28 | 27 | 26-16 | 15-14 | 13-8 | 7-0 |
|----|----|---------------|----|------------|------------|-----------------|------------|
| - | - | USBDIV | - | MUL | OUT | PLLCOUNT | DIV |

PMC_SR (PMC Status Register)

| 31-11 | 10 | 9 | 8 | 7-4 | 3 | 2 | 1 | 0 |
|-------|---------|---------|---------|-----|--------|-------------|---|-------|
| - | PCKRDY2 | PCKRDY1 | PCKRDY0 | - | MCKRDY | LOCK | - | MOSCS |



- PLL može raditi u dva frekvencijska područja
 - od 80MHz do 160 MHz OUT=00b
 - od 150MHz do 200 MHz OUT=10b
 ⇒ potrebno je podešiti OUT ovisno izlaznoj frekvenciji PLL-a
- izlazna frekvencija PLL-a

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

- PLLCK na putu prema procesoru prolazi kroz djelilo (*Prescaler*) koji dijeli frekvenciju s faktorom PRES

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

- uočiti
 - odabir MUL, DIV i PRES nije jednoznačan
 - odgovarajućom kombinacijom postiže se frekvencija blizu željene

Primjer

Uz frekvenciju glavnog oscilatora 18.432MHz potrebno je podesiti PLL tako da se dobije frekvencija od 48MHz.

- odabir

MUL=124, DIV=24, PRES=1

daje

$$f_{MCK} = f_{MAINCK} \frac{MUL + 1}{DIV \cdot 2^{PRES}} = 18.432\text{MHz} \frac{124 + 1}{24 \cdot 2^1} = 48\text{MHz}$$

- pritom je

$$f_{PLLCK} = 2^{PRES} f_{MCK} = 2^1 * 48\text{MHz} = 96\text{MHz}$$

što traži

OUT=00b (za područje od 80MHz do 160 MHz)

- utitravanje PLL-a čeka se dok PLL Counter ne odbroji od vrijednosti PLLCOUNT do 0
 - ovo vrijeme ovisi o vrijednostima R, C₁ i C₂ u filtru
 - često proizvođač mikrokontrolera daje ovaj parametar za dane vrijednosti R, C₁ i C₂

⇒ u našem slučaju odabrano je maksimalno vrijeme od cca 2ms
PLLCOUNT=63=0x3F

- programski odsječak

```

PMC_BASE      EQU      0xFFFFFC00 ; PMC Base Address
PMC_PLLR      EQU      0x2C       ; PMC_PLLR Offset
PMC_PLLR_Val  EQU      0x007C3F18 ;

        LDR      R0, =PMC_BASE

; Konfiguriranje PLL-a

        LDR      R1, =PMC_PLLR_Val
        STR      R1, [R0, #PMC_PLLR]

; Čekanje da se PLL utitra

PLL_Loop    LDR      R2, [R0, #PMC_SR]
            ANDS    R2, R2, #PMC_LOCK
            BEQ     PLL_Loop

```

6.5.5.3 Odabir takta

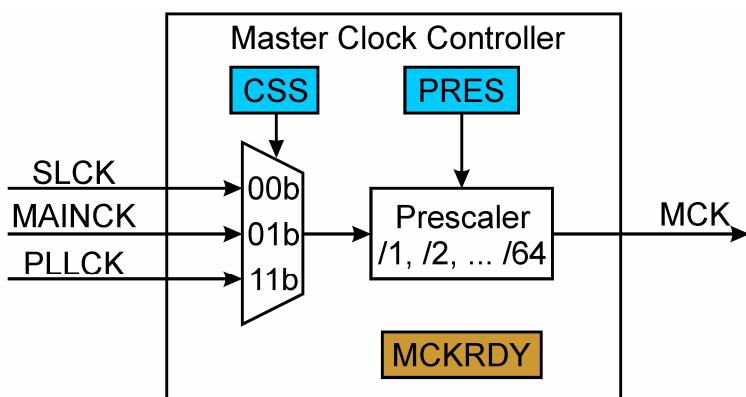
- registri za upravljanje glavnim kontrolerom takta

PMC_MCKR (Master Clock Register)

| | | | |
|------|---|------|-----|
| 31-5 | - | 4-2 | 1-0 |
| | | PRES | CSS |

PMC_SR (PMC Status Register)

| | | | | | | | | |
|-------|---------|---------|---------|-----|--------|------|---|-------|
| 31-11 | 10 | 9 | 8 | 7-4 | 3 | 2 | 1 | 0 |
| | PCKRDY2 | PCKRDY1 | PCKRDY0 | - | MCKRDY | LOCK | - | MOSCS |



- uočiti
 - PLLCK u pravilu ima previsoku frekvenciju za procesor
 - PLLCK na putu prema procesoru prolazi kroz djelilo (Prescaler)
 - MCK se ne smije pustiti u procesor prije nego se MCK uđe u stacionarno stanje**
⇒ MCK je spreman kad se postavi MCKRDY=1
- svaki upis u PMC_MCKR rezultira postavljanjem MCKRDY=0
- redoslijed postavljanja parametar
 - upisati željeni PRES
 - čekati sve doK je MCKRDY=1
 - upisati željeni CSS
 - čekati sve doK je MCKRDY=1

- programski odsječak

```
PMC_BASE      EQU      0xFFFFFC00 ; PMC Base Address
PMC_MCKR      EQU      0x30       ; PMC_MCKR Offset
PMC_MCKR_Val  EQU      0x00000007
PMC_PRES      EQU      (7<<2)    ; Prescaler Selection

        LDR      R0, =PMC_BASE

; Programiranje prescalera

        LDR      R1, =PMC_MCKR_Val
        AND      R1, #PMC_PRES
        STR      R1, [R0, #PMC_MCKR]
WAIT_Rdy1   LDR      R2, [R0, #PMC_SR]
        ANDS     R2, R2, #PMC_MCKRDY
        BEQ      WAIT_Rdy1

; Odabir izvora takta

        LDR      R1, =PMC_MCKR_Val
        STR      R1, [R0, #PMC_MCKR]
WAIT_Rdy2   LDR      R2, [R0, #PMC_SR]
        ANDS     R2, R2, #PMC_MCKRDY
```

- pitanje za vježbu
 - Zašto nakon postavljanja CSS treba čekati utitravanja, obzirom da se pauza izvodi programski na procesoru koji već radi s novim taktom?
- uočiti
 - nakon opisanih inicijalizacija sklopljiva za upravljanje taktom, takt za periferije i takt za USB je još uvijek ugašeni

6.5.6 Inicijalizacija stoga

- inicijalizacija stoga
 - podrazumijeva inicijalizaciju pokazivača na stog
SP=R13
- paziti
 - svaki način rada ima svoj vlastiti SP (vidi poglavlje 3.2.2.1)
⇒ potrebno je promijeniti način rada i upisati odgovarajuću vrijednost u SP
 - ponoviti postupak za svaki način rada
- način rada određen je bitovima M[0:4] u registru CPSR

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

- uočiti
 - istovremeno s postavljanjem polja M koje mijenja način rada, mogu se postaviti zastavice I i F (onemogućavanje prekida)
 - o ovom će biti još govora u poglavljju 7.7

- programski odsječak za inicijalizaciju stogova

```

; definicije nacina rada

Mode_USR      EQU      0x10
Mode_FIQ      EQU      0x11
Mode_IRQ      EQU      0x12
Mode_SVC      EQU      0x13
Mode_ABT      EQU      0x17
Mode_UND      EQU      0x1B
Mode_SYS      EQU      0x1F

; definicije velicina stogova

UND_Stack_Size  EQU      0x00000000
SVC_Stack_Size  EQU      0x00000008
ABT_Stack_Size  EQU      0x00000000
FIQ_Stack_Size  EQU      0x00000000
IRQ_Stack_Size  EQU      0x00000080
USR_Stack_Size  EQU      0x00000400

ISR_Stack_Size  EQU      (UND_Stack_Size+SVC_Stack_Size+\ 
                           ABT_Stack_Size+FIQ_Stack_Size+\ 
                           IRQ_Stack_Size)

; Alokacija memorije za stogove
        AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem    SPACE   USR_Stack_Size
__initial_sp SPACE   ISR_Stack_Size
Stack_Top

; Definicije bitova za onemogucavanje prekida
I_Bit        EQU      0x80
F_Bit        EQU      0x40

; Pocetak inicijalizacije pokazivaca na stog
        LDR      R0, =Stack_Top

; Inicijalizacija SP u nacinu rada Undefined
        MSR      CPSR_c, #Mode_UND:OR:I_Bit:OR:F_Bit
        MOV      SP, R0
        SUB      R0, R0, #UND_Stack_Size

; Inicijalizacija SP u nacinu rada Abort
        MSR      CPSR_c, #Mode_ABT:OR:I_Bit:OR:F_Bit
        MOV      SP, R0
        SUB      R0, R0, #ABT_Stack_Size

;
;          .
;          itd. za ostale nacine rada
;

EXPORT __initial_sp

```

6.5.7 Startup datoteka

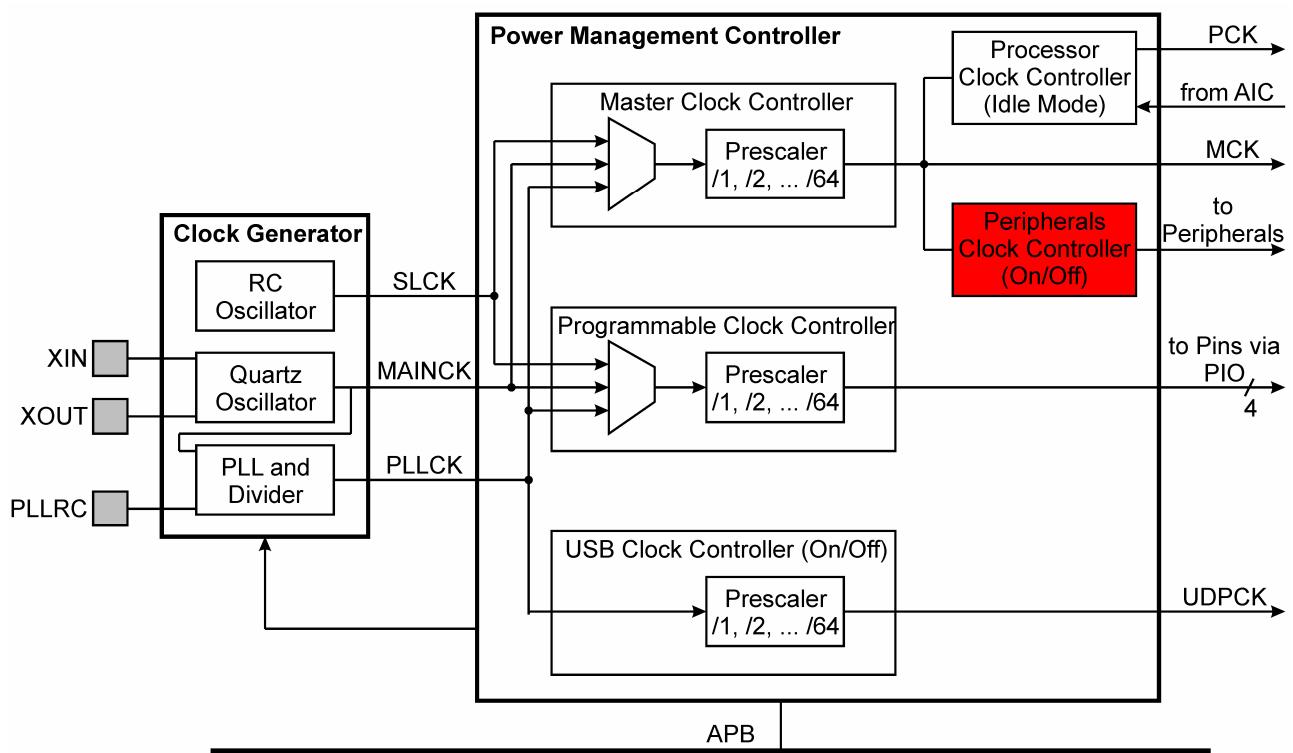
- sve opisane inicijalizacije sadržane su u startup datoteci koju obično daje proizvođač
- ponekad startup program sadrži i druge inicijalizacije
 - npr. remap memorije
- u uVision programskom okruženju postoji sustav za automatsko generiranje parametara startup datoteke (*Wizard*)
⇒ za ispravno korištenje *Wizard*-a potrebno je poznavati inicijalizacije !
- nakon inicijalizacija, startup datoteka sadrži programski odsječak

```
IMPORT  __main
LDR    R0, =__main
BX    R0          ; skoci na "main"
      ; promijeni skup instr. (ARM/Thumb)
```

- uočiti
 - nakon instrukcije BX napušta se kod za inicijalizaciju i počinje izvršavanje korisnikovog programa

6.6 Prvi program

- prvi program za uhodavanje sklopovlja treba biti jednostavan
⇒ npr. paljenje i gašenje svijetleće diode (*Light Emitting Diode, LED*)
- u našem primjeru, LED je spojen na **priklučak PB0**
⇒ treba "uključiti" PIOB upravljačko sklopovlje
⇒ treba konfigurirati PIOB upravljačko sklopovlje
- podsjetimo se
 - u sklopu osnovnih inicijalizacija podešen je glavni takt (MCK)
 - još je potrebno "pustiti" takt na željene periferije
⇒ o tome brine **Peripheral Clock Controller**



- sklop *Peripheral Clock Controller* vidi se kroz tri adrese
 - **PMC_PCSR** (*PMC Peripheral Clock Status Register*)
 - registr se može samo **pisati**
 - upis jedinice na odgovarajuće mjesto **uključuje** takt
 - **PMC_PCDR** (*PMC Peripheral Clock Disable Register*)
 - registr se može samo **pisati**
 - upis jedinice na odgovarajuće mjesto **isključuje** takt
 - **PMC_PCER** (*PMC Peripheral Clock Enable Register*)
 - registr se može samo **čitati**
 - sadrži 1 na bitovima "uključenih" periferija

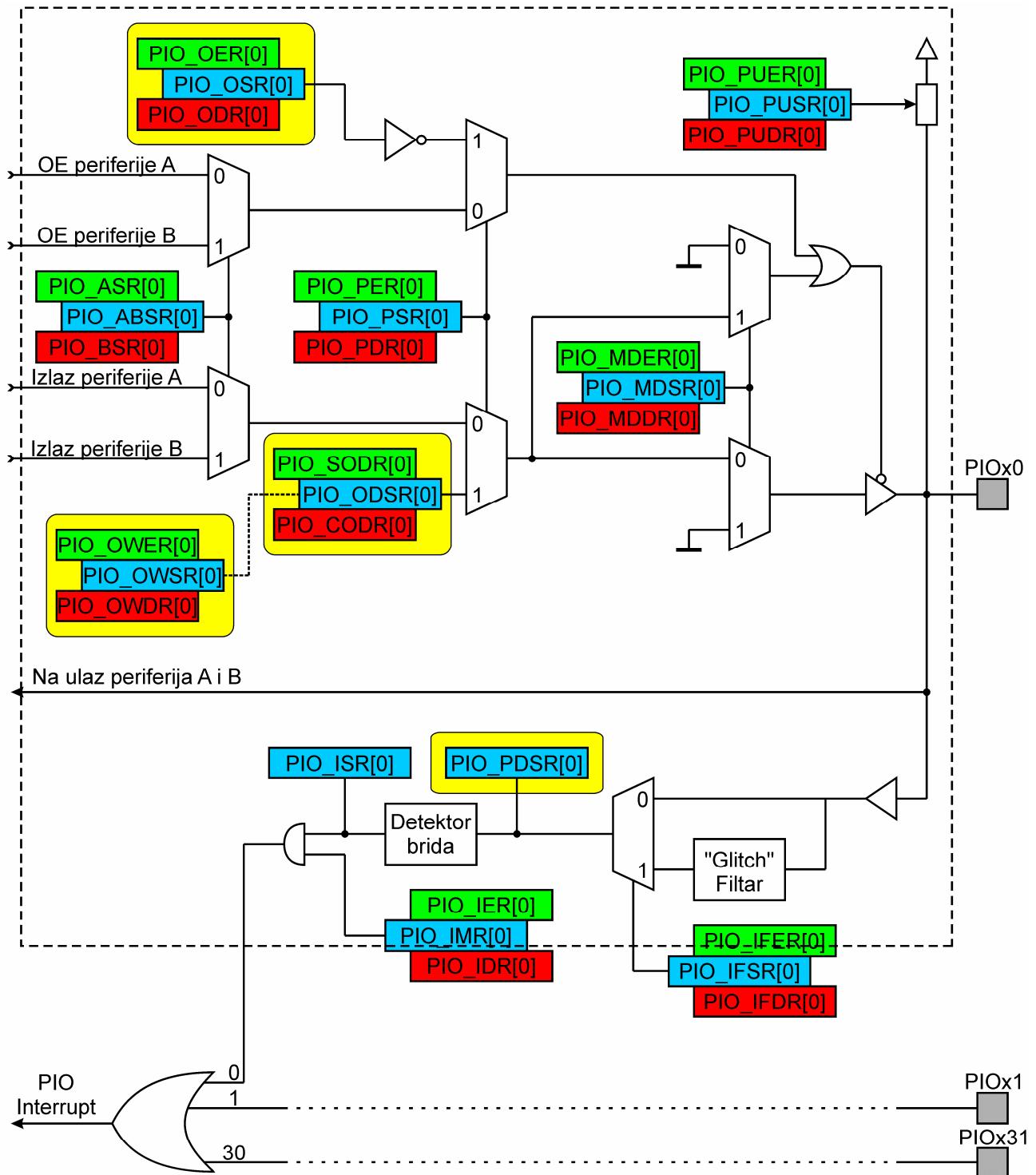
- raspored bitova u ovim registrima

| Bit | Identifikator (Peripheral ID, PID) | Oznaka periferije |
|-----|---------------------------------------|-------------------|
| - | 0 | AIC-FIQ |
| - | 1 | SYSC |
| 2 | 2 | PIOA |
| 3 | 3 | PIOB |
| 4 | 4 | SPI0 |
| 5 | 5 | SPI1 |
| 6 | 6 | US0 |
| 7 | 7 | US1 |
| 8 | 8 | SSC |
| 9 | 9 | TWI |
| 10 | 10 | PWMC |
| 11 | 11 | UDP |
| 12 | 12 | TC0 |
| 13 | 13 | TC1 |
| 14 | 14 | TC2 |
| 15 | 15 | CAN |

| Bit | Identifikator (Peripheral ID, PID) | Oznaka periferije |
|-----|---------------------------------------|-------------------|
| 16 | 16 | EMAC |
| 17 | 17 | ADC |
| 18 | - | - |
| 19 | - | - |
| 20 | - | - |
| 21 | - | - |
| 22 | | - |
| 23 | - | - |
| 24 | - | - |
| 25 | - | - |
| 26 | - | - |
| 27 | - | - |
| 28 | - | - |
| 29 | - | - |
| 30 | 30 | AIC-IRQ0 |
| 31 | 31 | AIC-IRQ1 |

- u našem primjeru koristimo PLOB
 ⇒ uključivanje takta (samo) za PLOB postiže se komandom
`*AT91C_PMC_PCER=(1<<AT91C_ID_PLOB); // AT91C_ID_PLOB=3`
- uočiti
 - registri nisu bit-adresabilni
 ali
 - razdvajanjem registara za *Enable* i *Disable* postiže se privid bit-adresabilnosti
 - periferijama koje odgovaraju bitovima u registru PMC_PCER u koje je upisana 0 *nije došlo do promjene stanja*
- opisani princip *Enable/Disable/Status* primjenjuje se u većini upravljačkih registara
 ⇒ vidi daljnji tekst

- upravljačko sklopolje jedne ulazno-izlazne linije i pripadajući upravljački registri



- registri za upravljanje izlaznim funkcijama priključka
 - OER/ODR/OSR (*Output Enable/Disable/Status Register*)
 - ASR/ABR/ABSR (*A Select / B Select / AB Status Register*)
 - PER/PDR/PSR (*PIO Enable/Disable/Status Register*)
 - SODR/CODR, ODSR (*Set/Clear/ Output Output Data Register, Output Data Status Register*)
 - PUSR/PUDR/PURS (*Pull-up Enable/Disable/Status Register*)
 - MDER/MDDR/MDSR (*Multi-driver Enable/Disable/Status Reg.*)
 - OWER/OWDR/OWSR (*Output Write Enable/Disable/Status Reg.*)
- registri za upravljanje ulaznim funkcijama priključka
 - IFER/IFDR/IFSR (*Glitch Input Filter Enable/Disable/Status Reg.*)
 - PDSR (*Pin Data Status Register*)
 - IER/IDR/IMR (*Interrupt Enable/Disable/Mask Register*)
 - ISR (*Interrupt Status Register*)
- stanje svih registara nakon reseta je 0x0000 0000
- uočiti
 - priključak može biti
 - povezan s periferijom A ili periferijom B
 - upravljan programski
 - priključak je dvosmjeran (*Bidirectional*)
 - stanje visoke impedancije
 - može biti upravljano od periferije A ili B
 - može biti upravljano programski
 - priključak ima pritezni otpornik prema napajanju koji se može isključiti
 - ukoliko je priključak upravljan programski
 - ⇒ upis se obavlja preko registara
 - **PIO_SODR** i **PIO_CODR** (bezuvjetno) ili
 - izravno upisom željenog stanja u **PIO_ODSR** pod uvjetom da je odgovarajući bit u **PIO_OWDR** jednak 1
 - ⇒ čitanje stanja na priključku obavlja se čitanjem registra **PIO_PDSR**

- u našem slučaju potrebno je

- odabrati programsku kontrolu priključkom PIOB0

```
*AT91C_PIOB_PER = AT91C_PIO_PB0; // LED_D102
```

- uključiti izlazno pojačalo na PIOB0

```
*AT91C_PIOB_OER = AT91C_PIO_PB0; // LED_D102
```

- program

```
#include <at91sam7x128.h>
#define LED_D102 AT91C_PIO_PB0 // AT91C_PIO_PB0=(1<<0)

int main(void) {
    // Deklaracije
    unsigned int i;

    // Ukljucivanje radnog takta za PIOB.
    *AT91C_PMC_PCER = (1 << AT91C_ID_PIOB);

    // PIO kontroler upravlja prikljuckom PIOB1
    *AT91C_PIOB_PER = LED_D102;
    // ukljucivanje izlaznog pojacala
    *AT91C_PIOB_OER = LED_D102;

    while(1) {
        *AT91C_PIOB_CODR = LED_D102; // iskljuci LED
        for(i=0; i<1000000; i++);
        *AT91C_PIOB_SODR = LED_D102; // ukljuci LED
        for(i=0; i<1000000; i++);
    }
}
```

- uočiti

- korišteno je adresiranje registara pomoću pokazivača

- adresiranje pomoću pokazivača na strukturu imalo bi oblik

```
// Deklaracije
AT91S_PMC *pPMC = AT91C_BASE_PMC;
// Ukljucivanje radnog takta za PIOB.
pPMC -> PMC_PCER = (1 << AT91C_ID_PIOB);
```

- pauza je izvedena pomoću "prazne" **for** petlje

- u "stvarnom" programu to se **NE RADI** ovako jer

- prevodilac može u fazi optimizacije ukloniti ovaj dio koda jer ne radi ništa

⇒ koristiti najnižu razinu optimizacije kod prevodioca

- vrijeme trajanja treba odrediti eksperimentalno

⇒ mjeranjem vremena izvršavanja petlje (u simulatoru)

6.7 Pitanja za provjeru znanja

1. Što obuhvaća elektromehanička provjera sklopa? Što obuhvaća funkcionalno uhodavanje sklopoljia?
2. Opisati princip učitavanja programske podrške u Flash memoriju mikrokontrolera pomoću namjenskog programatora.
3. Opisati princip učitavanja programske podrške u Flash memoriju mikrokontrolera pomoću SAM-BA programske podrške.
4. Opisati princip učitavanja programske podrške u Flash memoriju mikrokontrolera korištenjem sklopoljia za uhodavanje.
5. Napisati programski odsječak za inicijalizaciju prekidnih vektora. Prepostaviti da su prekidne rutine beskonačne petlje.
6. Opisati inicijalizaciju osnovnu inicijalizaciju memorijskog kontrolera. Kad je potrebno podesiti broj stanja čekanja a kad to nije potrebno?
7. Opisati princip rada sklopoljia koje pogoni priključak za vanjski reset. Koje osnovne inicijalizacije je potrebno napraviti ako mikrokontroler ima vanjsku tipku za reset?
8. Opisati osnovu inicijalizaciju Watch-dog sklopa u fazi razvoja sklopoljia. Zašto je ona potrebna? Kako treba inicijalizirati Watch-dog sklop u stvarnom radu?
9. Opisati inicijalizaciju sklopoljia glavnog oscilatora.
10. Opisati inicijalizaciju PLL sklopoljia.
11. Opisati sklopolje za odabir takta na kojem radi procesor. Kako se izvodi njegova inicijalizacija?. Na što je potrebno posebno paziti kod inicijalizacije ovog sklopoljia?

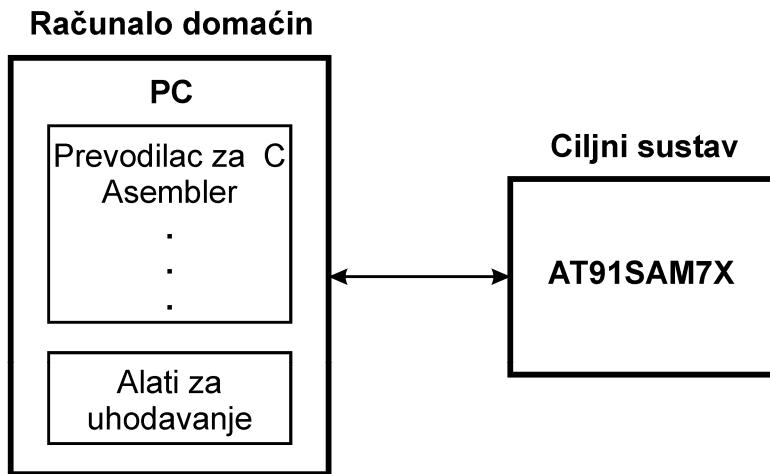
6.8 Zadaci za samostalni rad

1. U sklopu programskog paketa uVision, potrebno je napisati primjer programa za paljenje LED diode koji je objašnjen na predavanju. Tokom pisanja programa, posebnu pažnju posvetiti ispravnom konfiguriranju datoteke *startup.s*. Poslužiti se alatom za automatsko konfiguriranje. Proučiti sadržaj dobivene datoteke *startup.s* i uočiti funkcije pojedinih dijelova.

7 Programska podrška

7.1 Okolina za razvoj programske podrške

- okruženje za razvoj programske podrške



- 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.
- razvoj programske podrške do razine izvršnog koda
 - uvijek se izvodi na domaćinu
- uhodavanje se izvodi
 - isključivo na domaćinu
 - ⇒ koristi se simulator
 - isključivo na ciljnem sustavu
 - ⇒ program se učita u ciljni sustav i izvodi na sklopljivo
 - ⇒ ispitivanje ispravnosti izvodi se preko nekog vanjskog sučelja
 - npr. konzola s kojom se komunicira preko serijskog sučelja
 - djelomično na ciljnem sustavu a djelomično na domaćinu (*semihosting*)
 - ⇒ vidi daljni tekst

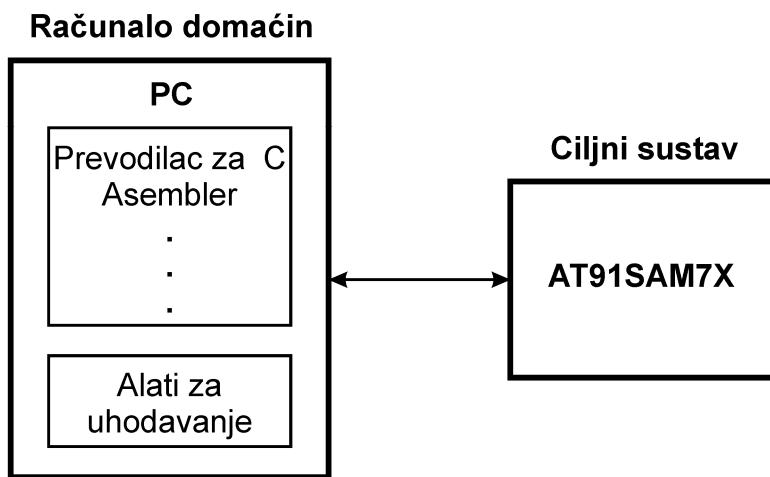
7.2 Biblioteke

7.2.1 Osnovni pojmovi

- biblioteke (*library*)
 - skup potprograma i drugih kodnih odsječaka koje korisnik ima na raspolaganju tokom razvoja programa
 - u objektnom programiranju biblioteke sadrže i klase
- razlikujemo
 - *runtime library*
 - *standard library*
 - *custom library*
- *runtime library*
 - kod prevodenja višeg programskog jezika (npr. C), neke operacije se mogu izravno mapirati u instrukcije procesora
 - npr. zbrajanje
 - 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*
 - ne miješati ove biblioteke s bibliotekama koje se povezuju u toku izvršavanja programa (*linking at run time*)
- *standard library*
 - biblioteka koju sadrži svaka implementacija dotičnog jezika
 - standardna biblioteka C jezika
 - ISO-C standard propisuje sučelja, a ne implementacije
 - ⇒ implementacija se ne bi trebala zvati *standard library*
 - sadrži osnovne matematičke funkcije, operacije sa stringovima, pretvorbe tipova, rad s datotekama, osnovne I/O funkcije (uglavnom za rad s konzolom)
 - sadrži zaglavlja (*headers*) (ukupno 24) i rutine
 - npr. **stdio.h**, **math.h**, itd.
- *custom library*
 - biblioteke koje sadrže specifične funkcije s aspekta aplikacije ili računalnog sustava

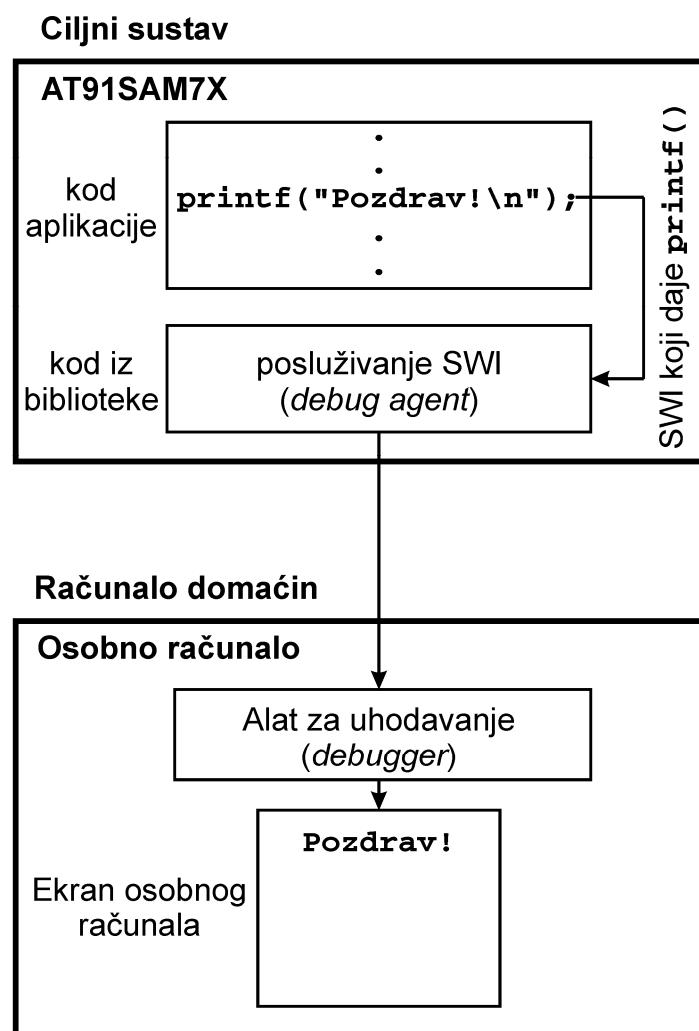
7.2.2 Funkcije koje koriste vanjsko sklopolje

- implementacija tj. programski kod mnogih standardnih funkcija specifičan je za pojedino sklopolje (*target dependency*)
 - ⇒ ovise o izvedbi sklopolja **izvan** procesora
- primjer
 - **printf**, **scanf**, **fread**, **fwrite**, i dr.
 - ove funkcije NISU ovisne o sklopolju (*target independent*)
 - ove funkcije pozivaju "niže" funkcije
 - **fgetc**, **fputc** i dr.
 - ove funkcije SU ovisne o sklopolju (*target dependent*)
- Kako su ove funkcije izvedene u ARM bibliotekama?
 - ⇒ **Niti jedna funkcija u biblioteci NIJE ovisna o sklopolju !**
 - **Kako su u tom slučaju funkcije implementirane?**
- podsjetimo se
 - okruženje za razvoj programske podrške



- programska podrška se može uhodavati djelomično na ciljnem sustavu a djelomično na računalu domaćinu (**semihosting**)
- funkcije koje koriste sklopolje u ARM bibliotekama implementirane su tako da da podržavaju semihosting
 - ⇒ Funkcije čiji rad ovisi o sklopolju tokom svog izvršavanja daju zahtjev za SWI.
 - ⇒ SWI poslužuje **agent** (*debug agent*)

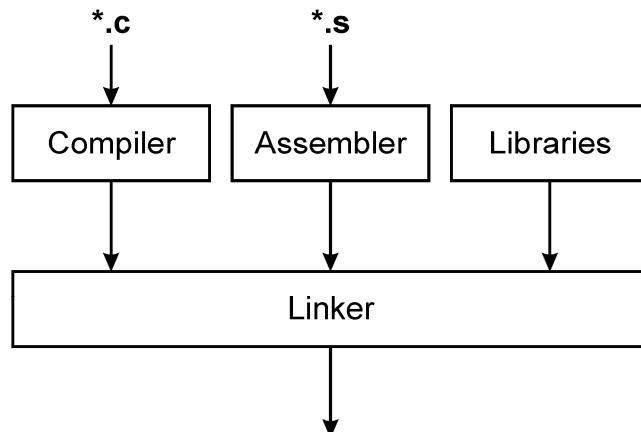
- primjer programa koji koristi funkciju **fwrite**
 - program se može izvoditi na simulatoru na PC-u
 - ⇒ **fwrite** poziva **fputc**
 - ⇒ **fputc** pomoću SWI komunicira s agentom
 - ⇒ agent otvara datoteku na osobnom računalu
 - program se može izvoditi na procesoru na ploči koja još nisu zalemljene sve periferije
 - ⇒ **fwrite** poziva **fputc**
 - ⇒ **fputc** pomoću SWI komunicira s agentom
 - ⇒ agent preko JTAG sučelja komunicira s PC-om
 - ⇒ program na PC-u otvara datoteku na PC-u
- primjer programa koji koristi funkciju **printf()**



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

7.2.3 Korištenje biblioteka

- biblioteke koristi sustav za povezivanje (*linker*)



- sustav za povezivanje podržava (za ARM prosecore)
 - "bare metal" model
 - tipičan za razvoj ugradbenih računalnih sustava
 - izlaz je cijelovit program koji radi izravno sa sklopoljem
 - program može sadržavati i operacijski sustav
 - djelomično povezivanje (*partial linking*)
 - izlaz kod ovakvog povezivanja predstavlja ulaz kod narednog povezivanja
- moguća su 2 načina korištenja biblioteka
 - program sadrži funkciju **main()**
 - ⇒ okruženje za biblioteke se inicijalizira
 - krajnji kod se može izvršavati
 - u semihost okruženju
 - ⇒ koriste se izvorne rutine iz biblioteka
 - ⇒ ovo je predefinirani način korištenja
 - na stvarnom sklopolju
 - ⇒ funkcije koje koriste I/O treba redefinirati
 - ⇒ u kod treba dodati simbol
__use_no_semihosting_swift
 - program NE sadrži funkciju **main()**
 - ⇒ okruženje za rad s bibliotekama se NE inicijalizira
 - ⇒ biblioteke se ne koriste ili se koriste samo djelomično
 - ovakav model naziva se **bare machine**
- razlikovari *bare metal* model povezivanja i *bare machine*
 - kod za *bare machine* je uvijek dobiveno korištenjem *bare metal* modela povezivanja
 - ⇒ ovaj kod ne inicijalizira biblioteke
 - krajnji kod koji inicijalizira i koristi biblioteke također je dobiven korištenjem *bare metal* modela povezivanja, ali se ne naziva *bare machine*

7.3 Bare machine

- ne postoji funkcija `main()`
- ne inicijalizira se okruženje za korištenje biblioteka

Primjer

Kako izgleda kod koji nastaje nakon prevođenja i povezivanja asemblerorskog koda koji izvodi osnovne inicijalizacije opisanog u poglaviju 6.5 i danog C-programma?

```
#include <at91sam7x128.h>

void glavni(void) {
    *AT91C_PMC_PCER = (1 << 0); // Takt za PIOB
    *AT91C_PIOB_PER = (1 << 0); // PIO Enable
    *AT91C_PIOB_OER = (1 << 0); // Output Enable
    *AT91C_PIOB_SODR = (1 << 0); // Uključi LED
}
```

Rješenje

- nakon inicijalizacija, potrebno je skočiti na `glavni`
⇒ kod za inicijalizaciju završava odsječkom

```
IMPORT glavni
LDR R0, =glavni
BX
```

- rezultat prevođenja i povezivanja

```

; pocetak STARTUP koda

    AREA RESET, CODE, READONLY ; odsjecak RESET
    ARM                      ; ARM skup instr.

; Inicijalizacija prekidnih vektora

    0x00000000 LDR PC,[PC,#0x0018] ; PC=0x0000 0058
    0x00000004 LDR PC,[PC,#0x0018]
    ;       ...
    0x00000008 LDR R0,[PC,#0x00C8]
    0x0000000C LDR R1,[PC,#0x00C8]
    0x00000010 STR R1,[R0,#0x0008]
    ;       ...

; Inicijalizacija sklopolija za reset

    0x00000058 LDR R0,[PC,#0x00C8]
    0x0000005C LDR R1,[PC,#0x00C8]
    0x00000060 STR R1,[R0,#0x0008]
    ;       ...

; Inicijalizacija memoriskog kontrolera
; Inicijalizacija Watch-dog sklopa
; Inicijalizacija izvora takta
; Inicijalizacija pokazivaca stoga i
; onemogućavanje prekida u pojedinim nacinima rada
; Ulaz u C kod

    IMPORT glavni
    LDR R0, = glavni
    0x00000120 LDR R0,[PC,#0x0020]
    BX R0
    0x00000124 BX R0
    0x00000128 DD 0xFFFFFD00
    ;       ...
    0x00000148 DD 0x0000014C

; Korisnikov root

void glavni(void) {
    *AT91C_PMC_PCER = (1 << 0); // Takt za PIOB
    0x0000014C MOV R0,#0x00000001
    0x00000150 MOV R1,#0x00000000
    0x00000154 STR R0,[R1,#-0x03F0]
    *AT91C_PIOB_PER = (1 << 0); // PIO Enable
    0x00000158 STR R0,[R1,#-0x0A00]
    *AT91C_PIOB_OER = (1 << 0); // Output Enable
    0x0000015C STR R0,[R1,#-0x09F0]
    *AT91C_PIOB_SODR = (1 << 0); // Ukljuci LED
    0x00000160 STR R0,[R1,#-0x09D0]
}
0x00000164 BX R14 ; skok na 0x0000 0000

```

- uočiti
 - postoji
 - kod koji je nastao prevođenjem inicijalizacijskog programa
 - kod koji je nastao prevođenjem C programa
 - nema
 - nikakvog dodatnog koda, npr.
 - koda za inicijalizaciju datoteka
 - sistemskih poziva za izlaz
 - nakon "izlaska" iz programa izvršavanje se nastavlja na adresi 0x0000 0000 !!!
 ⇒ potreban je **while(1);** na kraju funkcije **glavni()**

7.4 Okolina za rad s bibliotekama

Primjer

Kako izgleda kod koji nastaje nakon prevodenja i povezivanja programskih odsječaka iz prethodnog primjera, ako se funkcija zove **main()**?

```
#include <at91sam7x128.h>

int main(void) {
    *AT91C_PMC_PCER = (1 << 0); // Takt za PIOB
    *AT91C_PIOB_PER = (1 << 0); // PIO Enable
    *AT91C_PIOB_OER = (1 << 0); // Output Enable
    *AT91C_PIOB_SODR = (1 << 0); // Uključi LED
}
```

Rješenje

- **main()** je korijenska razina korisnikove aplikacije
- da bi se moglo koristiti biblioteke
 - ⇒ trazi da se okolina u kojoj se program izvršava bude inicijalizirana
 - ⇒ "ispod" funkcije **main()** nalaze se inicijalizacije

- ulaz u program je **__main** (uočiti dvije crtice ispred **main**)
⇒ kod za inicijalizaciju završava odsječkom

```
IMPORT  __main
LDR    R0, =__main
BX
```

- programski odsječak **__main** sadrži
 - poziv potprograma za učitavanja (*load*)
 - poziv ljudske koja radi s bibliotekama
- potprogram za učitavanja
 - kodni odsječak počinje na **__scatterload_rt2** (ili slično, ovisni o verziji alata)
 - u toku povezivanja dijelovi koda mogli su biti pohranjeni kao komprimirani (opcija *linkera* koju ovdje nećemo detaljno obrađivati)
 - potprogram za učitavanje čini sljedeće
 - dekomprimira komprimirane dijelove
 - učitava ih u odgovarajući dio memorije
 - inicijalizira dijelove memorije koji moraju imati vrijednost 0
- ljudska koja radi s bibliotekama
 - kodni odsječak počinje na **__rt_entry_sh** (ili slično)
 - kodni odsječak na **__rt_entry_sh** radi sljedeće
 - inicijalizira parametre stoga i *heap-a*
⇒ **__user_setup_stackheap** i
__user_initial_stackheap
 - inicijalizira okolinu za rad s bibliotekom
⇒ **__rt_lib_init**
 - poziva **main()**
 - poziva funkciju za izlaz iz programa
⇒ **exit()**
- funkcija za izlaz iz programa, **exit()**, između ostalog poziva
 - funkciju za "gašenje" biblioteka
⇒ **__rt_lib_shutdown**
 - funkciju za izlaz
⇒ **__sys_exit**

- rezultat prevođenja i povezivanja

- potprogrami su poredani po redoslijedu izvršavanja, a ne po mjestu gdje se nalaze u memoriji

- zbog jednostavnosti prikazana je samo struktura programa a ne sve naredbe

```

; ****
;          STARTUP KOD
; ****

; pocetak STARTUP koda

    AREA RESET, CODE, READONLY ; odsjecak RESET
        ARM                      ; ARM skup instr.

; Inicijalizacija prekidnih vektora

    0x00000000 LDR PC, [PC,#0x0018] ; PC=0x0000 0058
    0x00000004 LDR PC, [PC,#0x0018]
    ;      ...
; Inicijalizacija sklopolija za reset

    0x00000058 LDR R0, [PC,#0x00C8]
    0x0000005C LDR R1, [PC,#0x00C8]
    0x00000060 STR R1, [R0,#0x0008]
    ;      ...
; Inicijalizacija memoriskog kontrolera
; Inicijalizacija Watch-dog sklopa
; Inicijalizacija izvora takta
; Inicijalizacija pokazivaca stoga i
; onemogućavanje prekida u pojedinim nacinima rada
; Ulaz u C kod

    IMPORT __main
    LDR R0, =__main
    0x00000120 LDR R0, [PC,#0x0020]
    BX R0
    0x00000124 BX R0
    0x00000128 DD 0xFFFFFD00
    ;      ...
    0x00000148 DD 0x0000014C

; ****
;          ULAZ U PROGRAM
; ****

    __main:
    0x0000014C BL __scatterload_rt2(0x00000154)
    0x00000150 BL __rt_entry_sh(0x000001EC)

```

```
; ****
;          POTPROGRAM ZA UCITAVANJE KODA
; ****
; dekomprimiranj dijelova koda i učitavanje u memoriju
; inicijaliziranje ZI dijelova memorije

        __scatterload_rt2:

0x00000154 ADD R0,PC,#0x0000002C
0x00000158 LDMIA R0,{R10-R11}
;      ...

; ****
;          LJUSKA KOJA RADI S BIBLIOTEKAMA
; ****

; Ulaz u ljudsku

        __rt_entry_sh:

0x000001EC BL  __user_setup_stackheap(0x00000274)
0x000001F0 MOV R1,R2
        __rt_entry_postsh_1:
0x000001F4 BL  $Ven$AT$I$$__rt_lib_init(0x000001BC)

        $Ven$AT$I$$__rt_lib_init:
0x000001BC ADD R12,PC,#0x00000001
0x000001C0 BX  R12

; Inicijalizacija okoline za rad s bibliotekama

        __rt_lib_init:
0x000001C4 PUSH {R0-R4,LR}
0x000001C6 BL  $Ven$TA$I$$fp_init(0x00000308) - Part #1
0x000001C8 BL  $Ven$TA$I$$fp_init(0x00000308) - Part #2

        __rt_lib_init_user_alloc_1:
0x000001CA LDR R0,[SP,#0x0014]
;      ...
0x000001D4 BX  LR

        __rt_entry_postli_1:
0x000001F8 LDR R12,[PC,#0x001C]
;      ...
0x0000020C BX  R12 ; Poziv funkcije main
;      ...
0x00000218 BL  exit(0x000002D4) ; Izlaz
```

```

; *****
;      NEKI POTPROGRAMI KOJE POZIVA LJUSKA
; *****

; Inicijalizacija stoga

        __user_setup_stackheap:

0x00000274 MOV      R5,R14
;
;      ...
0x0000029C BL      __user_initial_stackheap(0x0000022C)
;
;      ...
0x000002D0 E12FFF1E BX      R14

        __user_initial_stackheap
        LDR      R0, = Heap_Mem
0x0000022C LDR R0,[PC,#0x000C]
        LDR R1, =(Stack_Mem + USR_Stack_Size)
0x00000230 LDR R1,[PC,#0x000C]
        LDR R2, =(Heap_Mem +      Heap_Size)
0x00000234 LDR R2,[PC,#0x0004]
        LDR R3, = Stack_Mem
0x00000238 LDR R3,[PC,#0x0008]
        BX      LR
0x0000023C BX      R14
0x00000240 DD      0x00200060
0x00000244 DD      0x00200460
0x00000248 DD      0x00200060

; Funkcija main

int main(void) {
    *AT91C_PMC_PCER = (1 << 0); // Takt za PIOB
0x0000024C MOV      R0,#0x00000001
0x00000250 MOV      R1,#0x00000000
0x00000254 STR      R0,[R1,-0x03F0]
    *AT91C_PIOB_PER = (1 << 0); // PIO Enable
0x00000258 STR      R0,[R1,-0x0A00]
    *AT91C_PIOB_OER = (1 << 0); // Output Enable
0x0000025C STR      R0,[R1,-0x09F0]
    *AT91C_PIOB_SODR = (1 << 0); // Ukljuci LED
0x00000260 STR      R0,[R1,-0x09D0]
}
0x00000264 MOV      R0,#0x00000000
0x00000268 BX      R14

; Izlaz

        exit:
0x000002D4 LSL      R4,R0,#0
;
;      ...
0x000002DE BL      $Ven$TA$I$$__rt_exit(0x00000220)
;
;      ...

```

```

$ven$TA$I$$__rt_exit:
0x00000220 BX PC
0x00000222 NOP
          __rt_exit_prels_1:
0x00000224 BL $Ven$AT$I$$__rt_lib_shutdown(0x000001D8)

          __rt_exit_exit:
0x00000228 BL __sys_exit(0x000002EC)

          __sys_exit:
0x000002EC LDR R1,[PC,#0x000C]
0x000002F0 MOV R0,#0x00000018
0x000002F4 SWI 0x00123456
0x000002F8 BX R14
0x000002FC DD 0x00000008
0x00000300 DD 0x00020026

; Deinicijalizacija okoline za rad s bibliotekama

$ven$AT$I$$__rt_lib_shutdown:
0x000001D8 ADD R12,PC,#0x00000001
0x000001DC BX R12
          __rt_lib_shutdown:
0x000001E0 PUSH {R4-R5}
0x000001E2 MOV R5,LR
          __rt_lib_shutdown_user_alloc_1:
0x000001E4 MOV LR,R5
0x000001E6 POP {R4-R5}
0x000001E8 BX LR
0x000001EA LSL R0,R0,#0

```

- uočiti
 - postoji
 - kod koji je nastao prevodenjem inicijalizacijskog programa
 - kod koji je nastao prevodenjem C programa
 - kod za inicijalizaciju datoteka
 - sistemskih pozivi za izlaz
- napomena
 - startup kod završava skokom na __main
 - ⇒ linker će staviti __main i sve što slijedi
 - ako se u startup stavi main umjesto __main, skok će ići na main
 - ⇒ linker će svejedno staviti __main, ali se taj dio koda neće koristiti i neće biti inicijalizirana okolina koja treba za rad s bibliotekama

- pogledajmo opet funkciju `_sys_exit`

```

_sys_exit:
0x000002EC LDR R1, [PC,#0x000C]
0x000002F0 MOV R0, #0x00000018
0x000002F4 SWI 0x00123456
0x000002F8 BX R14
0x000002FC DD 0x00000008
0x00000300 DD 0x00020026
```

- funkcija sadrži instrukciju SWI
- SWI broj je 0x00123456
 - ⇒ traži posluživanje agenta
 - ⇒ ova funkcija predviđena je za rad u semihost okolini
- da bi program mogao raditi na sklopolju, ovu funkciju je potrebno napisati ponovno (*retargeting*)

- pogledajmo opet funkciju `_user_initial_stackheap`

```

_user_initial_stackheap 5
LDR R0, = Heap_Mem
0x0000022C LDR R0, [PC,#0x000C]
          LDR R1, = (Stack_Mem + USR_Stack_Size)
0x00000230 LDR R1, [PC,#0x000C]
          LDR R2, = (Heap_Mem +     Heap_Size)
0x00000234 LDR R2, [PC,#0x0004]
          LDR R3, = Stack_Mem
0x00000238 LDR R3, [PC,#0x0008]
          BX LR
0x0000023C BX R14
0x00000240 DD 0x00200060
0x00000244 DD 0x00200460
0x00000248 DD 0x00200060
```

- ova funkcija je već napisana u okviru startup koda
- njoj odgovarajuća funkcija koja postoji u biblioteci, a također traži posluživanje agenta
- uočiti
 - biblioteku je potrebno oblikovati da bi odgovarala aplikaciji
 - ⇒ *library tayloring*
 - oblikovanje je obavezno u slučaju kad funkcije moraju raditi na sklopolju

7.5 Pisanje programa koji se izvršava na sklopolju

7.5.1 Oblikovanje funkcija za rad s vanjskim sklopoljem

- ako funkcija radi na stvarnom sklopolju, 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)**
- funkcije koje NEPOSREDNO ovise o *semihostu* (ARM)
 - funkcije za oblikovanje memorijskog modela
`__user_initial_stackheap`
 - funkcije za obradu grešaka i izlaz
`_sys_exit, _ttywrch`
 - ulazno-izlazne funkcije
`_sys_open, _sys_close, _sys_read, _sys_write,`
`_sys_iserror, _sys_ensure, _sys_command_string,`
`_sys_flen, _sys_seek, _sys_istty, _sys_tmpnam`
 - ostale funkcije
`clock(), _clock_init(), remove(), rename(),`
`system(), time()`
- funkcije i deklaracije koje POSREDNO ovise o *semihostu* (ARM)
(pozivaju neku od gornjih funkcija)
 - funkcije koje služe za rad s iznimkama
`__raise(), __default_signal_handler()`
 - funkcije koje se koriste u alociranju memorije
`__Heap_Initialize()`
 - funkcije koje poziva *printf* familija
`fputc(), ferror(), __stdout`
 - funkcije koje poziva *scanf* familija
`fgetc(), __backspace(), __stdin`
 - funkcije koje poziva familija ulazno izlaznih funkcija koje rade s nizovima (*streaming*)
 - `fwrite(), fread(), puts(), gets(), fputs(),`
 - `fgets(), ferror()`

7.5.2 Program "Pozdrav svijete!"

Primjer

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

Rješenje

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

7.5.2.1 Funkcija main()

- programski odsječak

```
#include <stdio.h>
#include <at91sam7x128.h>

extern void init_USART0 (void);

int main (void) {

    init_USART0(); // Vidi daljnji tekst

    printf ("Pozdrav svijete!\n\r");

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

- uočiti
 - koristi se funkcija **printf()** i **main()**
⇒ potrebno je napisati niže funkcije koji koristi **printf()** i sistemske pozive potrebne sa rad funkcije **main()**

7.5.2.2 Redefinicija ulazno-izlaznih funkcija

- pregled nižih funkcija koje koriste neke ulazno-izlazne funkcije
(ukupno ih ima 25)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---------|---|---|---|---|---|---|---|---|---|----|
| printf | X | - | X | X | X | - | - | - | - | - |
| fprintf | X | - | - | X | X | - | - | - | - | - |
| scanf | X | X | - | - | - | X | - | - | X | - |
| fscanf | X | - | - | - | - | X | - | - | X | - |
| putchar | X | - | X | X | - | - | - | - | - | - |
| getchar | X | X | - | - | - | X | - | - | - | - |
| puts | X | - | X | X | - | - | - | - | - | - |
| gets | X | X | - | - | X | X | - | - | - | - |
| fread | X | - | - | - | - | X | - | - | - | - |
| fwrite | X | - | - | X | - | - | - | - | - | - |
| itd. | | | | | | | | | | |

| | |
|---|----------|
| 1 | __FILE |
| 2 | __stdin |
| 3 | __stdout |
| 4 | fputc() |

| | |
|---|----------|
| 5 | ferror() |
| 6 | fgetc() |
| 7 | fgetwc() |
| 8 | fputwc() |

| | |
|----|----------------|
| 9 | __backspace() |
| 10 | _backspacewc() |
| | |
| | |

- potrebno je redefinirati sve funkcije koje neposredno ovise o semihostu
 - koristi se funkcija **printf()**
⇒ potrebno je redefinirati
__FILE, __stdout, fputc(), ferror()

- deklaracija struktura **__FILE** i **_stdout**
 - u **stdio.h** postoji deklaracija tipa

```
typedef struct __FILE FILE;
extern FILE __stdin, __stdout, __stderr;
```
 - **FILE** je struktura koji sadrži parametre potrebne za kontrolu prijenosa podataka na željenu poziciju
 - u **stdio.h** ne postoji deklaracija strukture **__FILE**
⇒ struktura se deklarira u skladu s potrebom korisnika
- funkcija **fputc()**

```
int fputc (int c, FILE * f );
```

 - upisuje karakter **c** u **f**
 - vraća upisani karakter
- funkcija **ferror()**

```
int ferror (FILE * f );
```

 - ispituje upis ili čitanje u **f** i u slučaju greške vraća broj različit od 0

7.5.2.3 Redefinicija sistemskih poziva

- potrebno je redefinirati sve funkcije koje neposredno ovise o *semihostu*
 - eventualni izlaz iz funkcije **main()** (vidi poglavljje 7.4)
⇒ potrebno je redefinirati funkciju
_sys_exit()
- napomena
 - kad se uključi simbol **__use_no_semihosting_swift**, linker će upozoriti na neke funkcije koje nisu redefinirane

7.5.2.4 Datoteka *retarget.c*

- sadrži redefinicije funkcija

```
#include <stdio.h>

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

extern void sendchar_USART0(int ch);

struct __FILE {
    int handle; // Ovdje se može nalaziti još polja
};

FILE __stdout;

int fputc(int ch, FILE *f) {
    sendchar_USART0(ch); // Korisnikova funkcija
    return (ch);
}

int ferror(FILE *f) {
    // Implementacija po zelji korisnika
    return 0;
}

void _sys_exit(int return_code) {
    while(1); // Korisnikova implementacija
}
```

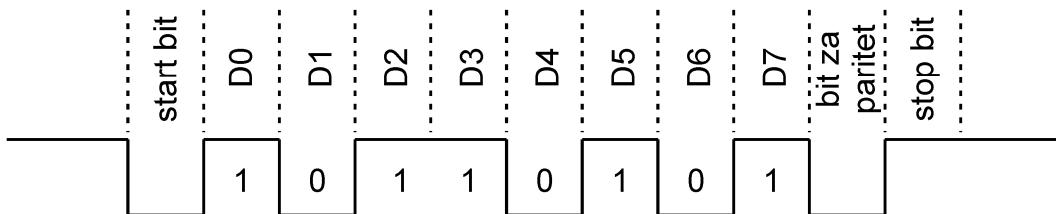
- uočiti
 - u ovom slučaju koristi se samo jedna izlazna jedinica
⇒FILE u **fputc()** se ne koristi, ali
 - FILE mora postojati u pozivu zbog kompatibilnosti
 - zbog **while(1)** u funkciji **main()**, do **sys_exit()** se ne dolazi
 - **while(1)** bi se u ovom slučaju mogao izostaviti pa bi program završio u petlji u funkciji **_sys_exit()**
- napomena
 - običaj je da se ova datoteka naziva **retarget.c**, ali to nije nužno
- u ovoj fazi potrebno je još implementirati funkcije


```
init_USART0()
sendchar_USART0()
```

 - ove funkcije rade s konfiguracijskim registrima sklopljuju !!!

7.5.2.5 Serijsko sučelje

- mikrokontrolери familije AT91SAM7X imaju
 - 2 USART sučelja
 - 1 DBGU sučelje (korišteno je u poglavlju 6.4.2)
- 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*)
 - sklopolje za slanje i primanje podataka
- prijenos preko RS232 sučelja
⇒ vidi predmet Ugradbeni računalni sustavi



- USART sklopa sadrže slijedeće registre
 - registri za inicijalizaciju rada sklopovlja
 - registar za upravljanje generatorom takta za prijenos **US_BRGR (USART Baud Rate Generator Register)**
 - registar za odabir načina rada **US_MR (USART Mode Register)**
 - registar za upravljanje radom sklopovlja (uključivanje i isključivanje RX i TX i dr.) **US_CR (USART Control Register)**
 - registri koji sadrže status sklop (je li podatak primljen ili poslan) **US_CSR (USART Channel Status Register)**
 - podatkovne registre
 - registar koji sadrži primljeni podak **US_RHR (USART Receive Holding Register)**
 - registar u koji se upisuje podatak koji se šalje **US_THR (USART Transmit Holding Register)**
 - registri za nadzor slanja i primanja **US_RTOR (USART Receiver Time-out Register)** **US_TTGR (USART Transmitter Timeguard Register)**
 - registre za upravljanje prekidima **US_IER (USART Interrupt Enable Register)** **US_IDR (USART Interrupt Disable Register)** **US_IMR (USART Interrupt Mask Register)**
 - specijalni registri za IrDA i ISO7816 načine rada **US_FIDI**, **US_NER**, **US_IF**

7.5.2.6 Funkcije koje rade sa serijskim sučeljem

- da bi USART sklop ispravno radio, **u našem primjeru** potrebno je napraviti sljedeće inicijalizacije
 - uključiti radni takt za USART0 i PIOA
 - vidi poglavlje 6.6 u kojem je opisan primjer omogućavanja takta za PIOB
 - konfigurirati PIOA tako da linija TXD0 bude spojena na vanjski priključak mikrokontrolera
 - vidi poglavlje 6.6 u kojem je opisan način konfiguracije priključaka PIOB
 - resetirati odašiljač
 - odabratи način rada USART sklopa
 - konfigurirati sklopovlje generatora takta za prijenos
 - omogućiti rad odašiljača
- programski odsječak za inicijalizaciju

```
#include <at91sam7x128.h>

#define MCK 48000000 // fMCK=48MHz
#define BR 9600 // Zeljena brzina prijenosa
#define CD (MCK/16/BR) // konst. za generator takta za prijenos

void init_USART0 (void) {
    *AT91C_PMC_PCER=(1<<AT91C_ID_US0) | // Takt za USART0
    (1<<AT91C_ID_PIOA); // Takt za PIOA

    *AT91C_PIOA_PDR=AT91C_PA1_TXD0; // Dodjela prikljucaka

    *AT91C_US0_CR=AT91C_US_RXDIS | // Unemoguci odasiljac
    AT91C_US_RSTTX; // Resetiraj odasiljac

    *AT91C_US0_MR=AT91C_US_USMODE_NORMAL | // normalan nacin rada
    AT91C_US_CLKS_CLOCK | // takt za USART je MCK
    AT91C_US_CHRL_8_BITS | // znak ima 8 bitova
    AT91C_US_NBSTOP_1_BIT | // 1 stop bit
    AT91C_US_PAR_NONE; // nema pariteta
    // MSBF=0 => prvo ide LSB

    *AT91C_US0_BRGR=CD; // inic. generatora takta za prijenos
    *AT91C_US0_CR=AT91C_US_TXEN; // enable odasiljac
}
```

- programski odsječak za ispis znaka na seriju obavlja slijedeće operacije
 - čeka u petlji da ode prethodni znak
 - upisuje znak u registar za slanje
- programski odsječak za ispis znaka na seriju

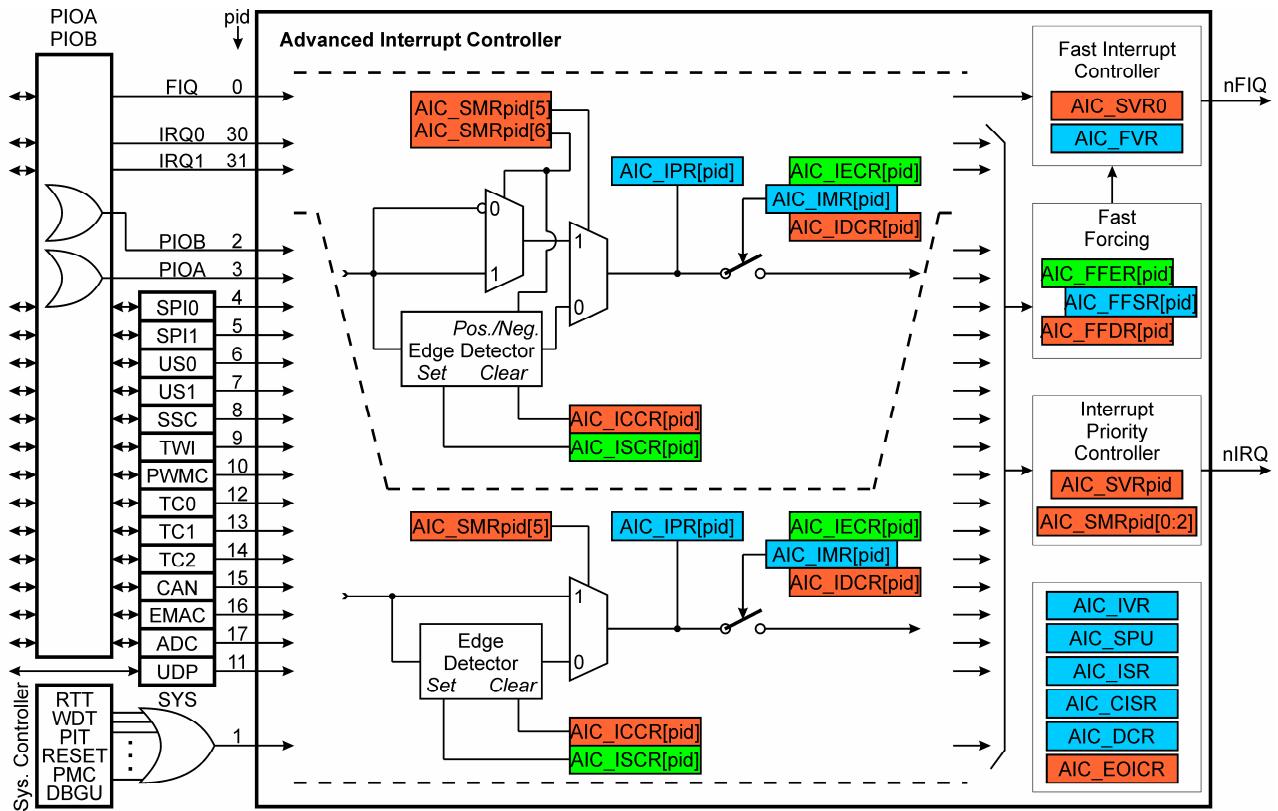
```
#include <at91sam7x128.h>

void sendchar_USART0 (int ch) {
    while (!(*AT91C_US0_CSR & AT91C_US_TXRDY));
        // AT91C_US0_CSR => USART Channel Status Register
        // AT91C_US_TXRDY=(0x1<<1) => TXRDY Interrupt
    *AT91C_US0_THR = ch; // Transmitter Holding Register
}
```

- paziti
 - ako program radi u simulatoru, ponekad (čitaj: nikad iz prvog pokušaja) ne radi na sklopoljju
 - primjer
 - simulator u prozoru za USART ispisuje poruku koju šalje funkcija `printf()`, BEZ OBZIRA je li brzina prijenosa dobro definirana!
- za vježbu je potrebno
 - uz pomoć podataka proizvođača mikrokotrolera (*Datasheets*) proučiti inicijalizaciju USART sklopoljja
 - uhodati program *Pozdrav svijete!* u simulatoru

7.6 Zahtjevi za prekid koje daje vanjsko sklopolje

- pojednostavljena blokovska shema prekidnog sustava



- uočiti
 - ulazno sklopolje za obradu prekida razlikuje se za
 - prekid koji dolazi izravno s vanjskog priključka μ C
 - prekid koji daje ugrađena periferija
 - vanjski zahtjev FIQ usmjerava se na nFIQ mikrokontrolera
 - svi ostali zahtjevi
 - usmjeravaju se na nIRQ mikrokontrolera
 - mogu se usmjeriti na nFIQ
 - \Rightarrow AIC_FFER[pid], AIC_FFER[pid]
- prekidi se obrađuju ovisno o prioritetu
 - postoji 8 prioriteta prekida (0 je najniži a 7 je najviši)
 - određeni su poljem PRIOR u AIC_SMRpid
- prekide je moguće onemogućiti na razini kontrolera prekida (AIC-a)
 - \Rightarrow upis jedinice u polja NIRQ i NFIQ u AIC_CISR

- registri za upravljanje prekidima

SMR0-SMR31 (*Source Mode Register*)

| 31-7 | 6-5 | 4-3 | 2-0 |
|------|----------------|-----|--------------|
| - | SRCTYPE | - | PRIOR |

IPR (*Interrupt Pending Register*)

IECR/IDRC, IMR (*Interrupt Enable/Disable Command Register, Interrupt Mode Register*)

ISCR/ICCR (*Interrupt Set/Clear Command Register*)

SVR0-SVR31 (*Source Vector Register*)

IVR (*Interrupt Vector Register*)

SPU (*Spurious Interrupt Vector Register*)

ISR (*Interrupt Status Register*)

| 31-5 | 4-0 |
|------|--------------|
| - | IRQID |

CISR (*Core Interrupt Status Register*)

| 31-2 | 1 | 0 |
|------|-------------|-------------|
| - | NIRQ | NFIQ |

EOICR (*End of Interrupt Command Register*)

FFER/FFDR/FFSR (*Fast Forcing Enable/Disable/Status Register*)

FVR (*FIQ Interrupt Vector Register*)

DCR (*Debug Control Register*)

| 31-2 | 1 | 0 |
|------|-------------|-------------|
| - | GMSK | PROT |

- stanje svih registara nakon reseta je 0x0000 0000

• 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
- popis identifikator **pid** dan je u tablici u poglavlju 6.6
- obrada prekida koji zatraži System Controller (SYS)
 - uočiti
 - 6 izvora prekida usmjereni 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

7.7 Primjer prekidnog programa

Primjer

Potrebno je napisati programsku podršku za obradu vanjskog prekida IRQ0. Prekidna funkcija mora brojiti prekide, te pohraniti njihov ukupan broj u globalnu varijablu.

Rješenje

- programska podrška mora osigurati
 - na razini osnovnih inicijalizacija
 - omogućavanje prekida razini procesorske jezgre
 - čitanje prekidnog vektora iz prekidnog kontrolera (AIC-a)
 - na aplikacijskoj razini
 - inicijalizaciju prekidnog sklopolja
 - obradu zahtjeva za prekid
 - obradu neregularnog prekida

7.7.1 Čitanje prekidnog vektora

- u našem slučaju prekid ćemo usmjeriti na nIRQ mikrokontrolera
- čitanje prekidnog vektora potrebno je osigurati u sklopu inicijalizacija

```

;      .
0xFFFFF000 ; BASE_AIC
;      .
;      .
→ 0xFFFFF100 ; AIC_IVR
;      .
;      .
0xFFFF FFFF ;      .           ; kraj mem. prostora

0x0000 0000 LDR PC,Reset_Addr ; pocetak mem. prostora
0x0000 0004 LDR PC,Undef_Addr
0x0000 0008 LDR PC,SWI_Addr
0x0000 000C LDR PC,PAbt_Addr
0x0000 0010 LDR PC,DAbt_Addr
0x0000 0014 NOP
→ 0x0000 0018 LDR PC,[PC,#-0xF20] ; citanje iz AIC_IVR
0x0000 001C LDR PC,FIQ_Addr
;      .
; ostale inicijalizacije iste su kao u
; poglavljju 6.5

```

- uočiti instrukciju

LDR PC, [PC,#-0xF20]

- u trenutku oduzimanja **PC, #-0xF20** je **PC=0x0000 001C**
 \Rightarrow čita se prekidni vektor s lokacije **0xFFFFF100**

7.7.2 Omogućavanje prekida na razini procesorske jezgre

- uočiti
 - linije nIRQ i nFIQ iz AIC ulaze u procesor
⇒ mora biti omogućen prekid na razini procesora
- u našem slučaju koristi ćemo nIRQ pa mora biti
 $CPSR \Rightarrow I=0$

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

- primjer inicijalizacije (vidi poglavlje 6.5.6)

```

; definicije nacina rada

Mode_USR      EQU      0x10
Mode_FIQ      EQU      0x11
Mode_IRQ      EQU      0x12
Mode_SVC      EQU      0x13
Mode_ABT      EQU      0x17
Mode_UND      EQU      0x1B
Mode_SYS      EQU      0x1F

; Definicije bitova za onemogucavanje prekida

I_Bit         EQU      0x80
F_Bit         EQU      0x40

; Inicijalizacija SP u nacinu rada Undefined

        MSR CPSR_c, #Mode_UND:OR:I_Bit:OR:F_Bit
; .
;

; Inicijalizacija SP u nacinu rada Abort

        MSR CPSR_c, #Mode_ABT:OR:I_Bit:OR:F_Bit
; .
;

; Inicijalizacija SP u nacinu rada User

; U ovom slucaju omogucen je samo IRQ
        MSR CPSR_c, #Mode_USR:OR:F_Bit

; .
;
;
```

- 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;
}
```

- kod u asembleru

```
void irq0_handler (void) __irq {
    STMDB R13!, {R0-R1} ; pohrana sadržaja reg.
    brojac_prekida++;
    LDR   R0, [PC, #0x0048]
    LDR   R0, [R0]
    ADD   R0, R0, #0x00000001
    LDR   R1, [PC, #0x003C]
    STR   R0, [R1]
    *AT91C_AIC_EOICR = 0;
    MOV   R0, #0x00000000
    STR   R0, [R0, #-0xED0]
}
LDMIA R13!, {R0-R1} ; povrat sadržaja reg.
SUBS  PC, R14, #0x00000004
```

- uočiti

- pohranu sadržaja registara koji će se koristiti u funkciji
- prekid je uzrokovao prijelaz u IRQ način rada
- u IRQ načinu rada postavljeno je I=1
- ako se želi omogućiti gniježđenje prekida mora se u prekidnoj funkciji postaviti I=0
 - ⇒ za detalje vidi [1] *AT91 ARM Thumb-based Microcontrollers*, Datasheets, Atmel 2007

7.8 Unutrašnji izvori prekida

- zahtjevi za prekid
 - vanjski
 - od vanjskog sklopolja
 - nIRQ i nFIQ ulazi procesora
 - pogoni ih AIC
 - ⇒ do 32 izvora prekida
 - unutrašnji
 - od unutrašnjeg sklopolja
 - zahtjevi za prekid koji su posljedica neregularnog rada
 - ⇒ nedefinirana instrukcija (*Undefined instruction*)
 - ⇒ nedefinirana adresa instrukcije (*Prefetch abort*)
 - ⇒ nedefinirana adresa podatka (*Data abort*)
 - programskih zahtjevi za prekid
 - instrukcija **SWI**
- ARM7TDMI
 - prekidni vektori

| Iznimka | Vektor | Način rada |
|---------------------------------|-------------|-----------------------|
| Reset | 0x0000 0000 | Supervisor |
| Nedefinirana instrukcija | 0x0000 0004 | <i>Undefined</i> |
| Programski prekid | 0x0000 0008 | Supervisor |
| Nedefinirana adresa instrukcije | 0x0000 000C | <i>Abort</i> |
| Nedefinirana adresa podatka | 0x0000 0010 | <i>Abort</i> |
| IRQ | 0x0000 0018 | <i>Interrupt</i> |
| FIQ | 0x0000 001C | <i>Fast Interrupt</i> |

- obrada u zahtjeva za prekid koji daje vanjsko sklopolje obrađena je u poglavljima 7.6 i 7.7 i u okviru laboratorijskih vježbi
 - kontroler prekida (AIC)
 - pisanje prekidnih funkcija
 - prevođenje i izvođenje prekidnih funkcija
- u dalnjem tekstu bit će obrađeno
 - obrada programskih zahtjeva za prekid
 - primjer agenta za semihostingu

7.9 Programski prekid

- programski prekid posljedica je izvođenja instrukcije
 - ARM skup instrukcija

`SWI{<cond>} <immed_24>`

- Thumb skup instrukcija

`SWI <immed_8>`

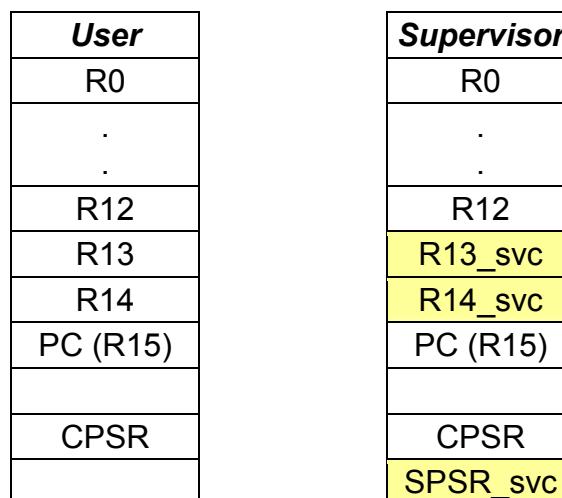
- primjer

```
SWI #120340 ; ARM skup instrukcija
SWI #10       ; Thumb skup instrukcija
```

- izvršavanje instrukcije SWI

⇒ ulazak u *Supervisor mode* (SVC)

```
R14_svc      = adresa instrukcije koja slijede instrukciju SWI
SPSR_svc    = CPSR
CPSR[4:0]    = 0b10011 // ulazak Supervisor nacin rada
CPSR[5]      = 0        // prijelaz na ARM skup instrukcija
CPSR[7]      = 1        // onemogućavanje IRQ
PC          = 0x00000008
```



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

- uočiti
 - promijenjen je način rada u *Supervisor*
 - u SWI (također i u ostale prekide) ulazi se u ARM načinu rada
 - automatski je zabranjen IRQ, ali ne i FIQ
 - postoji **samo jedan SPSR** za svaki način rada
 - ⇒ u slučaju gniježđenja prekida potrebno je na pohraniti i kasnije vratiti njegov sadržaj
- povratak iz *Supervisor* načina rada
 - ⇒ instrukcija koja učitava registra R15 (PC) iz R14_svc
 - primjer


```
MOV S PC, R14_svc ; S označava povrat CPSR
```

 - ⇒ ova instrukcija radi slijedeće

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

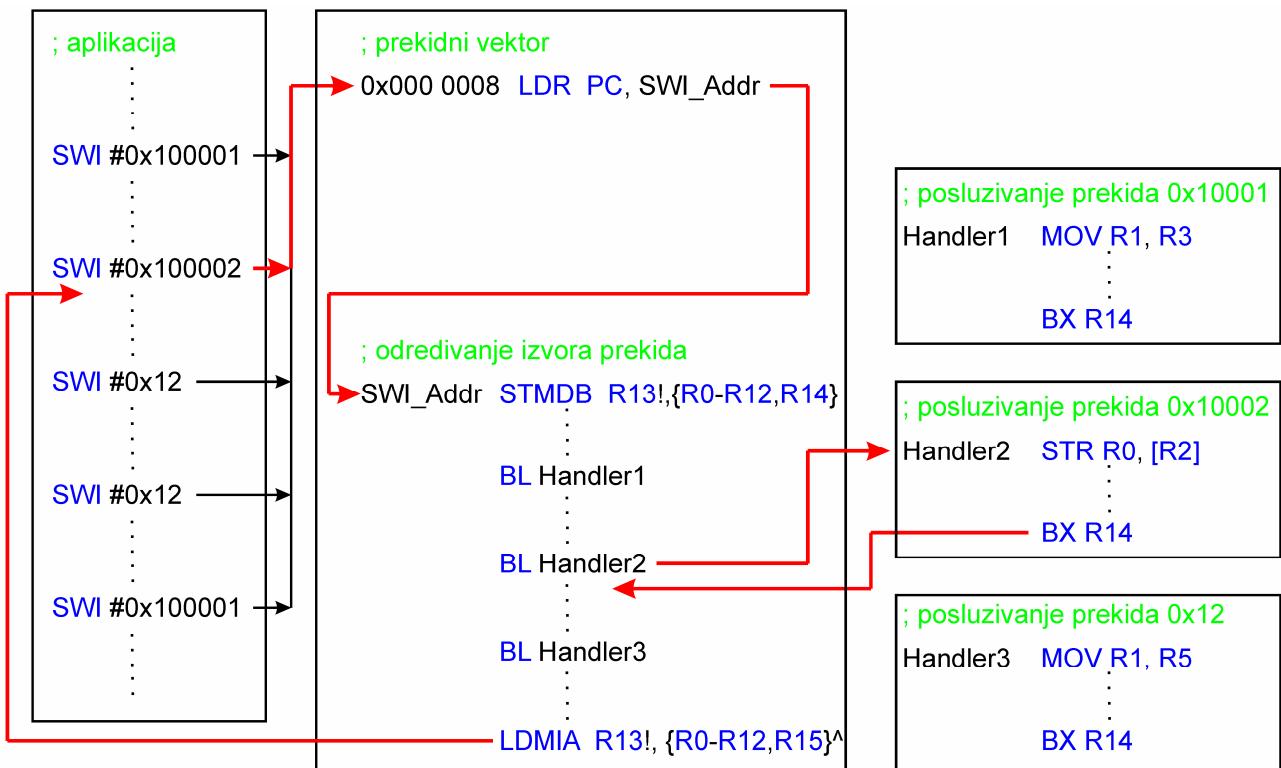
- razlika u posluživanju sklopoških i programskeh prekida
 - sklopoški prekidi
 - izaziva ih signal
 - točka u kojoj je prekinuto izvođenje programa je nepoznata
 - ⇒ odabir posluživanja izvodi se pomoću prekidnog vekora
 - ⇒ argumenti se mogu prenositi jedino preko statičkih (globalnih) varijabli
 - uočiti: registri (npr. u AIC) su globalne varijable (samim tim su i statičke)
 - programski prekidi
 - prekid je izazvan instrukcijom
 - 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

- čemu služi parametar instrukcije SWI

SWI{<cond>} <immed_24>

- procesor ignorira ovaj parametar
- ipak, čitanjem parametra može se odrediti tko je generirao SWI
- Kako to odrediti?
⇒ čitanjem operacijskog koda instrukcije, pomoću registra LR

- SWI poslužuje veći broj zahtjeva



- napomene

- slično ponašanje kao SWI ima i instrukcija BKPT (*Breakpoint*) koja se sreće u novijim ARM arhitekturama
⇒ BKPT izaziva programski prekid koji koristi prekidni vektor *nedefinirane adrese instrukcije* (*Prefetch abort*)
- u nekim arhitekturama mnemonik SWI (*SoftWare Interrupt*) je zamijenjen sa SVC (*SuperVisory Call*)

- SWI odnosno SVC može imati argumente

⇒ potrebno je poznavati mehanizam prijenosa argumenata u funkcije

7.10 Tipovi podataka

- podsjetimo se
 - standard koji opisuje jezik C, **ne definira eksplicitno** veličine pojedinih tipova podataka
 - veličine ovise o platformi na kojoj se implementiraju
 - primjer
 - veličina podataka tipa **int** na dvije različite platforme

| | C51 | ARM |
|-----------------|-----|-----|
| int | 16 | 32 |
| long int | 32 | 32 |
- podsjetimo se:
 - 8051 je 8-bitni, a ARM 32-bitni procesor
 - obično veličina tipa **int** odgovara izvornoj širini podatkovne sabirnice odnosno aritmetičke jedinice procesora
 - to ne vrijedi za 8 bitne procesore
- mnogi programeri koji godinama koriste jednu platformu rade greške kad prijeđu na drugu platformu
 - ⇒ problemi su posebno izraženi kod prenošenja izvornog koda s jedne platforme na drugu
- programeri (pogotovo početnici) ponekad deklariraju varijable po principu "bolje veće nego razmišljati"
 - kod pisanja aplikacijskih programa, to je u redu do razine poziva funkcija iz biblioteke
 - ⇒ kod funkcija iz biblioteke sučelja su već definirana
 - kod ugradbenih sustava **količina memorije je često ograničena**
 - ⇒ razmisliti prije deklaracije tipa koji je veći od izvorne širine riječi procesora
 - kod pisanja **programske podrške niske razine potrebno je paziti na stvarne veličine podataka**
 - ⇒ posebno je to važno kod posluživanja SWI
 - ⇒ vidi daljnji tekst

- širine podataka u našem slučaju (ARM jezgra i *RealView Compiler*)

| Tip | Širina u bitovima | Širina u bajtovima | Poravnavanje |
|-------------|-------------------|--------------------|-----------------|
| char | 8 | 1 | na bajt |
| short | 16 | 2 | na poluriječ |
| int | 32 | 4 | na riječ |
| long | 32 | 4 | na riječ |
| long long | 64 | 8 | na dvije riječi |
| float | 32 | 4 | na riječ |
| double | 64 | 8 | na dvije riječi |
| long double | 64 | 8 | na dvije riječi |
| pointer | 32 | 4 | na riječ |
| _Bool | 8 | 1 | na bajt |

- uočiti
 - podaci su poravnati u memoriji
 - **poravnavanje opisuje položaj podatka u memoriji**
 - poravnavanje na bajt
⇒ podatak se može nalaziti na bilo kojoj adresi
 - poravnavanje na poluriječ
⇒ podatak se nalazi na parnoj adresi
 - poravnavanje na riječ
⇒ podatak se nalazi na adresi koja je višekratnik broja 4
 - poravnavanje na dvije riječi
⇒ podatak se nalazi na adresi koja je višekratnik broja 8
 - **int** i **long** su jednako veliki
double i **long double** su jednako veliki
 - pokazivači su 32-bitni
⇒ dovoljno za adresiranje cijelog memorijskog prostora

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

Primjer

Analizirati kako se prenose argumenti u zadane funkcije, te kako se iz funkcija vraća rezultat.

```
// y=a+b+c-d+e
//   f_0 => a+(b+(c-d))+e
//   f_1 =>   (b+(c-d))
//   f_2 =>       (c-d)

int f_0(int a, int b, int c, int d, int e) {
    int y;
    y=a+f_1(b,c,d)+e;
    return y;
}

int f_1(int b, int c, int d) {
    int y;
    y=b+f_2(c,d);
    return y;
}

int f_2(int x, int y) {
    int z;
    z=x-y;
    return z;
}
```

Rješenje

- rezultat prevođenja i povezivanja

```

int f_0(int a, int b, int c, int d, int e) {
    int y;
    0x00000184 STMDB R13!, {R4-R9, R14}
    0x00000188 MOV R4, R0 ; R0 sadrži a
    0x0000018C MOV R5, R1 ; R1 sadrži b
    0x00000190 MOV R6, R2 ; R2 sadrži c
    0x00000194 MOV R7, R3 ; R3 sadrži d
    0x00000198 LDR R8, [R13, #0x001C] ; e je na stogu
    y=a+f_1(b,c,d)+e;
    0x0000019C MOV R2, R7
    0x000001A0 MOV R1, R6
    0x000001A4 MOV R0, R5
    0x000001A8 BL f_1(0x00000158)
    0x000001AC ADD R0, R0, R4
    0x000001B0 ADD R9, R0, R8
    return y;
    0x000001B4 MOV R0, R9 ; R0 sadrži rezultat
    0x000001B8 LDMIA R13!, {R4-R9, R14}
}

0x000001BC BX R14

int f_1(int b, int c, int d) {
    int y;
    0x00000158 STMDB R13!, {R4-R7, R14}
    0x0000015C MOV R4, R0
    0x00000160 MOV R5, R1
    0x00000164 MOV R6, R2
    y=b+f_2(c,d);
    0x00000168 MOV R1, R6
    0x0000016C MOV R0, R5
    0x00000170 BL f_2(0x0000014C)
    0x00000174 ADD R7, R0, R4
    return y;
    0x00000178 MOV R0, R7
    0x0000017C LDMIA R13!, {R4-R7, R14}
}

0x00000180 BX R14

int f_2(int x, int y) {
    int z;
    0x0000014C MOV R2, R0
    z=x-y;
    return z;
    0x00000150 SUB R0, R2, R1
}

0x00000154 BX R14

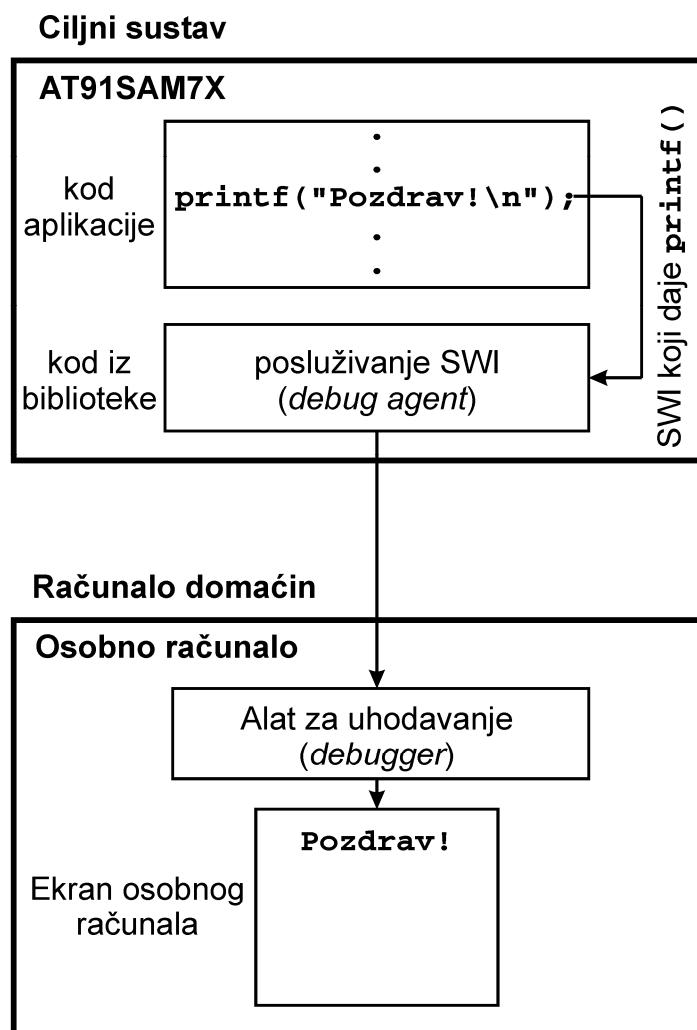
```

- 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 spremi se na stog
 - ⇒ vidi instrukcije STMDB i LDMIA
 - poziv funkcije
 - instrukcija BL (*branch and link*)
 - ⇒ adresa slijedeće instrukcije 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 programske prekid
 - komunikacija s računalom domaćinom
 - ima određen stupanj autonomije u izvršavanju
 - ⇒ u našem slučaju
 - simulira sklopolje neovisno o programu ciljnog sustava
 - obavlja radnje koje može zadati računalo domaćin
- **prijelazak s agenta na sklopolje 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
- za razliku od reimplementacije, kod SWI handlera svi pozivi prema sklopolju idu kroz jednu točku (SWI)
 - ⇒ to je bolje za uhodavanje
- uočiti
 - **semihost agent** i **SWI handler** koji obrađuje semihost pozive se ne razlikuju po filozofiji
 - ⇒ princip programiranja je isti
 - ipak, postoji razlika
 - semihost agent radi interaktivno s računalom domaćinom
 - ⇒ u tom smislu je autonoman
 - SWI handler proslijeđuje tražene operacije prema sklopolju
 - ⇒ gledano s aspekta ciljnog sustava, nema autonomiju

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
- **izlazni parametar**
 - sadržan u registru R0, ovisno o funkciji
- semihost operacije (ukupno 24 u trenutnoj inačici razvojnih alata)

```
0x01 => SYS_OPEN    // Otvaranje datoteke
0x02 => SYS_CLOSE   // Zatvaranje datoteke
0x03 => SYS_WRITEC  // Ispis karaktera na konzolu
0x04 => SYS_WRITE0  // stringa na konzolu
0x05 => SYS_WRITE   // Upis u datoteku
0x06 => SYS_READ    // Čitanje iz datoteke u meduspremnik
0x07 => SYS_READC   // Čitanje bajta s konzole
0x08 => SYS_ISERROR // Odredivanje je li vraćeni kod pogrešan

// itd.
```

7.12.3 Primjer posluživanja SWI

- primjer korištenja semihost okruženja

Primjer

Napisati program koji ispisuje poruku *Pozdrav svijete!*. Program se mora izvršavati **na sklopoljju** mikrokontrolera AT91SAM7X. Kao izlaznu jedinicu koristiti serijsko sučelje USART0. Za ispis koristiti funkciju **printf()**. Potrebno je koristiti **semihost** pozive.

Rješenje

- cijelokupni program obuhvaća
 - funkciju **main()**
 - inicijalizaciju SWI prekidnog vektora i sklopolja USART0
 - programsku podršku za posluživanje prekida
 - programsku podršku za rad sa sklopoljem USART0 koja radi u sklopu prekidnih funkcija

7.12.3.1 Funkcija main()

- programski odsječak

```
#include <stdio.h>

int main (void) {

    printf ("Pozdrav svijete!\n\r");
    printf ("Opet pozdrav!\r\n");

    while (1) {
        // ovdje dolazi neki koristan kod
    }

}
```

- paziti
 - nigdje u kodu neće biti korišten simbol
__use_no_semihosting_swi
⇒ linker neće upozoravati na funkcije koje traže posluživanje SWI
⇒ sve funkcije koje traže SWI (a koje se pozivaju) moraju biti implementirane
- uočiti
 - funkcija **main()** je "aplikacija"
 - aplikacija se nastoji održati neovisnom o sklopoljju
⇒ u funkciji **main()** zato nije napravljena inicijalizacija sklopolja
- inicijalizaciju sklopolja u ovom slučaju čini inicijalizacija sklopa USART0
⇒ premještena je u dio koji radi sa sklopoljem
 - radi se o inicijalizaciji
⇒ logičan izbor je *startup* odsječak tj. u dio koji sadrži osnovne inicijalizacije

7.12.3.2 Osnovne inicijalizacije

- za naš primjer potrebno su slijedeće inicijalizacije
 - inicijalizacija prekidnog vektora za SWI
⇒ potrebno za posluživanje prekida
 - definiranje veličine stoga u Supervisor načinu rada
⇒ jer SWI koristi Supervisor način rada
 - inicijalizacija serije
⇒ jer ćemo USART0 koristiti kao konzolu
- inicijalizacija prekidnog vektora

```

AREA    RESET, CODE, READONLY
ARM

Vectors      LDR    PC,Reset_Addr ; Reset
              LDR    PC,Undef_Addr ; Nedefinirana instrukcija
              LDR    PC,SWI_Addr ; Programski prekid
              LDR    PC,PAbt_Addr ; Nedefinirana adresa instr.
              LDR    PC,DAbt_Addr ; Nedefinirana adresa podatka
              NOP
              LDR    PC,IRQ_Addr  ; IRQ
              LDR    PC,FIQ_Addr  ; FIQ

IMPORT ASM_SWI_Handler

Reset_Addr   DCD    Reset_Handler
Undef_Addr   DCD    Undef_Handler
SWI_Addr    DCD    ASM_SWI_Handler
PAbt_Addr   DCD    PAbt_Handler
DAbt_Addr   DCD    DAbt_Handler
              DCD    0
IRQ_Addr    DCD    IRQ_Handler
FIQ_Addr    DCD    FIQ_Handler

Undef_Handler B     Undef_Handler
;SWI_Handler  B     SWI_Handler
PAbt_Handler B     PAbt_Handler
DAbt_Handler B     DAbt_Handler
IRQ_Handler  B     IRQ_Handler
FIQ_Handler  B     FIQ_Handler

```

- uočiti
 - početak koda za posluživanje SWI počinje na **ASM_SWI_Handler**
- inicijalizacija veličine stoga

```

UND_Stack_Size EQU 0x00000000
SVC_Stack_Size EQU 0x00000200 ; može učiniti i u Wizard-u
ABT_Stack_Size EQU 0x00000000
;
      ...

```

- odsječak asemblerorskog koda za inicijalizaciju serije

```
; Inicijalizacija sklopolja USART0
IMPORT init_USART0
BL    init_USART0

; Ulaz u C kod
IMPORT __main
LDR   R0, =__main
BX   R0
```

- C funkcija za inicijalizaciju serije (preuzeta iz poglavlja 7.5.2.6)

```
#include <at91sam7x128.h>

#define MCK 48000000 // fMCK=48MHz
#define BR 9600 // Zeljena brzina prijenosa
#define CD (MCK/16/BR) // konst. za generator takta za prijenos

void init_USART0 (void) {
    *AT91C_PMC_PCER=(1<<AT91C_ID_US0) | // Takt za USART0
    (1<<AT91C_IDPIOA); // Takt za PIOA

    *AT91C_PIOA_PDR=AT91C_PA1_TXD0; // Dodjela prikljucaka

    *AT91C_US0_CR=AT91C_US_RXDIS | // Unemoguci odasiljac
    AT91C_US_RSTTX; // Resetiraj odasiljac

    *AT91C_US0_MR=AT91C_US_USMODE_NORMAL | // normalan nacin rada
    AT91C_US_CLKS_CLOCK | // takt za USART je MCK
    AT91C_US_CHRL_8_BITS | // znak ima 8 bitova
    AT91C_US_NBSTOP_1_BIT | // 1 stop bit
    AT91C_US_PAR_NONE; // nema pariteta
    // MSBF=0 => prvo ide LSB

    *AT91C_US0_BRGR=CD; // inic. generatora takta za prijenos
    *AT91C_US0_CR=AT91C_US_TXEN; // enable odasiljac
}
```

- uočiti
 - kod iz funkcije `init_USART0()` mogao je biti izravno upisan u asemblerski startup kod
 - ovako je pregleđnije, a i koristimo već razvijenu funkciju

7.12.3.3 Posluživanje prekida

- posluživanje prekida sadrži
 - određivanje parametra **SWI_number**
⇒ iz operacijskog koda instrukcije SWI
 - poziv funkcije za obradu
⇒ ovisno o SWI_number
- programski odsječak u asembleru

```

PRESERVE8

AREA      ASM_SWI_Handler, CODE, READONLY
ARM                      ; mora biti ARM mod

EXPORT ASM_SWI_Handler
IMPORT C_SWI_Handler

T_bit EQU      0x20          ; Thumb bit (5) u CPSR/SPSR.

STMDB    R13!, {R0-R12,R14} ; Pohrana registara na stog.

MRS      R0, SPSR          ; R0=SPSR_SVC
TST      R0, #T_bit         ; SWI je dosao iz Thumb stanja?

                    ; DA=>
LDRNEH  R0, [R14,#-2]       ; Ucitaj operacijski kod
                            ; SWI instrukcije (Halfword)
BICNE   R0, R0, #0xFFFFFFFF00 ; Maskiraj "SWI number"

                    ; NE=>
LDREQ   R0, [R14,#-4]       ; Ucitaj operacijski kod
                            ; SWI instrukcije (Word)
BICEQ   R0, R0, #0xFF000000 ; Maskiraj "SWI number"

MOV      R1, R13            ; pokazivac na stog

BL      C_SWI_Handler       ; R0=SWI_nubmer
                            ; R1=pokazivac na argumete

LDMIA   R13!, {R0-R12,R15}^ ; Vracanje reg. sa stoga.

END

```

- uočiti
 - određivanje operacijskog koda instrukcije SWI
 - odrediti je li zahtjev došao iz ARM ili Thumb odsječka
 - pročitati operacijski kod instrukcije SWI
 - ⇒ poluriječ ako je zahtjev stigao iz Thumb moda
 - ⇒ riječ ako je zahtjev stigao iz ARM moda
 - maskirati odgovarajuće bitove
 - C funkcija se poziva s 2 argumenta
 - parametar **SWI_number** (u registru R0)
 - pokazivač na argumente (u registru R1)
 - posluživanje programskog prekida mora sačuvati sadržaj onih registara koje traže funkcije koje su ih pozvalе
 - ⇒ **stmbd** i **ldmia**
 - ovaj *handler* nije *reentrant*
 - ⇒ da bi bio *reentrant*, trebalo bi na početku spremiti, a na kraju vratiti SPSR
- 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;
    }
}
```
- uočiti
 - odsječak u asembleru i funkcija su pisani za opći slučaj
 - poslužuje SWI s parametrima 0x123456 i 0xAB
 - po želji se može proširiti za bilo koje parametre
 - rad ove funkcije bit će ilustriran na primjeru obrade *semihosting* poziva

7.12.3.4 Rad sa semihost pozivima

- pozivi operacija

```

void PURS_agent (int *arguments) {

    int service_no, r, *block;

    service_no = *arguments;                                // R0 sa stoga
    block      = (int *) (*arguments+1)); // R1 sa stoga

    switch(service_no) {
        case (0x01) : r = SYS_OPEN(block);                break; // <--
        case (0x02) : r = SYS_CLOSE(block);               break;
        case (0x03) : r = SYS_WRITEC(block);              break;
        case (0x04) : r = SYS_WRITE0(block);              break;
        case (0x05) : r = SYS_WRITE(block);               break; // <--
        case (0x06) : r = SYS_READ(block);                break;
        case (0x07) : r = SYS_READC(block);              break;
        case (0x08) : r = SYS_ISERROR(block);             break;
        case (0x09) : r = SYS_ISTATY(block);              break; // <--
        case (0x0A) : r = SYS_SEEK(block);                break;
        case (0x0C) : r = SYS_FLEN(block);                break;
        case (0x0D) : r = SYS_TMPNAM(block);              break;
        case (0x0E) : r = SYS_REMOVE(block);              break;
        case (0x0F) : r = SYS_REMOVE(block);              break;
        case (0x10) : r = SYS_CLOCK(block);              break;
        case (0x11) : r = SYS_TIME(block);                break;
        case (0x12) : r = SYS_SYSTEM(block);              break;
        case (0x13) : r = SYS_ERRNO(block);               break;
        case (0x15) : r = SYS_GET_CMDLINE(block);         break;
        case (0x16) : r = SYS_HEAPINFO(block);            break;
        case (0x17) :
            r=angel_SWIreason_EnterSVC(block);           break;
        case (0x18) :
            r=angel_SWIreason_ReportException(block); break;
        case (0x30) : r = SYS_ELAPSED(block);             break;
        case (0x31) : r = SYS_TICKFREQ(block);            break;
        // Nove arhitekture ovdje mozda budu imale jos poziva.
        // Zato default ...
        default      : r = SYS_DEF(service_no);          break;
    }

    *arguments=r;
}

```

- uočiti
 - argumenti programskog prekida čitaju se sa stoga
 - rezultat se stavlja izravno na stog
 - to je moguće jer SVE funkcije koje koriste ovakav semihosting moraju biti pisane tako da primaju jedan povratni argument
- argumenti se u funkcije **sys_...()** prenose preko pokazivača **block**
 ⇒ pokazivač **block** "došao" je kroz register R1 od funkcije koja je zatražila SWI
- za rad funkcije **printf()** na sklopolju potrebno je implementirati funkcije **SYS_OPEN()**, **SYS_WRITE()** i **SYS_ISTTY()**
- specifikacije ovih funkcija dane su u
 "RealView Compilation Tools - Developer Guide, Version 4.0, ARM 2009"
- **funkcija sys_OPEN()**
- funkcija otvara datoteku ili jedinicu
 - u slučaju semihostinga otvara ju na računalu domaćinu
 - u našem slučaju otvara ju na ciljnom sustavu
- ulazni argumenti
 - **block** (tj. R1 s kakvim je pozvan SWI) pokazuje na blok od 3 riječi
 - 1. riječ - pokazivač na string koji sadrži ime datoteke
 - 2. riječ - **int** koji opisuje način otvaranja (*opening mode*)
 - 3. riječ - **int** koji sadrži duljinu stringa s imenom datoteke
- izlazni argument (register R0)
 - pokazivač na datoteku (handle), $\neq 0$, ako nije došlo do greške
 - -1 ako je došlo do greške (ako datoteka ili jedinica nije otvorena)
- konzola
 - string "**:tt**" označava konzolu

- izvedba funkcije za zadani primjer

```
int SYS_OPEN(int *block){  
    char *device_name = (char *) *block;  
    //int opening_mode = *(block+1);  
    //int name_length = *(block+2);  
  
    if (!strncmp(device_name, ":tt", 3)) {  
        return 1; // tty  
    } else { // ostalo nije implementirano  
        return -1;  
    }  
}
```

- funkcija **SYS_ISTTY()**

- funkcija provjerava je li dana datoteka "spojena" s interaktivnom jedinicom (npr. konzolom)
- ulazni argument
 - **block** pokazuje na pokazivač na datoteku (koja je ranije otvorena)
- izlazni argument
 - 1 ako je dana datoteka interaktivna jedinica
 - 0 ako dana datoteka NIJE interaktivna jedinica
 - ≠ 0 i ≠ 1 ako je došlo do greške
- izvedba funkcije za zadani primjer

```
int SYS_ISTTY(int *block){  
    if (1==*block) {  
        return 1; // tty  
    } else { // ostalo nije implementirano  
        return 0;  
    }  
}
```

- funkcija **SYS_WRITE()**
- funkcija upisuje otvara datoteku ili jedinicu
 - u slučaju semihostinga otvara ju na računalu domaćinu
 - u našem slučaju otvara ju na ciljnog sustavu
- ulazni argumenti
 - **block** pokazuje na blok od 3 riječi
 - 1. riječ - pokazivač na datoteku (prethodno otvorenu)
 - 2. riječ - pokazivač na podatke koji se upisuju
 - 3. riječ - **int** koji sadrži broj bajtova koji se upisuju
- izlazni argument
 - 0 ako su podaci uspješno upisani ili
 - broj bajtova koji nisu upisani
- izvedba funkcije za zadani primjer

```
int SYS_WRITE(int *block){  
    int handle = *block;  
    char *str = (char *) (*(block+1));  
    int string_size= *(block+2);  
  
    if (1==handle) { // tty  
        while(string_size != 0) {  
            sendchar_USART0(*str);  
            str++;  
            string_size--;  
        }  
        return 0;  
    } else { // ostalo nije implementirano  
        send_string("SYS_WRITE : handle nije implementiran\n\r");  
        return string_size;  
    }  
}
```

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

```
int SYS_CLOSE(int *block){  
    send_string("Funkcija nije implementirana: SYS_CLOSE \n");  
    return 0;  
}  
  
// itd.
```

- **funkcije za rad sa sklopoljjem USART0**
- funkcija se koristi za ispis karaktera na seriju
 - koristit ćemo funkciju iz poglavlja 7.5.2.6

```
void sendchar_USART0 (int ch) {  
    while (!(*AT91C_US0_CSR & AT91C_US_TXRDY));  
        // AT91C_US_TXRDY=(0x1<<1) => TXRDY Interrupt  
    *AT91C_US0_THR = ch; // Transmitter Holding Register  
}
```

- **pomoćne funkcije**
- funkcija za ispis poruka neimplementiranih funkcija
 - izvedena je tako da izravno upisuje u USART0
- primjer izvedbe

```
void send_string (char *s) {
    while(*s != '\0') {
        sendchar_USART0(*s);
        s++;
    }
}
```

- poruke neimplementiranih funkcija mogu se ispisivati i pomoću funkcije **printf()**
 - u tom slučaju u SWI bi se ulazilo više puta
⇒ posluživanje SWI bi moralo biti izvedeno kao *reentrant*
 - u prvoj fazi uhodavanja poželjno je jednostavnije rješenje
- uočiti
 - kao što je rečeno ranije, **PURS_agent()** iskorišten je za implementaciju funkcija iz biblioteke koje koriste sklopoljje
⇒ u ovom smislu funkcija **PURS_agent()** nije agent
 - takav naziv je odabran zato jer bi kostur zapisanje agenta imao isti oblik
- u ponudi ARM razvojnih alata nalazi se nekoliko semihosting agenata koji su namijenjeni uhodavanju programske podrške

7.12.3.5Poruke o greškama

- tokom izvršavanja programa mogu se pojaviti greške koje se javljaju na konzolu
 - ovaj ispis koristi funkciju `_ttywrch()`
- funkcija `_ttywrch()` mora moći ispisati karakter na konzolu
⇒ koristi *semihosting* i to funkciju `SYS_WRITEC()`

Primjer

Neke funkcije iz standardnih biblioteka alociraju memoriju tokom izvršavanja programa. Stoga je potrebno predvidjeti *heap* odgovarajuće veličine. Što bi se dogodilo tokom izvršavanja programa "Pozdrav svijete!" opisanog u prethodnom primjeru, ako je *heap* premalen.

Rješenje

- na konzolu bi bila ispisana poruka
Out of heap memory
- funkcija `SYS_WRITEC()`
- ulazni argument
 - `block` pokazuje na karakter koji treba upisati na konzolu (u našem slučaju USART0)
- izlazni argument
 - ne postoji, tj. može se vratiti bilo koja vrijednost u R0
- izvedba funkcije za zadani primjer

```
int SYS_WRITEC(int *block){  
    sendchar_USART0(*block);  
    return 0;  
}
```

- uočiti
 - kod uhodavanja programske podrške ovakvi ispisimaju neprocjenjivu ulogu

7.13 Zadaci za samostalni rad

1. U dokumentacije proizvođača mikrokontrolera AT91 ARM Thumb-based Microcontrollers, Datasheets, Atmel 2007 potrebno je pronaći opisa brojila TC0. Proučiti rad brojila. Napisati programsku podršku koja će omogućiti da LED dioda spojena na priključak PPB1 trepće s periodom 2s. Koristiti prekidni način rada.
2. Programsku podršku opisanu u poglavlju 7.12 Semihosting potrebno je nadopuniti tako da podržava i rad s funkcijom **fprintf()**, za slučajeve kad ona ispisuje string na konzolu i na serijsko sučelje USART0. Pozivi funkcije **fprintf()** u tim slučajevima imaju oblik

```
// Ispis na konzolu
fprintf (stdout,"Ispis na konzolu preko fprintf.\r\n");

// Ispis na seriju
FILE *fSerija;
fSerija=fopen ("serija","w");
fprintf (fSerija,"Ispis na seriju preko fprintf.\n\r");
fclose (fSerija);
```

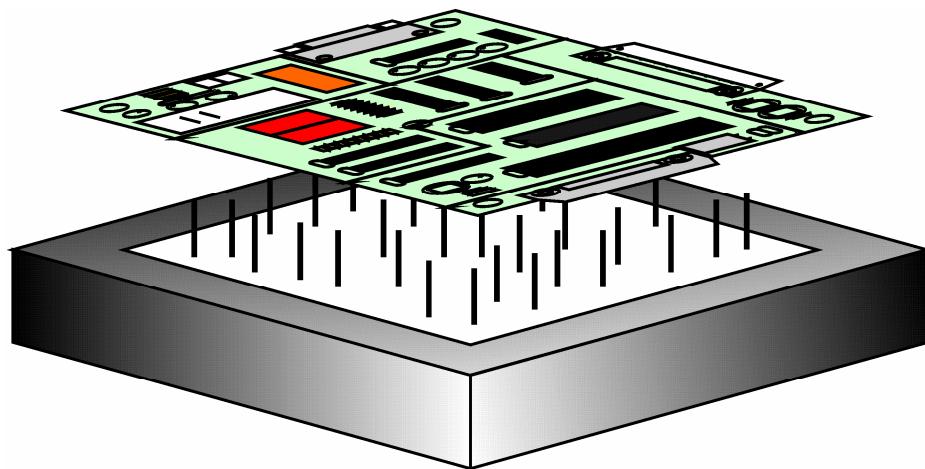
8 Uhodavanje i ispitivanje sustava

8.1 Osnovni pojmovi

- sklopolje i programska podrška URS-a mogu biti vrlo složeni
 - razvoj se radi postupno
⇒ sklopolje i program se uhodavaju u malim cjelinama
 - proizvodnja je podijeljena u više faza
⇒ u svakoj fazi radi se posebna provjera pojedinih podsustava
- **uhodavanje**
 - provjera ispravnosti sustava tokom razvoja uređaja
 - uključuje modifikaciju sklopolja i tek napisane programske podrške
- **ispitivanje**
 - provjera ispravnosti sustava tokom proizvodnje
 - uključuje otklanjanje pogrešaka nastalih u montaži i instalaciji
- **simulator**
 - sustav koji **korištenjem abstraktnog modela određuje odziv** izvornog sustava
 - ako je simulator u programu, on se izvodi na računalu
 - primjer: simulacija njihala u MATLAB-u
 - primjer: simulacija ARM mikrokontrolera na osobnom rač.
 - ako je simulator sklop, on u pravilu nema funkcionalnost izvornog sklopa
 - npr. simulacija njihala pomoću električnih mreža
- **emulator**
 - sustav koji **u potpunosti reproducira vanjsko ponašanje** izvornog sustava
⇒ s vanjskih "priključnica" emulator nije moguće razlikovati od izvornog sustava
 - ako je izvorni sustav sklop, i emulator je sklop (ako je izvorni sustav program, i emulator je program)
 - simulator može biti sastavni dio emulatora
⇒ npr. spor procesor može se emulirati brzim koji izvodi simulaciju abstraktnog modela sporog procesora
 - emulator reproducira ponašanje izvornog sustava pa se u ovom slučaju simulacija MORA izvoditi u realnom vremenu

8.2 Razvoj alata za uhodavanje i ispitivanje

- provjera ispravnosti sklopovlja već je spomenuta u kontekstu uhodavanja sklopovlja
⇒ vidi poglavlje 6 Puštanje u pogon
- ovdje će ta tema biti obrađena s aspekta metodologije i sustava koji se koriste u uhodavanjima i ispitivanjima URS-ova
- sredinom 70-tih godina 20. stoljeća, tehnologija štampanih pločica koristila se u masovnoj proizvodnji
⇒ postojala je potreba za provjerom ispravnosti sklopovlja tokom proizvodnje
 - alati za provjeru koristili su sonde s iglicama (*bed of nails*)
 - provjera spojeva (*integrity test*)
⇒ izvodila se mjeranjem impedancije između iglica
 - funkcionalna provjera (*integrity test*)
 - mogla se izvoditi samo rudimentarno
⇒ na iglice su se dovodili pobudni signali



- uočiti
 - iglice služe da se ostvari kontakt s priključcima
 - kontakt se lako može ostvariti s donje strane štampane pločice
- ⇒ problemi

- problemi
 - pojava komponenata za površinsku montažu (*Surface Mounted Components, SMD*)
 - ⇒ razmak između priključaka je malen
 - ⇒ komponente se leme s obje strane pločice
 - pojava višeslojnih štampanih pločica
 - ⇒ spojevi na unutrašnjim slojevima mogu se ispitati samo ako prospoji prolaze kroz pločicu (*Through Hole Vias*), ali ne ako se nalaze između slojeva (*Buried Vias*)
 - razvoj sustava sadržanog na jednom integriranom sklopu (*System on Chip Design, SoC Design*)
 - ⇒ vanjski priključci pojedinih komponenata su nedostupni
 - uočiti da ovaj problem nije postojao 70-tih i 80-tih godina
- rješenje
 - sredinom 80-tih godina okupila se grupa koja je pokušala riješiti spomenute probleme
 - u početku grupa je okupljala inženjere iz europskih kompanija
 - ⇒ nazvana *Joint European Test Action Group (JETAG)*
 - kasnije su se pridružili i američki inženjeri
 - ⇒ grupa je nazvana *Joint Test Action Group (JTAG)*

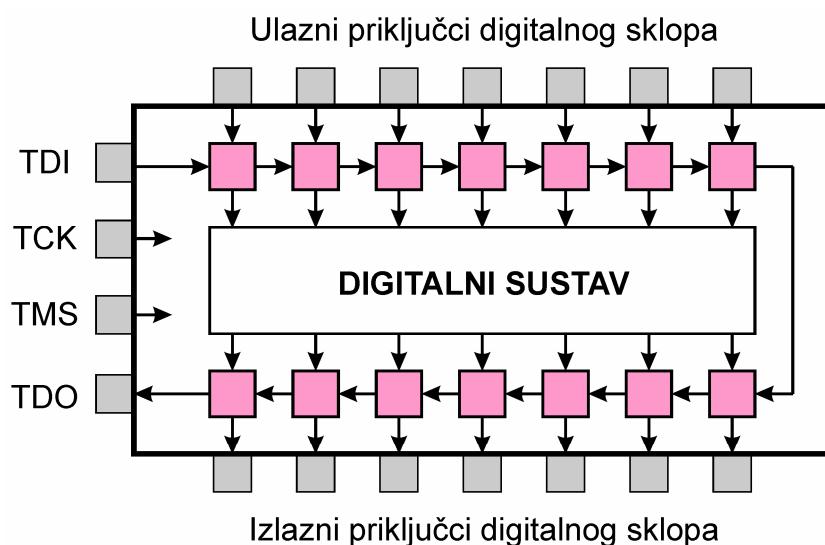
⇒ rezultat rada JTAG grupe je standard koji opisuje takozvanu *Boundary Scan* arhitekturu

 - *IEEE standard test access port and boundary-scan architecture*, IEEE Standard 1149.1-1990 i 1149.1-2001
 - ⇒ opisuje ispitivanje digitalnih sustava
 - postoje i inačice standarda koje obuhvaćaju miješane analogno-digitalne sustave
 - standard se stalno razvija, radi se na novim inačicama, ...
- danas praktički sve složenije komponente podržavaju ispitivanje temeljeno na spomenutom standardu
- komponente sadrže sklopove koji podržavaju
 - provjeru povezanosti na štampanoj pločici
 - učitavanje konfiguracije
 - primjer: učitavanje programa u memoriju mikrokontrolera
 - funkcionalno ispitivanje

8.3 Boundary scan arhitektura

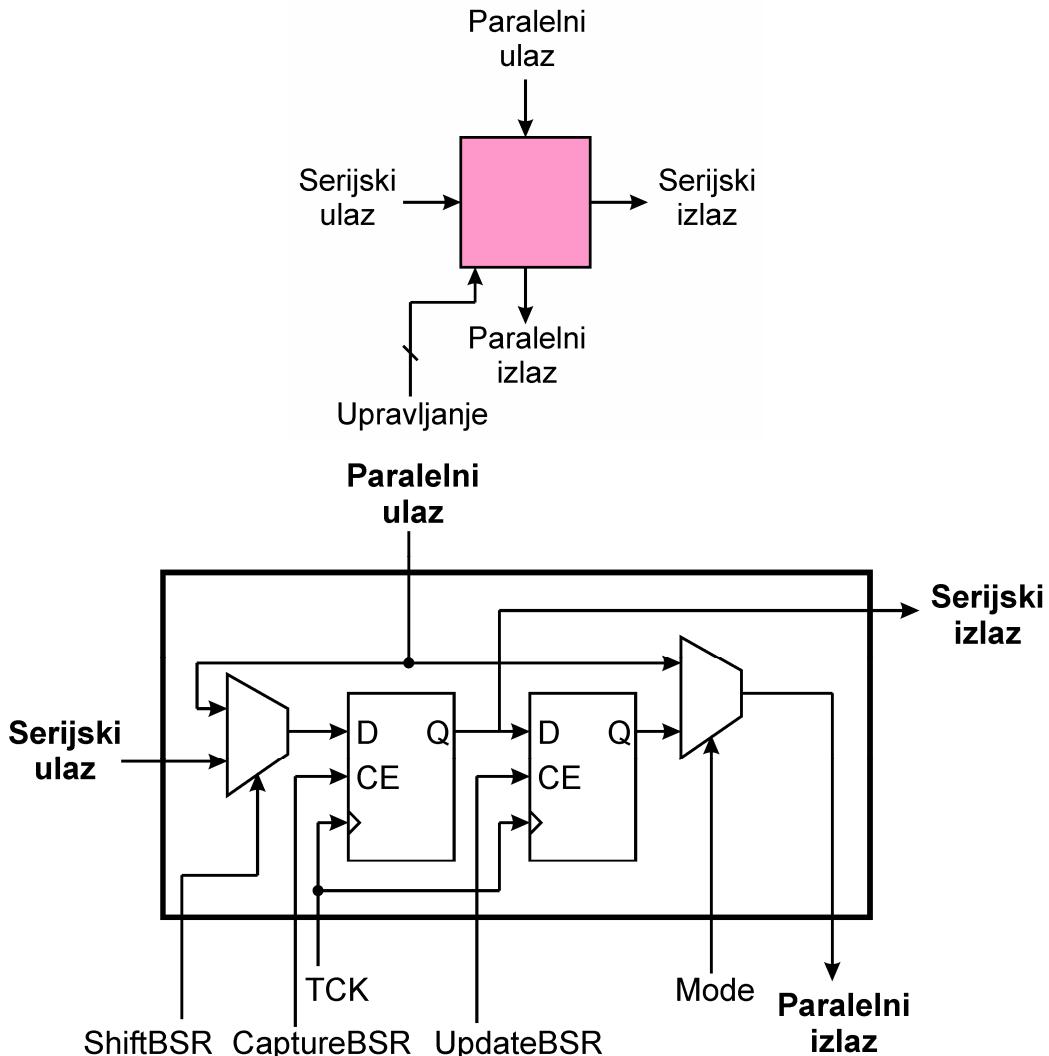
8.3.1 Princip rada

- svakom digitalnom integriranom sklopu na priključnice (na *chip*) se dodaju takozvane ***boundary-scan*** ćelije
 - ćelije zamjenjuju ranije spomenute iglice
 - ćelije omogućuju pristup svakom priključku (osim mase i napajanja)
 - u regularnom radu ćelije ne utječu na rad sklopa



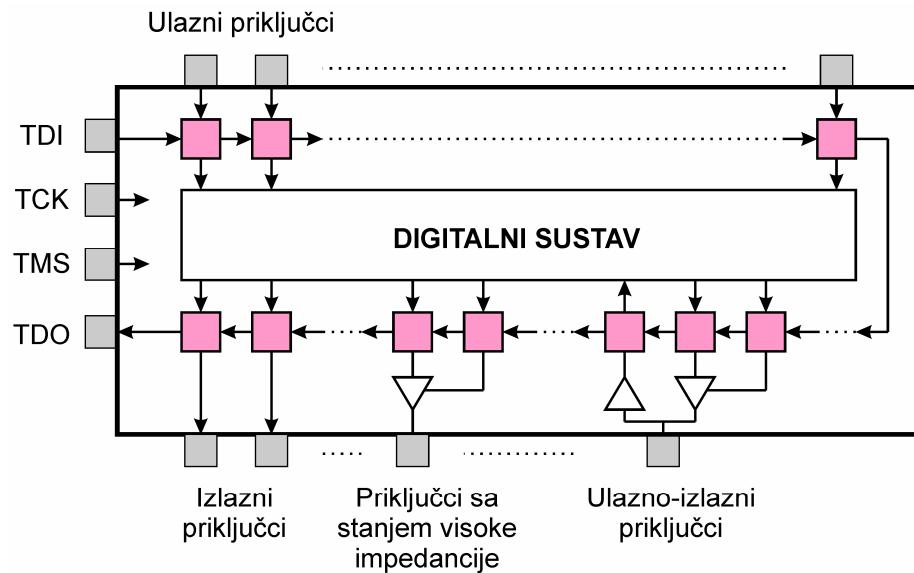
- komunikacija s ćelijama
 - ćelije formiraju posmačni registar, tzv. ***Boundary-scan*** registar
 - u registar se piše i iz njega se čita **paralelno**
 - ulazni priključci integriranog sklopa
⇒ ulaz u registar su priključci
⇒ izlazi ćelija spojeni su na digitalno sklopovlje
 - izlazni priključci integriranog sklopa
⇒ ulaz u registar su signali iz digitalnog sklopovlja
⇒ izlazi ćelija spojeni su na priključke
 - u registar se piše i iz njega se čita i **serijski**
 - ⇒ priključak *Test Data In*, TDI
 - ⇒ priključak *Test Data Out*, TDO
 - komunikacija je sinkrona s taktom
 - ⇒ priključak *Test Clock*, TCK
 - operacija koja se izvodi određena je upravljačkim signalom
 - ⇒ priključak *Test Mode Select*, TMS

- izvedba ćelije
 - standard opisuje isključivo funkciju ćelije, a ne i izvedbu
 - primjer izvedbe

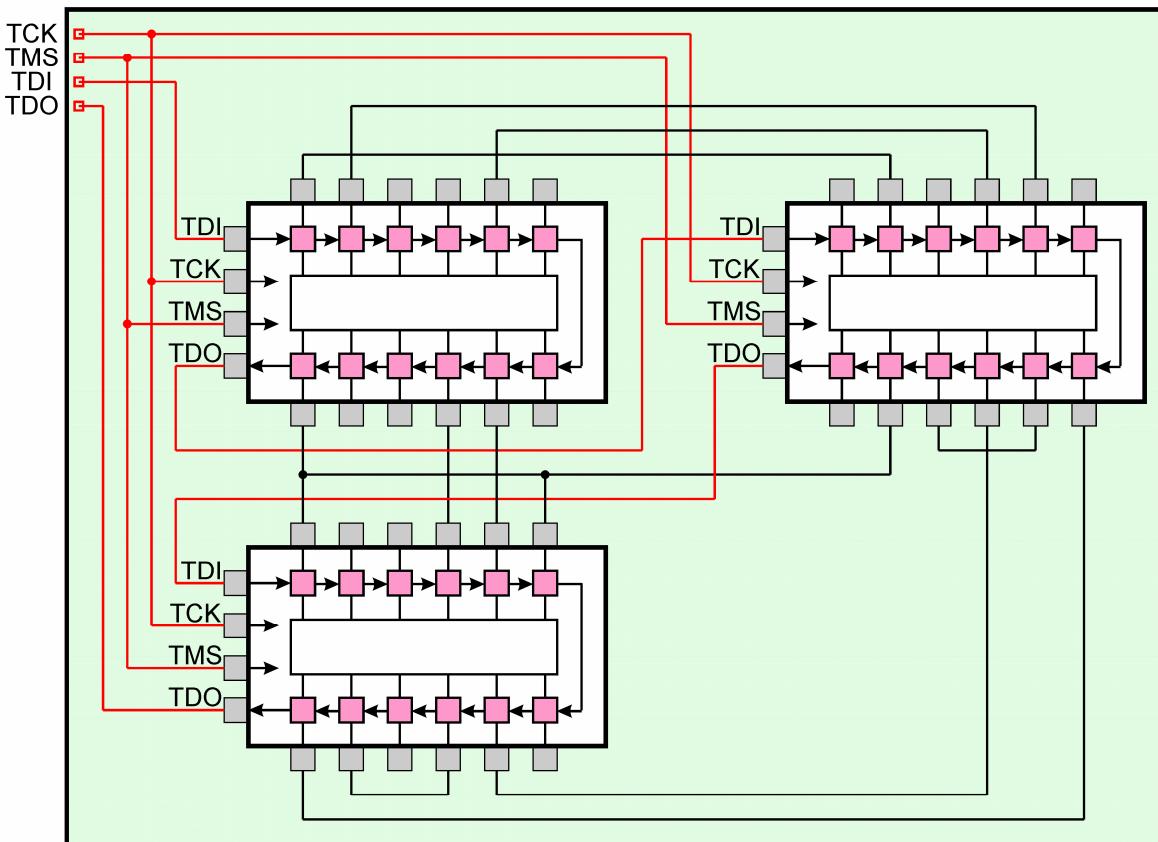


- uočiti operacije
 - prolaz podatka kroz ćeliju (*transparency*)
 - regularni rad sklopa (a ne *boundary scan*)
 - izvodi se izlaznim multipleksorom
 - ⇒ kašnjenje ovakvog multipleksora je zanemarivo
 - ⇒ ćelije praktički ne utječu na rad sklopa
 - učitavanje podatka s paralelnog ulaza (*capture*)
 - upis podatka na paralelni izlaz (*update*)
 - posmak podatka u lancu (*shift*)
 - ⇒ upis i čitanje registra

- broj ćelija može biti veći od broja priključaka kad postoje
 - ⇒ priključci sa stanjem visoke impedancije (*3-state pins*)
 - ⇒ dvosmjerni priključci (*bidirectional pins*)



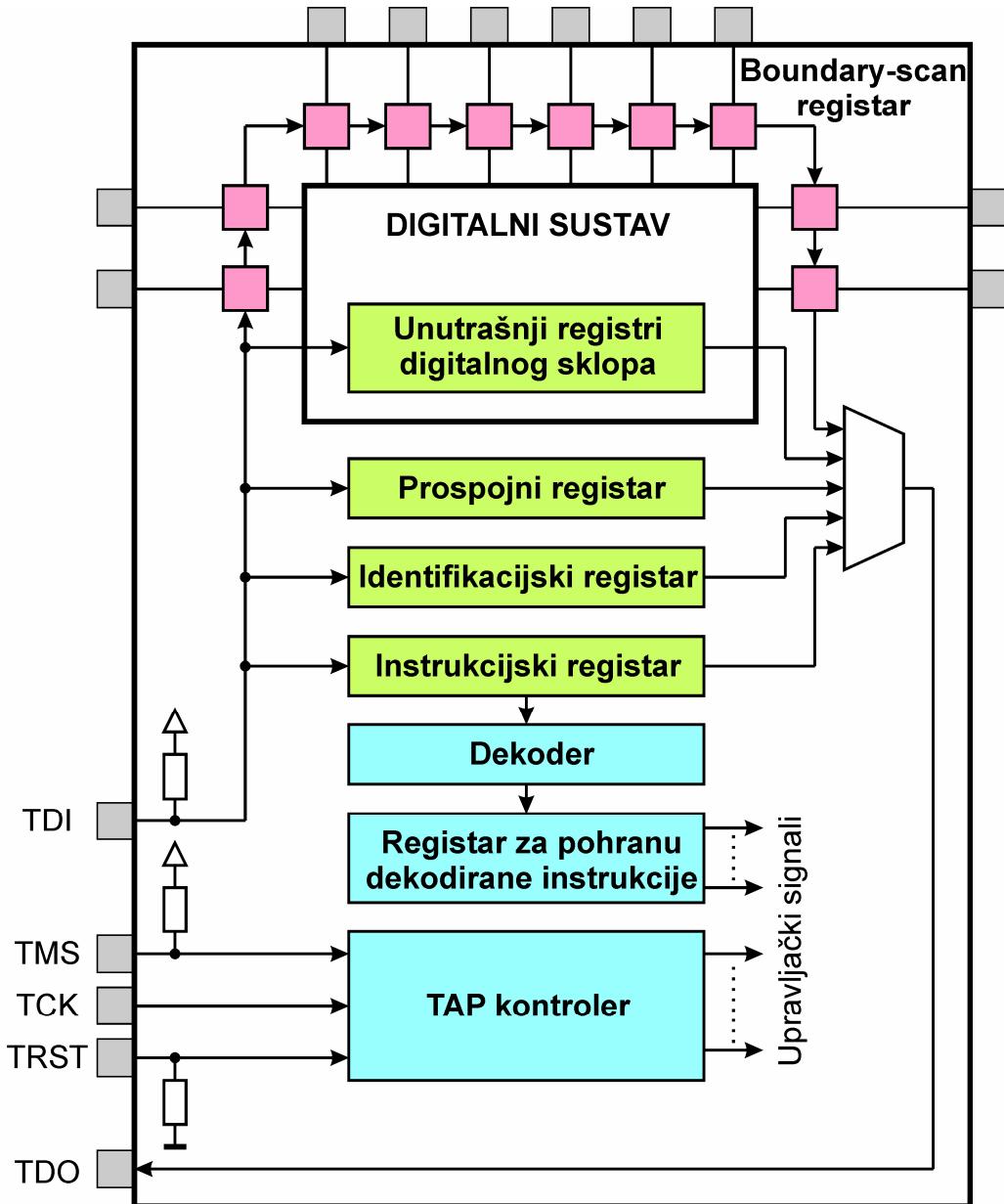
- povezivanje komponenata na štampanoj pločici



- uočiti
 - SVE komponente povezane su u JEDAN lanac (*chain*)

8.3.2 Izvedba sklopolja

- izvedba prema standardu IEEE Standard 1149.1



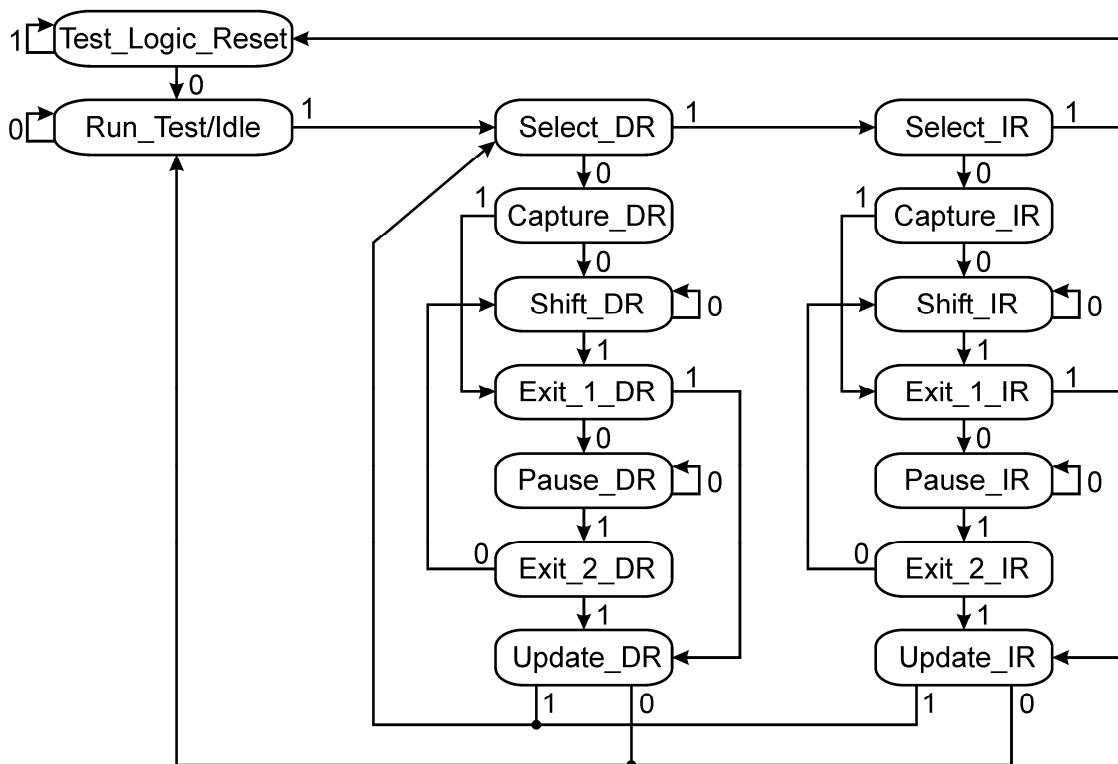
- uočiti
 - sučelje boundary-scan sklopoljva čine priključci
 - TDI, TMS, TCK, TDO koji su uvijek prisutni
 - TRST koji prema standardu može, ali ne mora postojati
 - ovo sučelje se izvorno zove ***Test Access Port (TAP)***
 - sučelje se često naziva i **JTAG sučelje (JTAG interface)**
 - registri koji obavezno postoje
 - *Boundary-scan* registar
 - sadrži sve ćelije
 - broj bitova, n, veći je ili jednak broju digitalnih priključaka
 - prolazni registar (*By-pass register*)
 - $n=1$ (1 bistabil)
 - kad ima mnogo komponenata u lancu, lanac sadrži mnogo bitova
 - ⇒ punjenje lanca trajalo dugo
 - ⇒ ako se želi preskočiti neka komponenta u lancu, ona se konfigurira tako da koristi prolazni registar
 - Instrukcijski registar (*Instruction register*)
 - u ovaj registar učitava se trenutna instrukcija
 - postoje 4 instrukcije koje sklopolje obavezno mora podržavati i proizvoljan broj dodatnih instrukcija
 - ⇒ $n \geq 2$
 - opcionalno, TAP može sadržavati i
 - Identifikacijski registar (*Identification register*)
 - ovaj registar je u pravilu izведен kao EEPROM ili FLASH
 - raspored bitova u registru
- | | | | |
|----------------|--------------------------|---------------------------|----------|
| 31-28 | 27-12 | 11-1 | 0 |
| Verzija | Oznaka komponente | Oznaka proizvođača | 1 |
- razne registre koji pripadaju digitalnom sustavu

8.3.3 Upravljanje sklopovljem

- upravljanje cjelokupnim sklopovljem je izvedeno pomoću TAP kontrolera

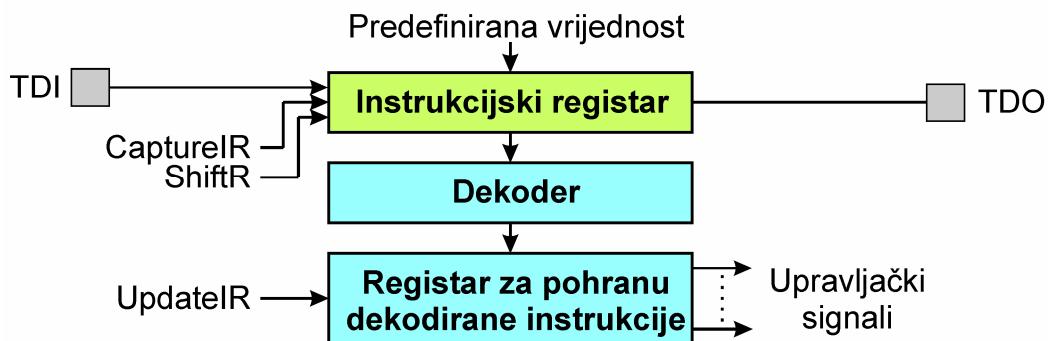


- TAP kontroler je automat s konačnim brojem stanja
 - definiran je standardom
 - prijelazi između stanja se odvijaju na rastući brid signala TCK
 - uvjeti prijelaza su stanja signala TMS

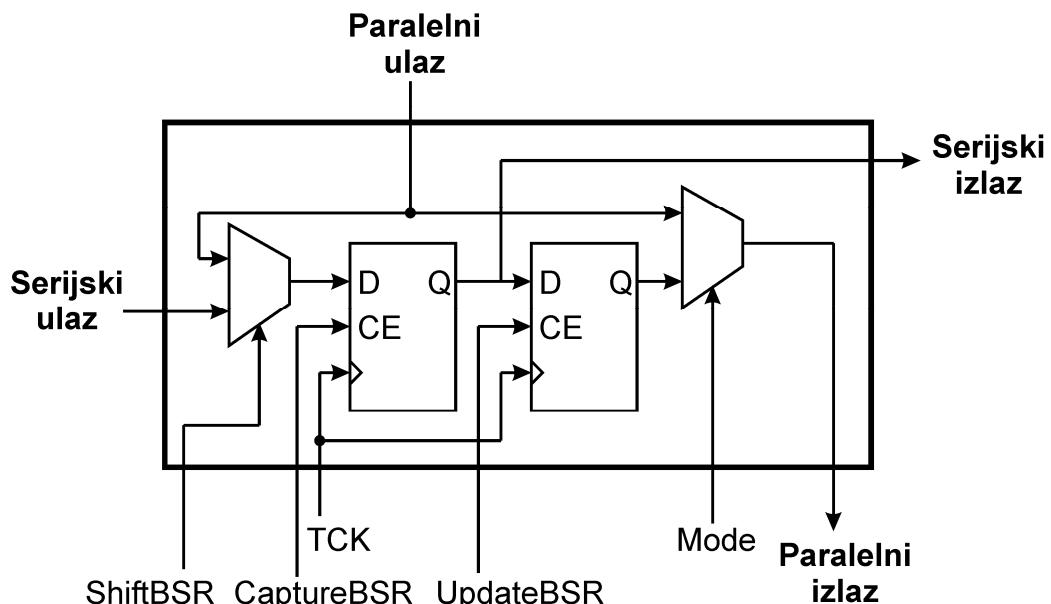


- uočiti
 - nakon pojave napajanja, FSM je u stanju *Reset* sve dok je TMS=1
 - pojavom TMS=0, FSM prelazi u stanje *Idle*
 - nakon stanja *Idle* moguće je prijeći u
 - granu za upravljanje svim podatkovnim registrima (TMS=1)
 - granu za upravljanje instrukcijskim registrom (TMS=1,1)
 - stanje *Reset* (TMS=1,1,1)

- upravljanje Instrukcijskim registrom obuhvaća
 - upis predefinirane konstante
 - aktivni signal **CaptureIR**
 - koristi se kod autotesta sklopovlja za testiranje
 - posmak i upis bita s priključka TDI
 - aktivni signal **ShiftIR**
 - koristi se za upis instrukcije
 - pohrana upisane instrukcije u registar za pohranu
 - aktivni signal **UpdateIR**
 - koristi se za čuvanje tekuće instrukcije



- upravljanje podatkovnim registrima
 - za upravljanje koristi signale iz TAP kontrolera
 - **CaptureDR**
 - **ShiftDR**
 - **UpdateDR**
 - koji je registar prozvan ovisi o upravljačkim signalima koje daje Registar za pohranu dekodirane instrukcije

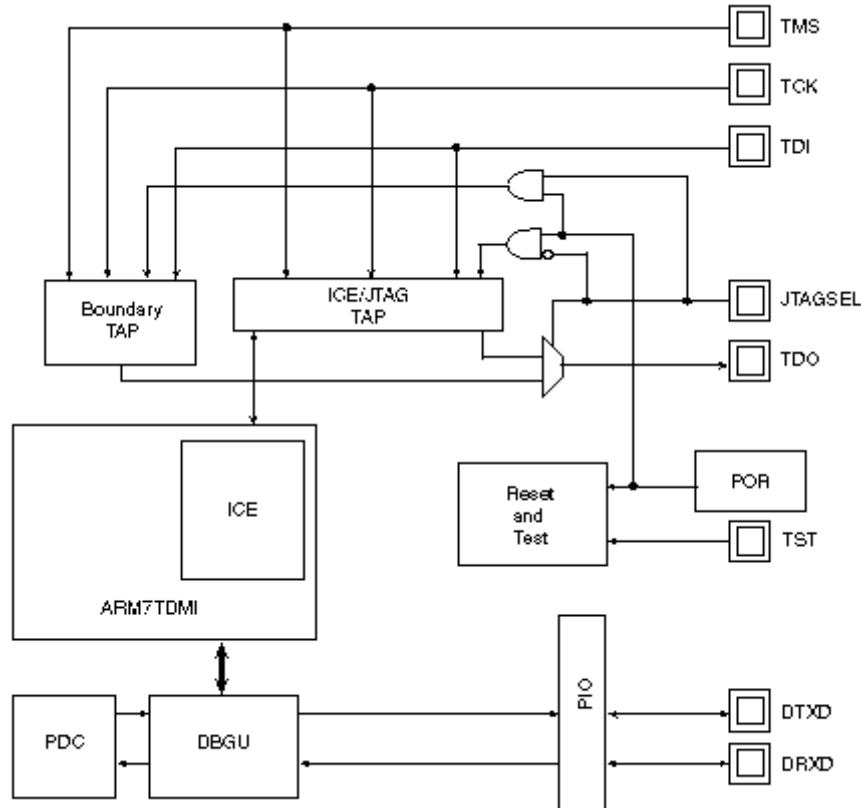


8.3.4 Rad sa sklopoljem

- instrukcije koje se upisuju u Instrukcijski registar pridijeljena su simbolička imena
- prema standardu boundary-scan sklopolje mora podržavati 4 instrukcije
 - Bypass
 - između TDI i TDO postavlja se Prospojni registar
 - Sample
 - priprema uzorkovanja paralelnog ulaza u Boundary-scan registar
 - Preload
 - priprema slanja signala na paralelni izlaz Boundary-scan registra
 - Extest
 - između TDI i TDO postavlja se Boundary-scan registar
- prema standardu postoje i opcionalne instrukcije
 - Intest, Runbist, Highz, Clamp i dr.
- korisnik prema potrebi smije dodavati vlastite instrukcije
- da bi se automatizirao rad s *Boundary-scan* sklopoljem koriste se posebni jezici
 - jezici koji opisuju implementaciju IEEE1149.1 standarda u pojedinom sklopu (temelje se na jeziku VHDL)
 - *Boundary-Scan Description Language* (BSDL)
 - *Hierarchical Scan Description Language* (HSDL)
 - jezici za opis pobude i odziva
 - *Serial Vector Format* (SVF)
 - *Standard Test And Programming Language* (STAPL)
- proizvođači komponenata obično isporučuju i opise *Boundary-scan* implementacije u nekom od spomenutih jezika
- ovom prilikom nećemo ulaziti u detalje rada s ovim jezicima

8.4 Ispitivanja i uhodavanje mikrokontrolera familije AT91SAM7X

- pojednostavljena blokovska shema sklopolja za ispitivanje i uhodavanje



- uočiti
 - JTAG se ne mora koristiti isključivo za boundary scan
 - u ovom slučaju postoje 2 TAP-a
 - TAP za klasični boundary-scan mikrokontrolera
 - TAP za pristup sklopolju za uhodavanje procesorske jezgre
 - željeni TAP se odabire postavljanjem stanja na priključku JTAGSEL
 - stanje JTAGSEL mora biti definirano prije reseta
⇒ stanje se ne može (ne smije) mijenjati u toku rada

8.4.1 *Boundary-scan* mikrokontrolera

- TAP za boundary scan
 - čini ga *Boundary-scan* sklopolje opisano u prethodnim poglavljima
 - selektirano je za JTAGSEL=1
- sadrži registre
 - Prospojni registar
 - Instrukcijski registar
 - *Boundary-scan* registar (BSR)
 - ima ukupno 187 bitova (ćelija)
 - većina priključaka tretira se kao ulazno-izlazni
⇒ koriste se do 3 ćelije po priključku
 - primjer

| Priklučak mikrokontrolera | Tip priključka | Bit u BSR | Tip ćelije u BSR |
|---------------------------|----------------|-----------|------------------|
| PA30/IRQ0/PCK2 | ulazno-izlazni | 187 | ulazna |
| | | 186 | izlazna |
| | | 185 | upravljačka |
| PA0/RXD0 | ulazno-izlazni | 184 | ulazna |
| | | 183 | izlazna |
| | | 182 | upravljačka |
| itd. | itd. | itd. | itd. |

- Identifikacijski registar
 - podsjetimo se

| | | | |
|---------|-------------------|--------------------|---|
| 31-28 | 27-12 | 11-1 | 0 |
| Verzija | Oznaka komponente | Oznaka proizvođača | 1 |

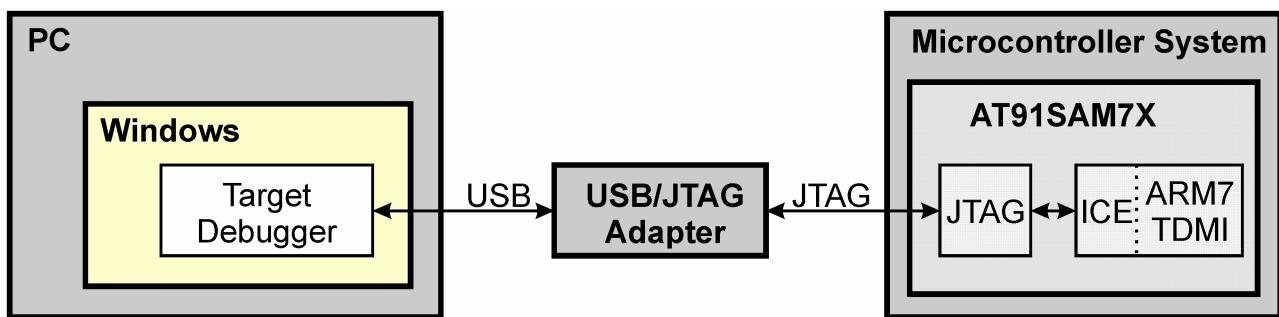
- kod mikrokontrolera AT91SAM7X128

| | | | |
|-------|--------|-------|---|
| 31-28 | 27-12 | 11-1 | 0 |
| 0x0 | 0x5B16 | 0x01F | 1 |

- podržava instrukcije
 - Sample, Extest and Bypass (ne koristi komandu Preload)

8.4.2 Uhodavanje sustava pomoću sklopovlja za uhodavanje

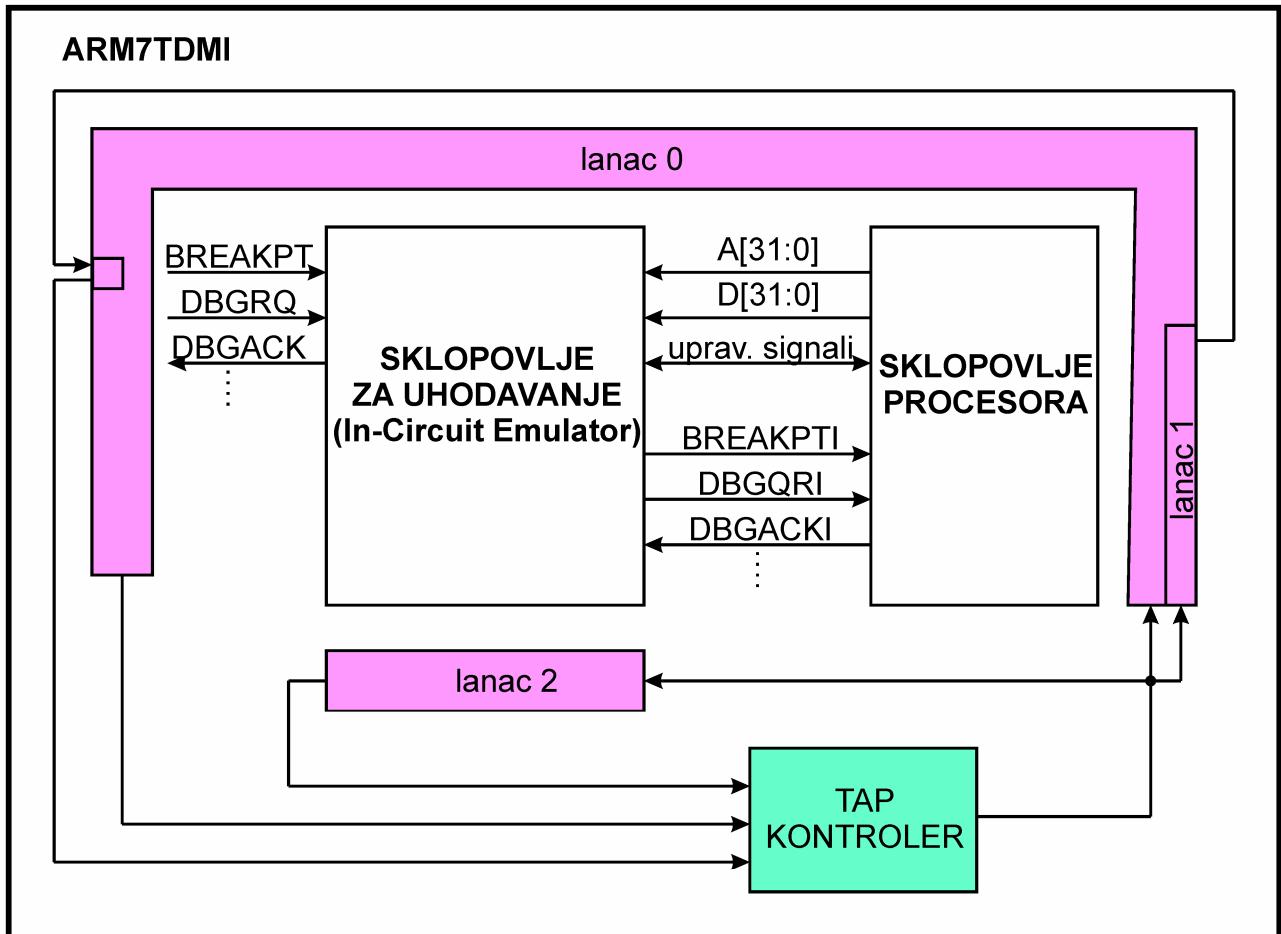
- ideja
 - program se izvršava na ciljnog sustavu (*target system*)
 - npr. računalni sustav koji sadrži µC AT91SAM7X
 - računalo domaćin (*host computer*) ima pristup sklopovlju ciljnog sustava
 - npr. osobno računalo
 - za vezu računala domaćina i ciljnog sustava koristi se sklopovlje za pretvorbu protokola
 - npr. - USB /JTAG
 - ULINK, JLINK i sl.



- postupak uhodavanja
 - temelji se na presretanju željenih operacija procesora
 - presretanje izvodi posebno sklopovlje koje je priključeno na sabirnice procesora
 - ovo sklopovlje prati signale i uzrokuje
 - točka prekida (*breakpoint*)
 - ⇒ do prijelaza u debug stanje dolazi **kad se počne izvršavati određena (označena) instrukcija**
 - točka nadzora (*watchpoint*)
 - ⇒ do prijelaza u debug stanje dolazi **kad se pristupi određenom podatku u memoriji**

8.4.3 Sklopovlje za uhodavanje procesorske jezgre

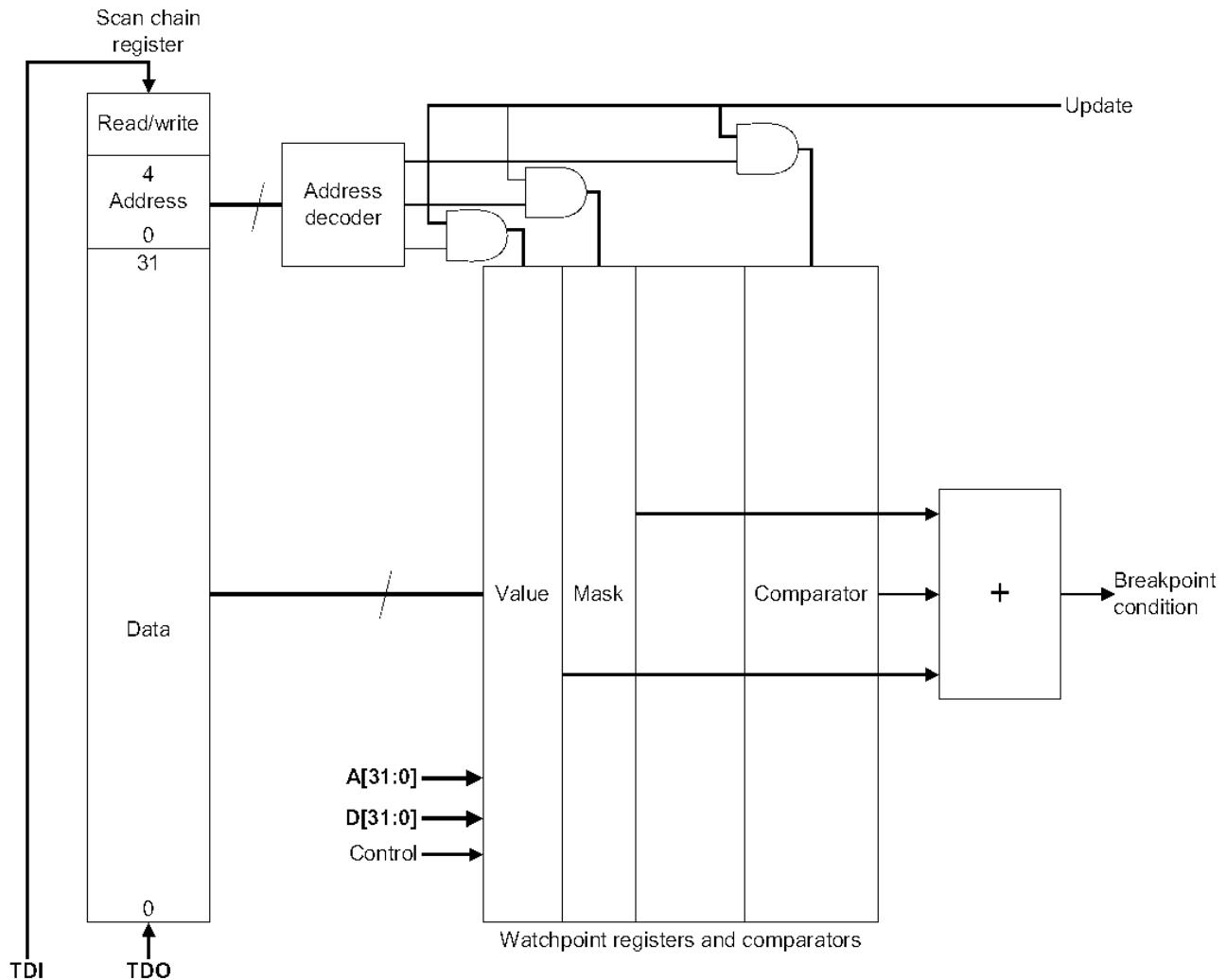
- pojednostavljena blokovska shema sklopovlja za uhodavanje



- za pristup jezgri tokom uhodavanja koriste se 3 lanca
 - lanac 0 sadrži
 - sve podatkovne, adresne i upravljačke priključke procesora
 - upravljačke signale sklopovlja za uhodavanje
 - lanac 1 je podskup lanca 0 i sadrži
 - podatkovnu sabirnicu
 - signal za prijelaz procesora u debug način rada (BREAKPT)
 - lanac 2
 - služi za pristup registrima sklopovlja za uhodavanje

- procesor može biti
 - u normalnom stanju
⇒ procesor izvršava program
 - debug stanju
⇒ procesor je pod kontrolom sklopolja za uhodavanje
- prijelaz procesora u debug stanje
 - ICE sklopolje aktivira signal BREAKPTI
 - procesor odgovara postavljanjem signala DBGACKI
- u debug stanju, procesor može raditi u jednom od dva načina rada
 - *halt* način rada (*halt mode*)
 - dolazi zahtjev za prijelaz u debug stanje
 - procesorska jezgra se zaustavlja i odspaja od ostatka sustava
 - računalo domaćin preko JTAG sučelja komunicira sa sklopoljem za uhodavanje
 - računalo domaćin može u jezgru ubaciti instrukciju STM i njome "pročitati" sadržaj svih registara
 - uočiti
 - u *halt* načinu rada procesor ne radi i ako dođe zahtjev za prekid od vanjskog sklopolja on neće biti poslužen
 - monitor načina rada (*monitor mode*)
 - dolazi zahtjev za prijelaz u debug stanje
 - ovisno o izvoru zahtjeva generira se prekid
 - ⇒ *breakpoint* generira *Instruction Abort*
 - ⇒ *watchpoint* generira *Data Abort*
 - komunikaciju s računalom domaćinom učestvuje takozvani **monitor program** koji se izvodi na procesoru
 - monitor program može komunicirati s ICE sklopoljem
 - procesor vidi ICE sklopolje kao koprocesor 14
 - uočiti
 - u monitor načinu rada procesor i dalje radi i prekidi od vanjskog sklopolja će biti posluženi

- sklopovlje *In-Circuit Emulatora* sadrži 2 *watchpoint* jedinice
- blokovska shema *watchpoint* jedinice



- uočiti
 - *watchpoint* jedinica sadrži
 - komparator
 - radi usporedbu očekivane i stvarne vrijednosti na sabirnici
 - kad se na sabirnici pojavi očekivana vrijednost, komparator indicira breakpoint ili watchpoint
 - registre
 - sadrže željene vrijednosti na sabirnici i maske za pojedine bitove koje ne treba pratiti
 - registri se programiraju preko JTAG sučelja

- napomena
 - proizvođač ovo sklopolje za uhodavanje naziva *In-Circuit Emulator*
 - Što je *In-Circuit Emulator* procesora?
 - to je sklop koji potpuno zamjenjuje procesor
 - okolno sklopolje ne razlikuje emulator od stvarnog procesora
 - programer ima potpunu kontrolu nad izvršavanjem koda (učitavanje, *breakpoints*, *watchpoints*)
 - u ovom slučaju **ne radi** se o pravom *In-Circuit Emulatoru* već o sklopu za uhodavanje koji podsjeća na *logički analizator*
 - ipak, ovo sklopolje omogućava kontrolu nad izvršavanjem programa dok je procesor ugrađen u sklopolje
⇒ zato se ovdje koristi naziv *In-Circuit Emulator*

8.5 Pitanja za provjeru znanja

1. Objasniti razliku između simulatora i emulatora.
2. Opisati *boundary-scan* arhitekturu. Nacrtati spoj sklopovlja za testiranje u slučaju kad se na jednoj štampanoj pločici nalazi više integriranih sklopova. Nacrtati primjer čelije u *Boundary-scan* registru i opisati njen rd.
3. Opisati izvedbu sklopovlja za *boundary-scan*. Koje registre sadrži sklopovlje i čemu oni služe?
4. Opisati rad TAP kontrolera. Na primjeru upisa u Instrukcijski registar opisati rad automata s konačnim brojem stanja koji se nalazi u TAP kontroleru.
5. Koje sklopove za uhodavanje i ispitivanje sadrže mikrokontroleri familije AT91SAM7X? Nacrtati blokovsku shemu sklopova za uhodavanje i opisati ulogu pojedinih komponenata.
6. Opisati sklopovlje za uhodavanje koje sadrži procesorska jezgra ARM7TDMI. Što je to *In-Circuit Emulator*, koje sklopove on sadrži i kako mu se pristupa sa strane JTAG sučelja? Što je *breakpoint* i *watchpoint*? Koje načini rada postoje u debug stanju procesora ARM7TDMI. Po čemu se oni razlikuju?

9 Periferno sklopolje u mikrokontrolerima

- mikrokontroleri sadrže razne periferijske sklopove (vidi poglavlje 3.1 Osnovna blokovska shema)
 - sklopovi za komunikaciju
 - *Universal Serial Bus* (USB)
 - *Universal Synchronous Asynchronous Receiver/Transmitter* (USART)
 - *Serial Peripheral Interface* (SPI)
 - *Two-wire Interface* (TWI)
 - *Ethernet MAC (Media Access Control) Controller* (EMAC)
 - *Controller Area Network* (CAN)
 - sklopovi za obradu analognih signala
 - analogno digitalni pretvarač, (*Analog-to-Digital Converter*, ADC)
 - pulsno širinski modulator, (*Pulse Width Modulator*, PWM)
 - sklopovi za pogon sučelja
 - grafički kontroler (ne postoji u familiji AT91SAM7X)
- neki od ovih sklopova već su spominjani ranije
⇒ USART
- u dalnjem tekstu bit će ukratko opisani najvažniji pojmovi vezani uz češće korištene spomenute sklopove

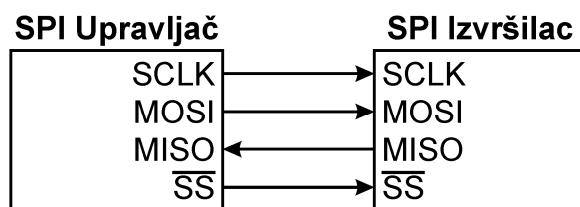
9.1 Sklopovlje za komunikaciju

9.1.1 Serial Peripheral Interface

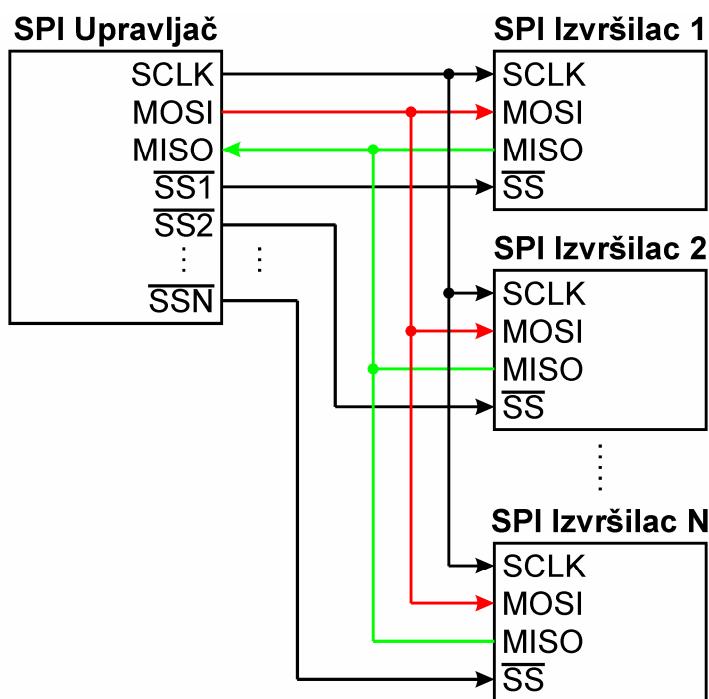
- *Serial Peripheral Interface* (SPI)

9.1.1.1 Principa rada SPI sučelja

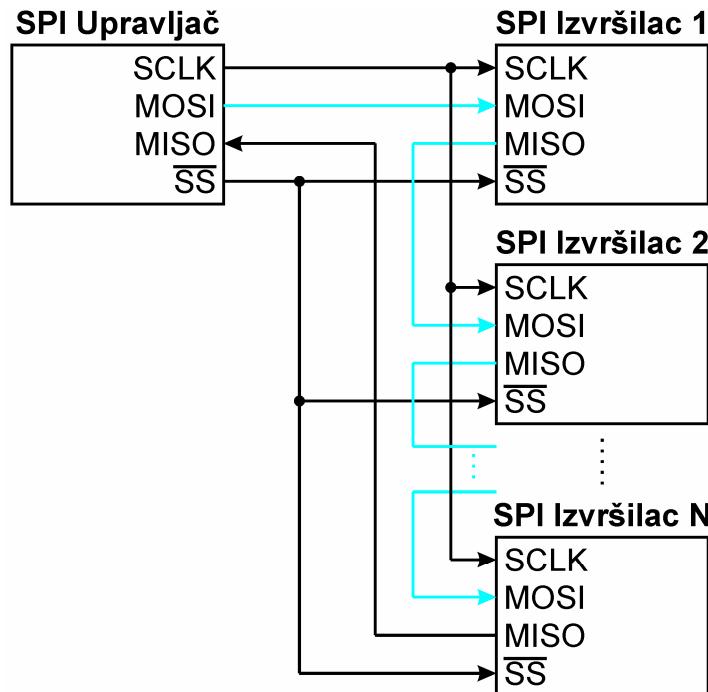
- SPI sabirnicu uvela je kompanija Motorola
- karakteristike SPI sabirnice
 - serijska (*serial*)
 - sinkrona (*synchronous*)
 - dvosmjerna (*full duplex*)
- spoj upravljača i izvršilaca
 - jedan izvršioc



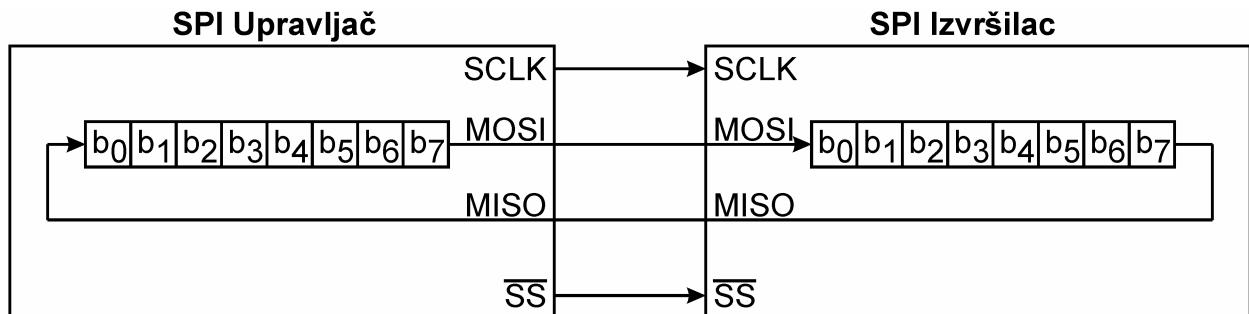
- više od jednog izvršioca - paralelan spoj



- više od jednog izvršioca - spoj u lanac

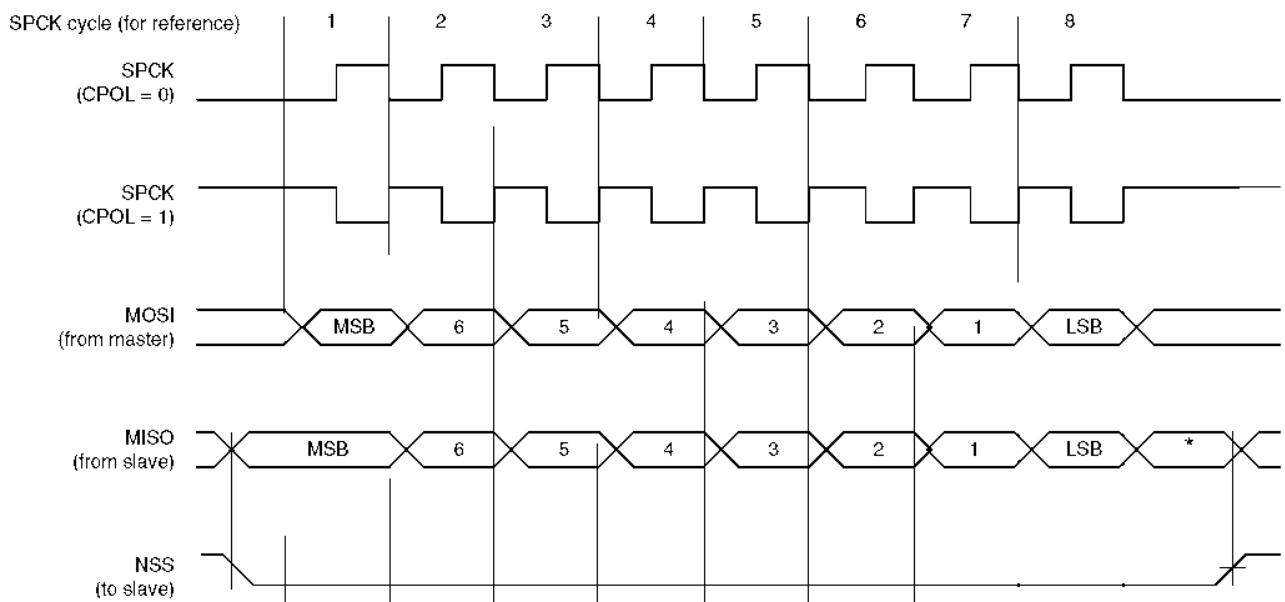


- signali
 - Serial Clock*, SCLK
 - Master Output Slave Input*, MOSI
 - Slave Output Master Input*, MISO
 - Slave Select*, SS
- način prijenosa podataka



- odašiljački i prijemni registar zajedno čine posmačni registar
- komunikacija je uvijek dvosmjerna
- prenosi se 8, 12, 16 ili neki drugi dogovoren broj bitova
- upravljač
 - inicira komunikaciju, generira takt i emitira podatke
- izvršilac
 - emitira podatke

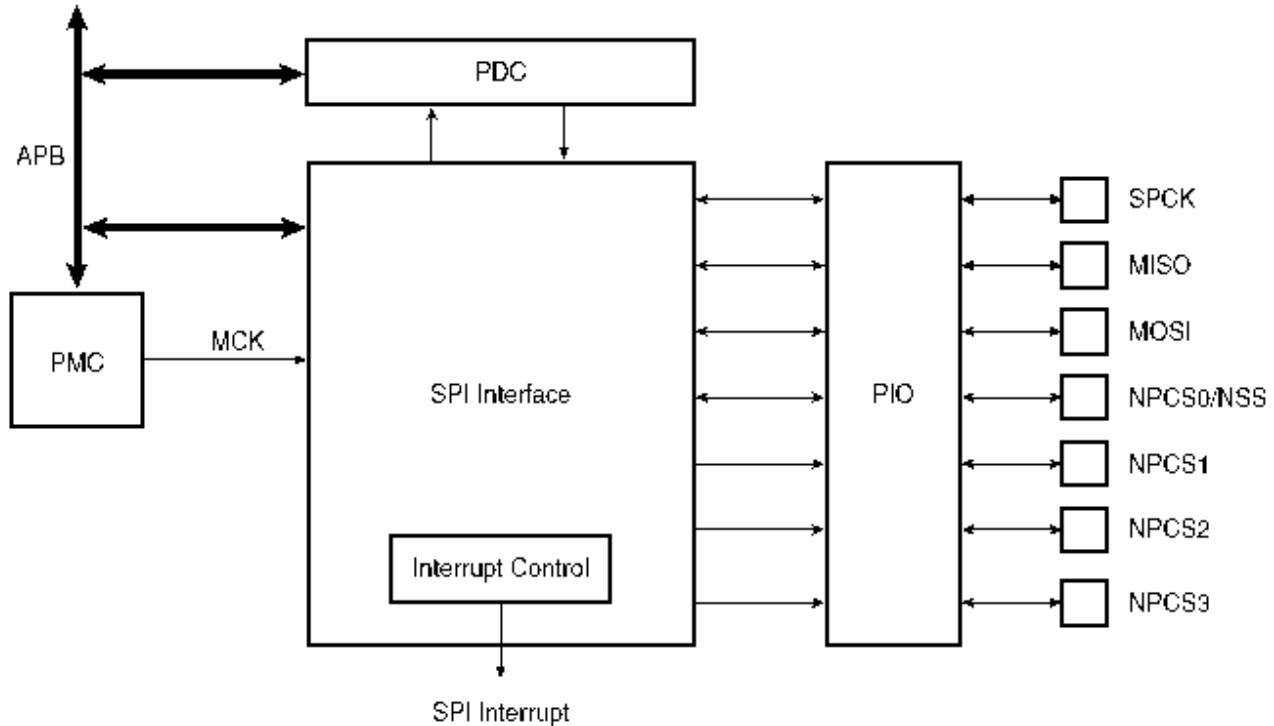
- uočiti
 - ako je komunikacija jednosmjerna
⇒ podatak koji je došao u drugom smjeru se zanemaruje
 - adresiranje izvršioca izvedeno je pomoću linije SS
 - adresiranje nije dio protokola
⇒ komunikacija je vrlo učinkovita
- signali
 - prijenos podatka sinkron je s taktom
 - moguće su 4 kombinacije polariteta i faze takta
 - takt aktivan visoko, rastući brid
 - takt aktivan visoko, padajući brid
 - takt aktivan nisko, rastući brid
 - takt aktivan nisko, padajući brid
 - primjer



- upotreba SPI sabirnice
 - komunikacija sa serijskim vanjskim jedinicama (npr. memorijama)
 - komunikacija između dvaju procesora
 - programiranje memorije mikrokontrolera (npr. Chipcon CC1010)
- brzina prijenosa
⇒ do 100 MBit/s

9.1.1.2 SPI sučelje u mikrokontrolerima familije AT91SAM7X

- mikrokontroler AT91SAM7X sadrži 2 SPI sučelja



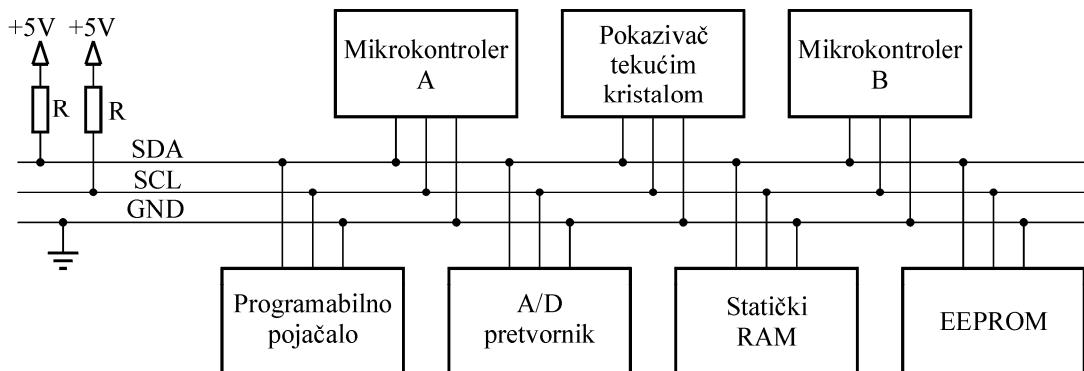
- uočiti
 - SPI sučelja izvedena su kao sekundarna funkcija priključaka PIOA
 - svako sučelje može raditi kao upravljač
 - podržava do 4 izvršioca
⇒ slave select izlazi: NPCS0, NPCS1, NPCS2, NPCS3
 - sučelje može raditi i kao izvršilac
 - slave select ulaz: NSS
- polaritet i faza takta, kao i čitav niz drugih parametara određen je sadržajem upravljačkih registara
⇒ vidi podatke proizvođača

9.1.2 Two-wire Interface

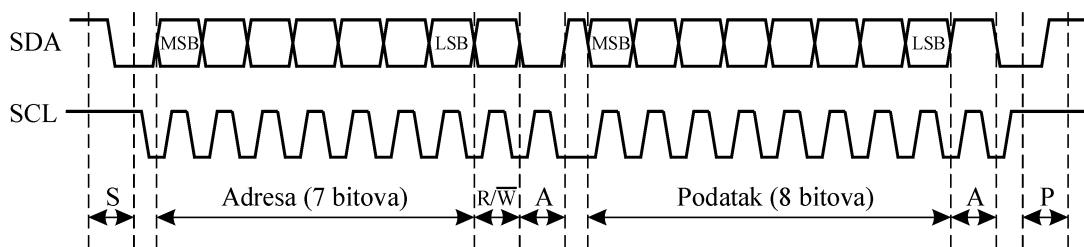
- Two-wire Interface (TWI)

9.1.2.1 Princip rada TWI sučelja

- ovo sučelje je kompatibilno s I²C sabirnicom
⇒ vidi predmet Ugradbeni računalni sustavi
- signali
 - Two-Wire Clock, TWCK
 - odgovara I²C signalu Serial Clock, SCL
 - Two-Wire Data, TWD
 - odgovara I²C signalu Serial Data, SDA
- spajanje komponenata na I²C sabirnicu



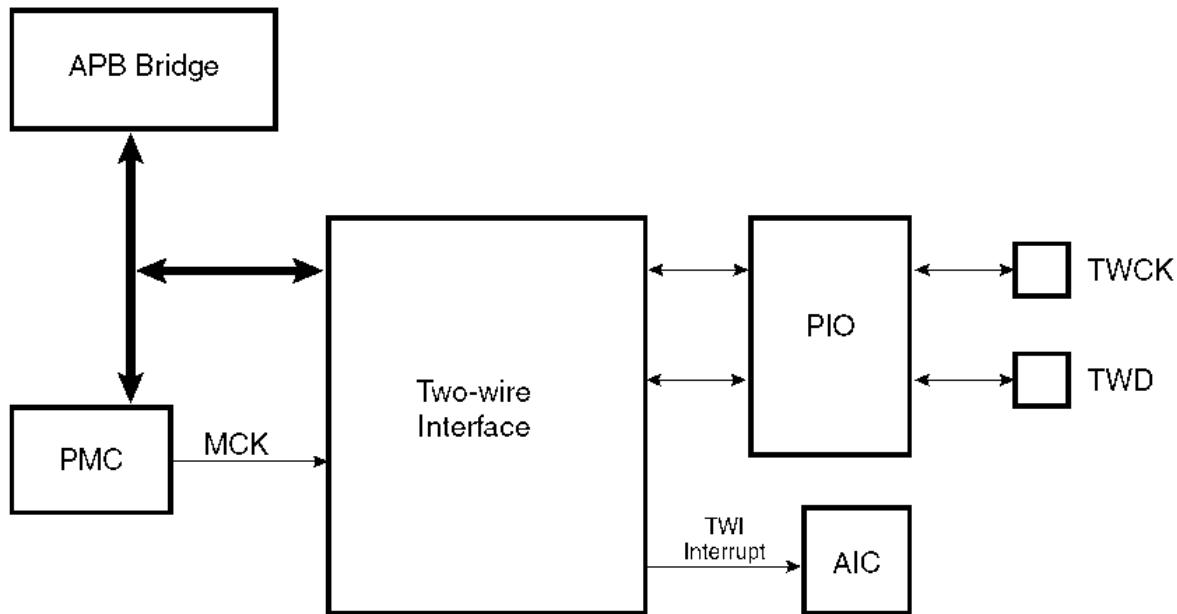
- signali na sabirnici



- za razliku od I²C standarda, TWI sklopoljje koje sadrže razni mikrokontroleri može imati programabilne parametre
 - npr. start i stop bit (postoji ili ne postoji)

9.1.2.2 TWI sučelje u mikrokontrolerima familije AT91SAM7X

- mikrokontroler AT91SAM7X sadrži TWI sučelje



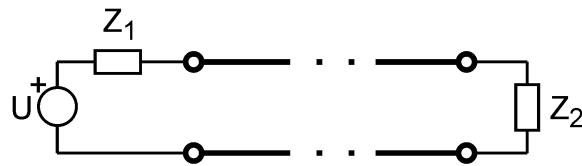
- uočiti
 - u sklopu URS-a, pokazana je programska emulacija I^2C sučelja
 - u ovom slučaju sabirnicu pogoni sklopovlje
 - za razliku od SPI sučelja, ovdje je adresiranje jedinica sadržano u protokolu
- parametri sklopovlja kao što su polaritet i faza takta, kao i čitav niz drugih parametara određen je sadržajem upravljačkih registara
⇒ vidi podatke proizvođača

9.1.3 Universal Serial Bus

- *Universal Serial Bus (USB)*

9.1.3.1 Prijenos signala linijom

- podsjetimo se linija
 - vidi predmet Električni krugovi
- prijenos signala linijom



- parametri linije
 - primarni R [Ω/m], G [S/m], L [H/m], C [F/m]
 - sekundarni parametri linije
 - zrcalna impedancija

$$Z_0 = \sqrt{\frac{R + sL}{G + sC}}$$

- Z_0 je ulazna impedancija beskonačno duge linije
- zrcalna konstanta prijenosa

$$\gamma = \sqrt{(R + sL)(G + sC)}$$

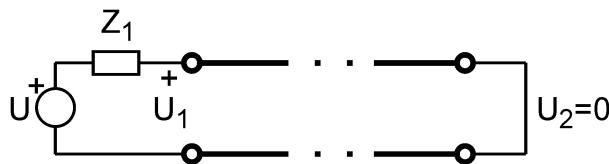
- posebni slučajevi
 - linija bez gubitaka $\Rightarrow R=0, G=0$
 - linija bez izobličenja $\Rightarrow \frac{R}{L} = \frac{G}{C}$
 - u oba slučaja Z_0 je realan i iznosi

$$Z_0 = \sqrt{\frac{L}{C}}$$

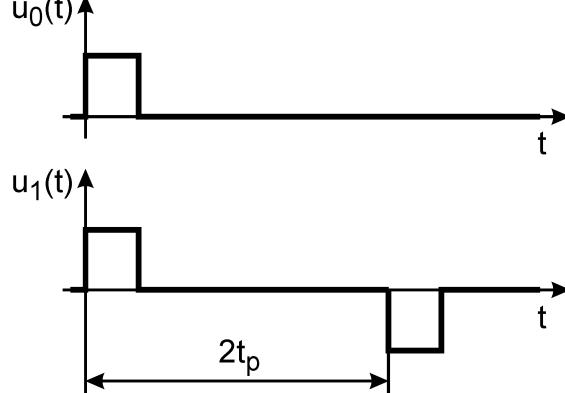
- propagacija impulsa duž linije
 - beskonačno duga linija
⇒ impuls doveden na ulaz linije propagira duž linije
 - linija konačne duljine
 - $Z_2 = Z_0$ (linija zaključena zrcalnom impedancijom)
 - obzirom da je Z_0 ulazna impedancija beskonačno duge linije, linija koja je zaključena zrcalnom impedancijom s ulaza se električki "vidi" kao beskonačna
⇒ impuls doveden na ulaz linije propagira do kraja linije
 - $Z_2 \neq Z_0$ (linija nije zaključena zrcalnom impedancijom)
⇒ dolazi do refleksije na diskontinuitetu impedancije
 - faktor refleksije

$$\Gamma = \frac{Z_2 - Z_0}{Z_2 + Z_0}$$

- primjer refleksije na liniji
 - pretpostavimo da je $Z_1 = Z_0$, a $Z_2 = 0$



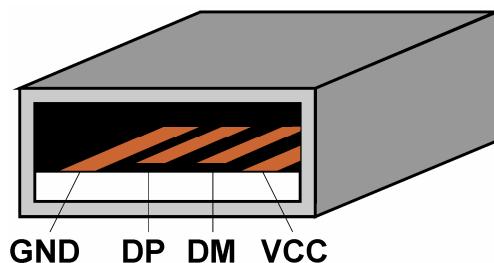
- faktor refleksije
- $$\Gamma = \frac{Z_2 - Z_0}{Z_2 + Z_0} = \frac{0 - Z_0}{0 + Z_0} = -1$$
- primjer valnih oblika signala na liniji



t_p je vrijeme propagacije signala od početka do kraja linije

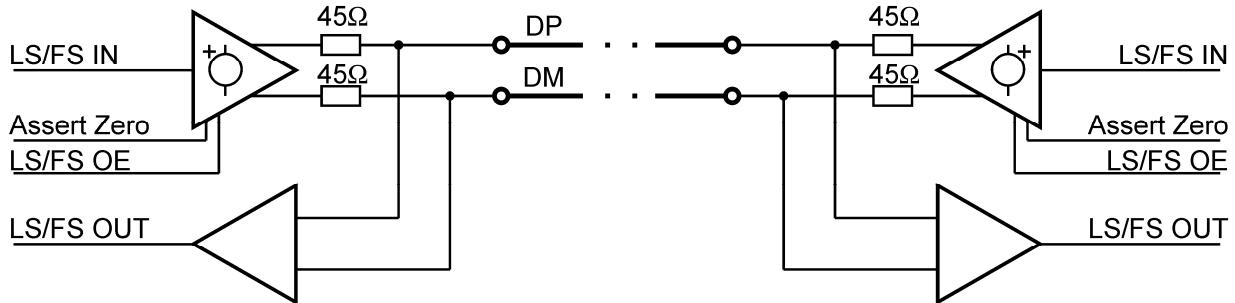
9.1.3.2 Sklopovlje USB sučelja

- postojeći standardi
 - USB 1.0 \Rightarrow izlazi iz upotrebe
 - teoretska brzina prijenosa 1.5MBit/s
 \Rightarrow *low speed* (LS)
 - USB 1.1 \Rightarrow izlazi iz upotrebe
 - teoretska brzina prijenosa 12MBit/s
 \Rightarrow *full speed* (FS)
 - **USB 2.0** \Rightarrow trenutno je u upotrebi
 - teoretska brzina prijenosa 480MBit/s
 \Rightarrow *hi speed* (HS)
 - USB 3.0 \Rightarrow trenutno je u razvoju
 - teoretska brzina prijenosa 4.8GBit/s
 \Rightarrow *super speed* (SS)
- signali USB priključka - primjer za konentor tipa "A"
 - priključci
 - masa, GND
 - napajanje, VCC
 - priključci za simetričnu podatkovnu liniju
 - DP ili D+
 - DM ili D-

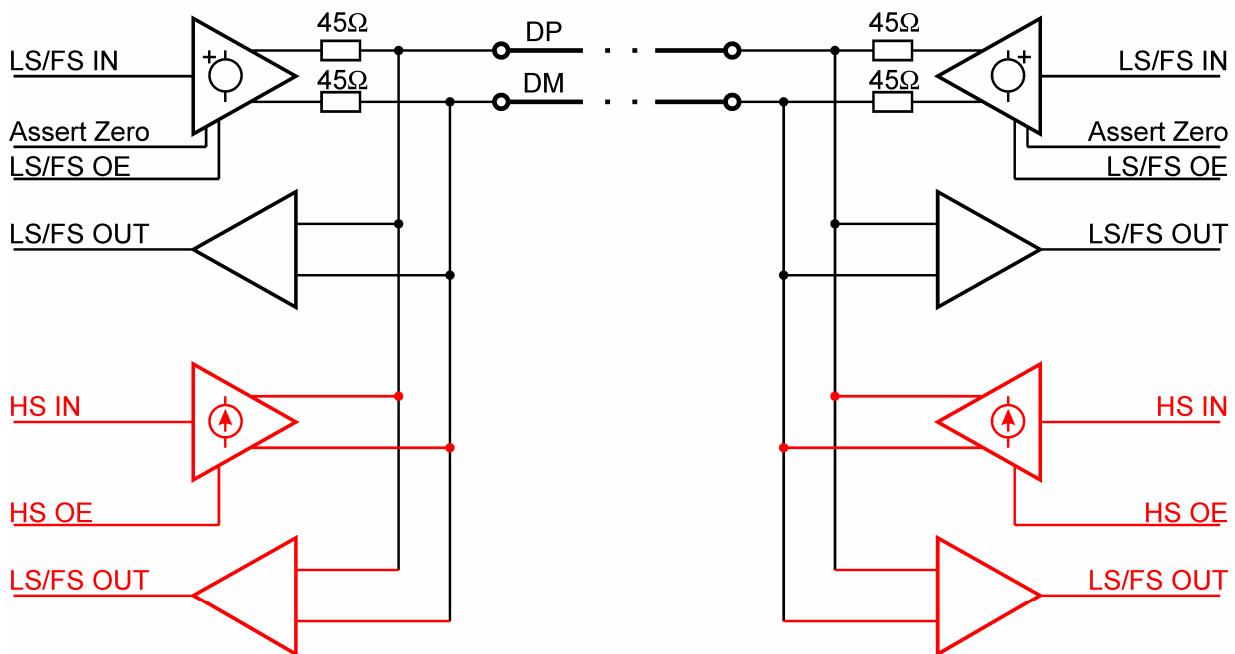


- uočiti
 - priključci GND i VCC su dulji
 \Rightarrow kod uključivanja prvo se dovodi napajanje
- prijenos
 - smjer se mijenja (*half duplex*) prema protokolu
- zrcalna impedancija linije koja se koristi za USB prijenos iznosi
 45Ω
- diferencijalni napon na liniji
 $\pm 400 \text{ mV}$

- pogonsko sklopolje koje odgovara standardu USB 2.0 mora podržavati *low speed, full speed i high speed*
- sklopolje potrebno za *low-speed* i *full-speed* komunikaciju



- uočiti
 - linija je zaključena zrcalnom impedancijom na oba kraja
 - ovo sklopolje podržava standarde USB 1.0 i USB 1.1
- dodatno sklopolje potrebno za *high-speed* komunikaciju

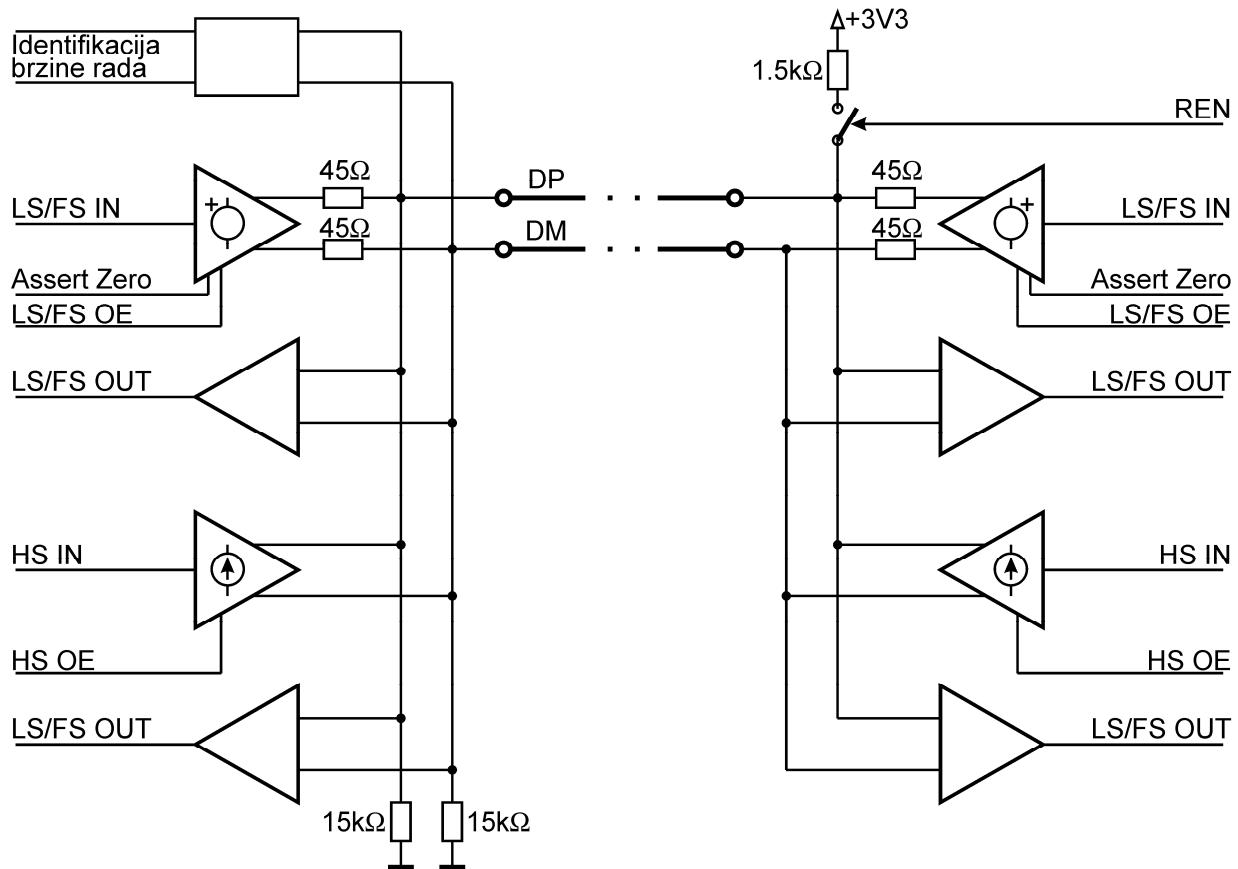


- high speed komunikacija izvodi se na slijedeći način
 - obje linije u LS/FS pojačalu daju napon od 0 V
(signal *Assert Zero*)
⇒ linije su i dalje zaključene impedancijom od 45Ω
⇒ pojačalo koje se koristi za *high-speed* mora imati strujni izlaz
- uočiti
 - kad je isključeno, sklopolje za HS komunikaciju ne narušava rad sklopolja za LS i FS komunikaciju

- brzine prijenosa određuju se
 - za *low speed* i za *full speed*
⇒ sklopoškom identifikacijom
 - za *high speed*
 - prvo se sklopoški identificira FS
 - korištenjem protokola (pregovaranje) jedinice dogovore brzinu za HS komunikaciju
- sklopoška identifikacija brzine prijenosa
 - jedinica koja se priključuje na USB ima pritezni otpornik od $15\text{ k}\Omega$ spojen prema napajanju
 - *low speed* ⇒ pritezni otpornih je na liniji DM
 - *full speed* ⇒ pritezni otpornih je na liniji DP
 - komunikacija uvijek (kod USB 2.0) počinje kao *full-speed*
⇒ postoji pritezni otpornik na liniji DP
 - kod prijelaza na *high speed*, ovaj otpornik potrebno je isključiti
⇒ potrebna je sklopka
- definiranje razine za slučaj kad jedinica nije spojena
 - kad jedinica nije spojena, aktivni su prijemni sklopovi
 - da bi bila definirana razina na njihovim ulazima, potrebno je spojiti sa DP i DM prema masi otpornike iznosa od cca $15\text{k}\Omega$

- pojednostavljena blokovska shema cjelokupnog sklopovlja

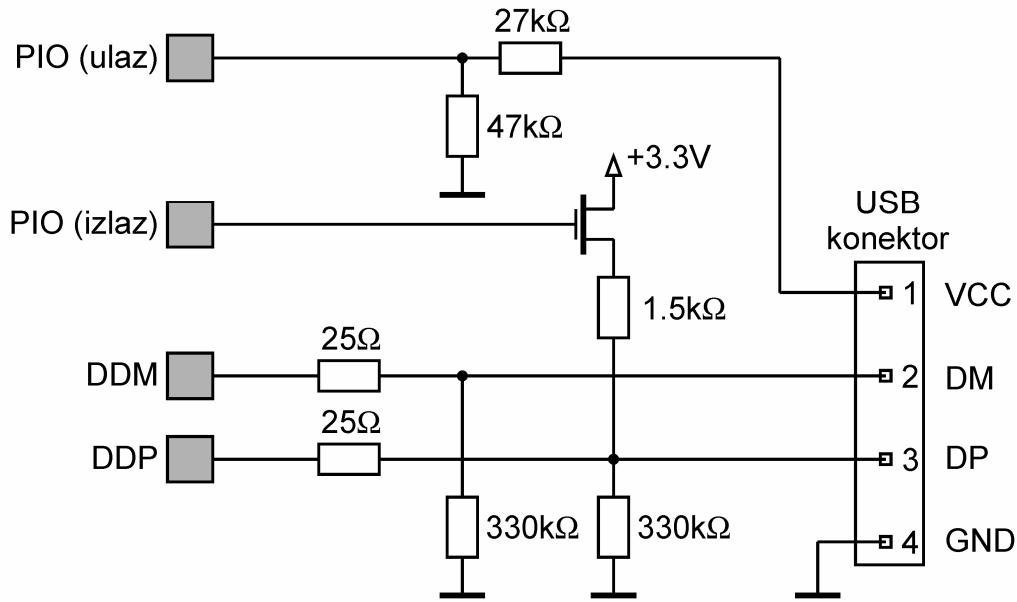
Računalo domaćin



- računalo domaćin sadrži i izvor napajanja, VCC
 - najveća struja koju izvor može dati iznosi
 - nakon što je obavljena identifikacija
 - tipično 500 mA
 - prije nego što je obavljena identifikacija
(u trenutku uključenja USB jedinice)
 - tipično 50mA

9.1.3.3 USB sučelje u mikrokontrolerima familije AT91SAM7X

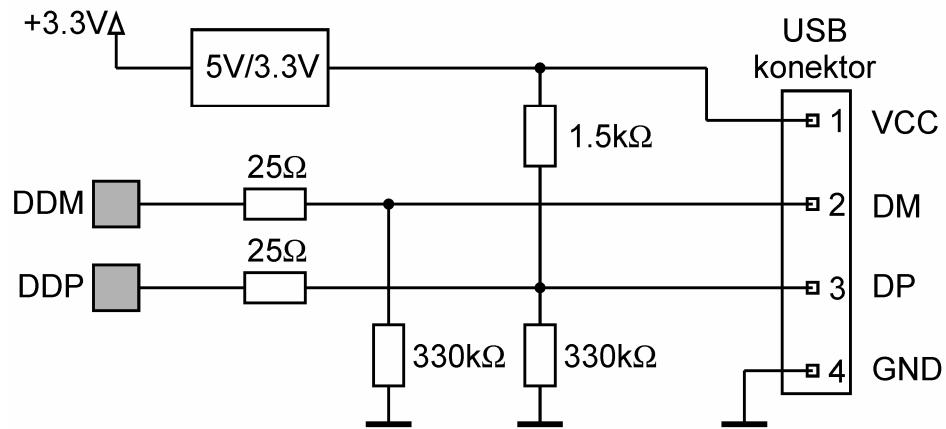
- *USB Device Port* (UDP) ove familije podržava rad s USB 2.0 standardom
- mikrokontroler sadrži sklopolje odašiljača i prijemnika
- izvana je potrebno spojiti slijedeće komponente



- uočiti
 - zaključni otpornici koje sadrže izlazna pojačala su takvi da je potrebno još dodati vanjske otpornike od 25 ohma (to specificira proizvođač mikrokontrolera)
 - otpornici iznos 330k služe za definiranje nulte razine kad mikrokontroler nije spojen na računalo domaćin
⇒ time se smanjuje potrošnja struje kad je UDP omogućen ali nije spojen
 - isključivanje priteznog otpornika na liniji DP izvedeno je pomoću izlaznog priključka PIO sklopa
 - indikacija priključenja mikrokontrolera na računalo domaćin može se izvesti pomoću ulaznog priključka PIO sklopa
⇒ ispituje se pojava napajanja na USB konektoru

- napomena

- SAM-BA ne koristi HS načina rada
⇒ ako se USB koristi samo za programiranje *Flash* memorije,
nije potrebno predvidjeti odspajanje priteznog otpornika na
liniji DDP
- USB se može koristiti i kao izvor napajanja
- vanjsko sklopovlje u tom slučaju ima oblik



9.2 Pitanja za provjeru znanja

1. Opisati komunikaciju sklopova povezanih SPI sučeljima. Opisati načine spajanja upravljača i izvršilaca. Opisati SPI sklopovlje koje sadrže mikrokontoleri familije AT91SAM7X.
2. Opisati način spajanja sklopova na TWI sabirnicu. Opisati komunikaciju sklopovla na toj sabirnici. U čemu je razlika između SPI i TWI sabirnica? S kojom poznatom sabirnicom je kompatibilna TWI sabirnica?
3. Navesti izraze za zrcalnu impedanciju i konstantu prijenosa linije. Koji oblik ti izrazi imaju za slučaj linije bez gubitaka i linije bez izobličenja? Koje je fizikalno značenje zrcalne impedancije. Objasniti pojavu refleksije na liniji. Kako se ona izbjegava? Što je faktor refleksije.
4. Nacrtati sklopovsku izvedbu USB sučelja. Objasniti ulogu pojedinih komponenata.
5. Nacrtati i opisati komponente koje je na mikrokontrolerima familije AT91SAM7X potrebno dodati izvana za ispravan rad USB sučelja. Objasniti ulogu pojedinih komponenata.

10 Završne napomene

- ovi materijali obuhvaćaju gradivo
 - arhitekturu popularnih 32-bitnih mikrokontrolera
 - razvoj sklopovlja s 32-bitnim mikrokontrolerima
 - specifičnosti programske podrške
 - alate za razvoj sklopovlja i programske podrške
- izneseno gradivo predstavlja zaokruženu cjelinu koja je dovoljna
- stečena znanja dovoljna su za samostalan daljnji rad u ovom području
- za daljnji rad preporučuju se predmeti koji nadovezuju na ovu tematiku ili su dio ovog područja
- nastavak na ovo gradivo čine predmeti koji obuhvaćaju
 - programsku podršku ugradbenih računalnih sustava
 - operacijske sustave za ugradbene računalne sustave
 - alate za razvoj digitalnih sustava
 - rad s procesorima s mekom jezgrom

11 Literatura

- [1] *AT91 ARM Thumb-based Microcontrollers*, Datasheets, Atmel 2007
- [2] *ARM7TDMI Technical Reference Manual*, Revision r4p1, ARM 2004
- [3] *ARM Architecture Reference Manual*, ARM 2005
- [4] M. Vučić, *Upotreba mikrokontrolera u ugrađenim računalnim sustavima*, FER, 2007
- [5] *AT91 ISP/SAM-BA User Guide*, Atmel 2009
- [6] *RealView Developer Kit - Compiler and Libraries Guide*, Version 2.2, ARM 2005
- [7] *uVision4 User's GuideKeil*, ARM 2005
- [8] *Procedure Call Standard for the ARM Architecture*, ARM 2009
- [9] *RealView Compilation Tools - Developer Guide*, Version 4.0, ARM 2009
- [10] *Test Access Port and Boundary-Scan Architecture*, IEEE Standard 1149.1-2001
- [11] *Universal Serial Bus Specification*, Revision 2.0, Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, 2000