Odabrane sistemske periferije LPC2138

Milan Mijalković

- Struktura magistrala
- Memorijska mapa
- Modul za ubrzan pristup FLEŠ memoriji (MAM)
- Remapiranje vektora izizetaka (MEMMAP)
- Punilac (*bootloader*)
- Programiranje FLEŠ memorije (ISP i IAP)
- Zaključana fazna petlja (PLL)
- Modul za obradu spoljašnjih zahteva za prekid

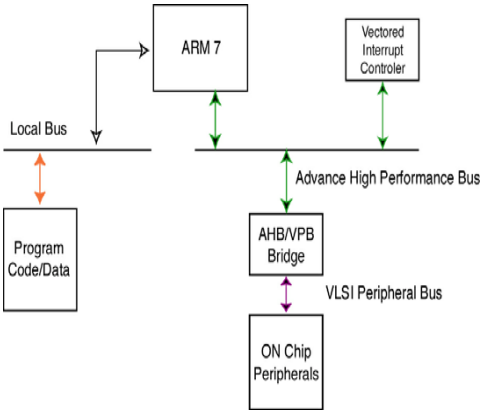
Struktura magistrala

Iako mikrokontroleri familije LPC2000 imaju više unutrašnjih magistrala, za programera, adresni prostor je linearan. To znači da sve komponente (memoriske lokacije, registri, periferijski registri...) povezane na jezgro imaju jednu od 2^{32} adresa jedinstvenog adresniog prostora sa tridesetdvobitnom adresom. Postoje tri unutrašnje magistrale: **AHB** (*Advanced High performance Bus*), **VPB** (*VLSI*[1] *Peripheral Bus*) i **MLB** (*Memory Local Bus*).

AHB je unapređena brza magistrala standardizovana od strane ARM-a. Ovo je najbrži način povezivanja jezgra ARM7 sa periferijama i na ovu magistrale je povezan vektorski kontroler prekida (VIC) i most ka VPB magistrali. Takt sa kojim rade periferije povezane za ovu magistralu je isti kao takt sa kojim radi jezgro (takt označen sa CCLK)

Sve preostale ugrađene periferije, označene kao „korisničke periferije" su povezane na **VPB**. VBP most sadrzi delilac takta jezgra, pa VPB magistrala moze da koristi sporiji takt od takta koji koristi jezgro ARM7 i AHB. Ovo je korisno iz dva razloga. Prvo, moguće je napajati korisničke periferije sporijim taktom u odnosu na takt jezgra i tako smanjiti potrošnju. Drugi razlog je to što proizvođač čipa (nezavisno od ARM standarda) ima mogućnost da doda svoju periferiju, sporiju od ARM jezgra, a da ne dođe do uskog grla na AHB magistrali. Sve ugrađene korisničke periferije mogu da rade i sa taktom od 60MHz, tako da VPB magistrala može da se podesi i da radi sa istim taktom kao i periferije na AHB magistrali. Važno je napomenuti da je posle reseta VPB delilac podešen da snizi AHB takt za četiri puta, tako da je, posle reseta, takt sa kojim rade korisničke periferije (povezane na VPB) ¼ takta CPU. Takt periferija povezanih na VPB je označen sa PCLK.

Konačno, postoji treća lokalna magistrala, **MLB** koja se koristi da se poveže CPU sa ugrađenom fleš-memorijom i ugrađenim RAM-om. Vezivanje programskog koda i podataka na AHB magistralu je bilo moguće, ali bi to izazvalo zastoje na magistrali pod nekim uslovima.

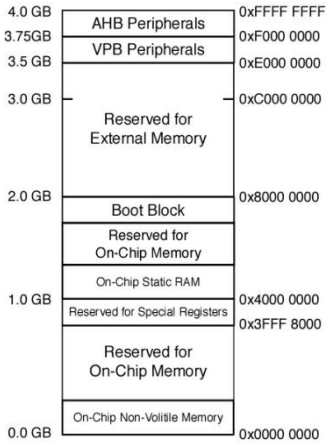


Iako, što se programera tiče, LPC2000 ima linearan adresni porstor, odnosno, programer vidi sve komponente kao da su na istoj magistrali i u istom adresnom prostoru, u stvarnosti postoji nekoliko unutrasnjih magistrala. Važno je biti svestan razlika između njih i kako utiču na performanse procesora.

Memorijska Mapa

Uprkos broju unutrašnjih magistrala, LPC2000 ima kompletno linearnu memorisku mapu. Memorijska mapa je prikazana u nastavku.

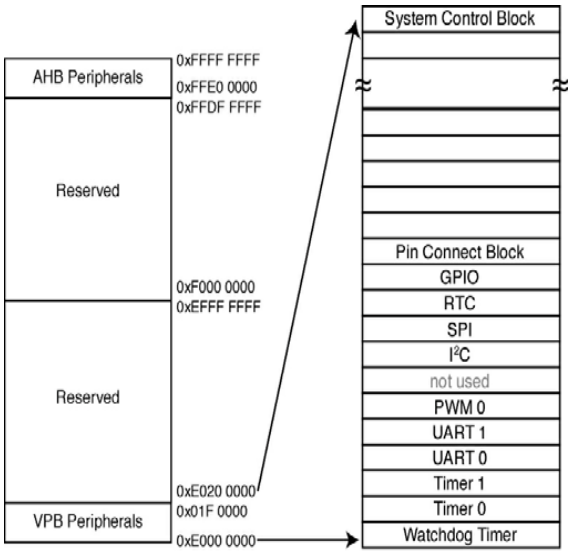
[1] VLSI (*Very Large Scale Integration*) je skraćenica koja se koristi za komponente sa velikim stepenom integracije (velikim brojem logičkih kola). U novijim verzijama kataloga ova magistrala sa naziva APB (*Advanced Pripheral Bus*), oznaka VPB se više ne koristi.



Memorijska mapa LPC-a 2000 uključuje zone za ugrađeni fleš (*on chip flash*), ugrađeni SRAM, preprogramirani punilac (*bootloader*), ugrađene periferije i spoljnu memoriju (*External memory*).

Ugrađena fleš-memorija zauzima prostor počev od 0x00000000 pa nadalje, dok prostor namenjen memoriji tipa RAM počinje od 0x4000000.

LPC2000 komponente se isporučuju sa preprogramiranim puniocem fleš-memorije (*bootloader*) i monitorskim programom za otklanjanje grešaka (*ARM real monitor debugger*). Ovi programi su smešteni u opsegu 0x7FFF D000 – 0x7FFF FFFF. Region između 0x8000000 – 0xE000000 je rezervisan za spoljnu memoriju.



Sve korisničke periferije su vezane na VPB magistralu.Svaka periferija poseduje 16K adresni opseg za svoje registre.

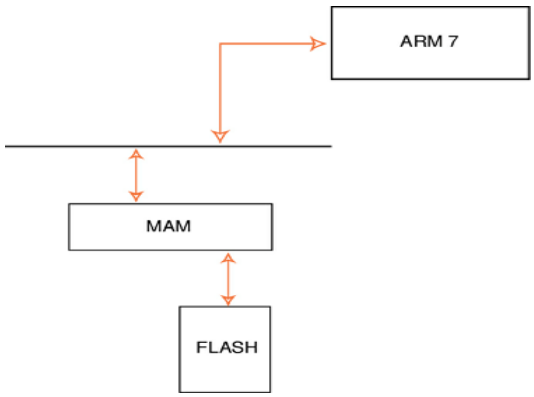
Korisničke periferije vezane na VBP su sve mapirane u području između 0xE000 0000 i 0xE020 0000 i svaka periferija zauzima po jednu 16K memorijsku stranicu. Vektorski kontroler prekida - VIC (*Vector Interrupt Controller*) je lociran na kraju adresnog prostora, počev od 0xFFFF F000 kao jedina perifriferija povezana na AHB.

Ako korisnički kod pokuša da pristupi memoriji izvan ovih regiona, ili nepostojećoj memorijskoj adresi unutar njih, CPU će izazvati izuzetak označen kao *abbort*. Ovaj mehanizam je hardverski definisan u dizajnu procesora i nemoguće ga je isključiti.

Modul za ubrzani pristup memoriji

Modul za ubrzani pristup memoriji - MAM (*Memory Accelerator Module*) je ključan za brzo izvršavanje instrukcija smeštemih u fleš-memoriji.

MAM se nalazi na lokalnoj magistrali i smešten je između fleš-memorije i ARM7 jezgra.

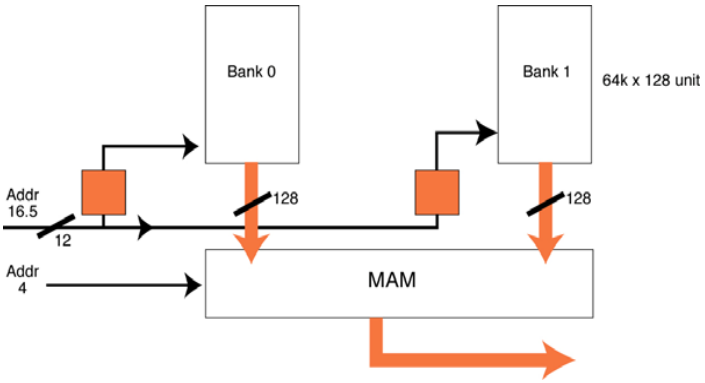


Izvršavanje instrukcija iz ugrađenog fleša je usko grlo svih ARM7 implementacija. Filips je dodao MAM koji značajno povećava performanse ARM7 CPU-a svim programima koji se izvršavaju iz fleš-memorije.

Jedna od glavnih mana u dizajnu mikrokontrolera visokih performansi u čipu je vreme pristupa fleš-memorije. ARM CPU je sposoban da radi sa taktom do 80MHz, ali ugrađeni fleš (po trenutno dostupnoj komercijalnoj tehnologiji) ima vreme pristupa od 50ns, što bi maksimalni takt ograničilo na 20MHz (što je četvrtina mogućeg takta procesora). Postoji više načina da se zaobiđe ovaj problem. Najjednostavniji je da se kritični delovi programa smeste na ram, i da se pokreću iz rama. Kako RAM ima mnogo brže vreme pristupna, sveukupne performanse bi se značajno poboljšale. Loša strana ovakvog procesa je što je integrisani ram na čipu ograničeni resurs i veoma je dragocen. Ovakvo rešenje takođe veoma ograničava veličinu koda aplikacije koju bismo mogli tako da pokrenemo.

Drugi pristup bi bio da se postavi keš (*cache*) na čip. Keš je mali deo memorije smešten između procesora i fleš-memorije, koji umapred prikuplja delove koda koji će se uskoro izvršavati. U dobro projektovanom kešu, procesor će koristii keš kada god je to moguće. Sa druge strane potpun keš je kompleksna periferija koja zahteva veliki broj kapija (gates) i kao posledcu zauzima veliki deo fizičkog prostora LPC2000. Ovo se kosi sa filozofijom ARM7 dizajna koji teži ka jednostavnosti. Druga mana potpunog keša je da vreme pokretanja koda koji koristi keš nije determinističko, i ne bi se moglo koristiti ni u jednoj aplikaciji gde je potrebna predvidljivost i ponavljanje.

Modul za ubrzani pristup memoriji – MAM je kompromis između kompleksnosti potpunog keš sistema s jedne strane, i jednostavnosti rešenja u kome je fleš-memorija direktno povezana za CPU s druge.

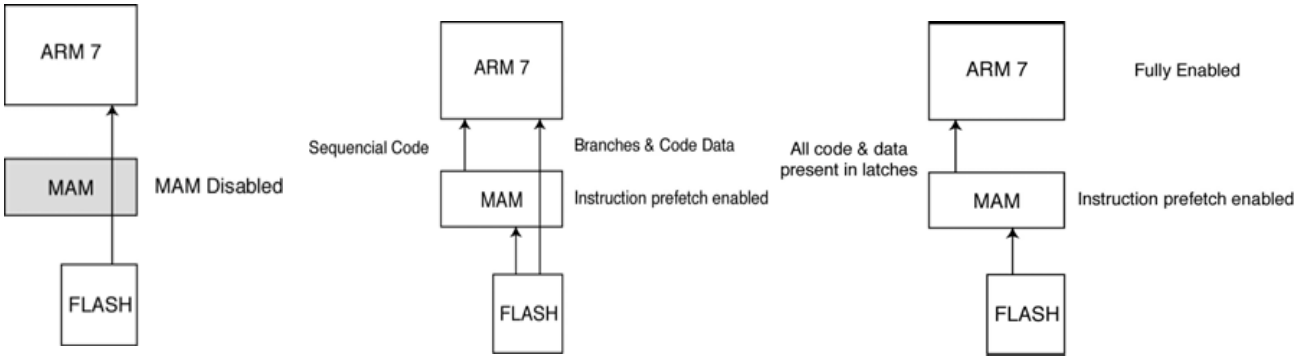


Fleš-memorija je organizovana kao dva odvojena bloka (*interleaved banks*) sa podacima širine 128 bita. Jedan flash pristup iz MAM-a dohvata četiri ARM instrukcije ili osam THUMB instrukcija.

Kao i u slučaju keš-memorije, idealno bi bilo da MAM uvek u svojoj lokalnoj memoriji sadrži sledeću instrukciju koju CPU treba da izvrši. Fleš-memorija je podeljena u dva bloka (*banks*) sa podacima širine 128 bita kojima se pristupa naizmenično. Podatak dužine 256 bita (32 bajta) se deli na dva dela, nižih 16 bajtova se smešta u blok 0 (*Bank 0*), viših 16 u blok 1 (*Bank1*). Jedan pristup fleš-memoriji može da dohvati četiri ARM instrukcije (4x32 bita) ili osam THUMB instrukcija (8x16 bita). Korisnički kôd je podeljen u dva bloka, tako da u toku izvršvanja kôda preuzetog iz jednog bloka, MAM preuzima kôd iz drugog i na raspolaganju je jezgru čim se završi prethodno preuzeti kôd. Ova tehnika radi posebno dobro sa ARM instrukcijama koje koriste bliska grananja ili kratke petlje.

Kompleksnost MAM-a je nevidljiva za korisnika, a inicijalizacija i konfiguracija se obavljaju upisom u dva registra: registar prilagođenja brzina pristupa (*timing register*) i kontrolni registar (*control register*). Postoje i dodatni statusni registri koji daju informacije o efikasnosti MAM-a. Registar prilagođenja brzina se koristi za kontrolu odnosa između takta CPU i vremena pristupa fleš-memoriji. Tri bita najmanje pozicione vrednostiu ovog registra određuju potrebni broj perioda CPU-takta za pristup fleš-memoriji (1 do 8). Ako je CPU takt (CCLK) 60 MHz, potrebne su 3 periode za pristup fleš-memoriji kojoj se ne može pristupati taktom bržim od 20 MHz. Tako se za svaka tri CPU ciklusa, preuzmu četiri instrukcije što obezbeđuje da MAM, u slučaju sekvencijalnog kôda, uvek ima raspoložive instrukcije za izvršavanje.

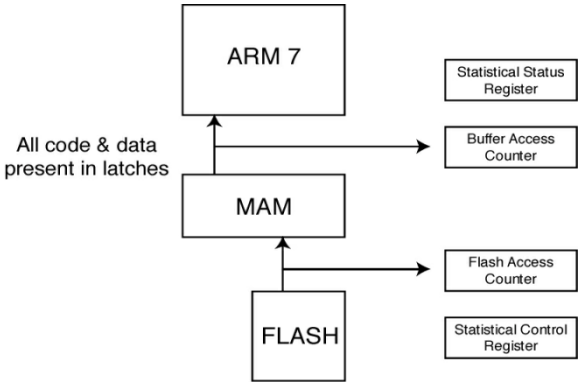
Kontrolnim registrom se određuje da li je MAM isključen, dalimično aktivan ili potpuno aktivan.



Posle reseta MAM je isključen i sav pristup kôdu i konstantama se obavlja direktno iz fleša (*MAM Disabled*) kao u blok šemi na slici levo. Moguće je da se MAM delimično aktivira tako da sav sekvencijalan kôd bude preuziman preko njega, ali se programu u slučaju grananja, kao i konstama smeštenim u fleš-memoriji, pristupa direktno (slika u sredini). Konačno, MAM može biti potpuno aktivan tako da se svi podaci iz fleš-memoroje uvek preuzimaju preko MAM (slika desno).

Razlog za što su ostavljene sve tri mogućnosti leži uglavnom u činjenici da vreme izvršavanja kôd koji se preuzima preko MAM nije unapred tačno određeno i nije uvek isto za isti kôd pa se, u primenama gde je ponovljivost vremena izvršavanja kritična, uticaj na vreme izvršavanja može umanjiti ili potpuno isključiti isključenjem MAM. Isključenjem se takođe smanjuje potrošnja.

Postoje softverske alatke – alanizatori kvaliteta kôda (*performance analyzers*) koje su u stanju da procene vreme izvršavanja kôda i sa uključenim MAM.



MAM sadrži i statističke registre koji sadrže broj pristupa fleš-memoriji kao i broj pristupa MAM-u tako da može da se izračuna efektivnost MAM-a

Za ocenu efikasnosti MAM, postoje statistički registri, bazirani oko dva brojača koji beleže broj pristupa flešu i broj pristupa u MAM baferu. Iz odnosa sadržaja ova dva registra, moguće je proceniti efikasnost MAM u konkretnoj primeni od interesa. Postoje i kontrolni registri kojima se definišu detanji šta i kako će brojači statističkih registara brojati.

Primer korišćenja MAM

Primer koda prikazan ispod startuje LPC2000 sa PLL-om podešenim na 60MHz i sa isključenim MAM-om. Osam svetlećih dioda je redom povezano za P0.16 do P0.23 (dioda vezana za P0.16 je krajnje desno) i u svakom trenutku svetli jedna od osam (u početku krajnje desna). Posle pauze definisane praznom petljom koja se ponavlja milion puta, dioda koja svetli se pomera za jedno mesto ulevo sve do poslednje kada se sekvenca ponavlja ispočetka. Za AD0 ulaz povezan je potenciometar koji može da mu dovede napon 0 do 3,3 volta. Kada je napon manji od pola (rezultat AD konverzije manji od 512, MAM je isključen, u suprotnom, MAM je uključen. Efekat MAM se vidi u brzini pomeranja položaja diode koja svetli.

```
int main(void)
{
   unsigned int delay, val;
   unsigned int FLASHer = 0x00010000;  // definisanje lokalnih prom.
   IO0DIR = 0x00FF0000;                // svi GPIO izlazni
   VPBDIV = 0x02;
   AD0CR   = 0x00270601;               // Inic. A/D: 10-bit Ain0 @ 3MHz
   AD0CR  |= 0x01000000;               // Start A/D konverzije
   while(1)
   {
   do
   val = AD0DR;                        // Čitanje A/D Data registra
   while ((val & 0x80000000) == 0);    // Čekanje kraja konverzije
   val = ((val >> 6) & 0x03FF);
   if (val > 512)
   {
      MAMCR = 0;
      MAMTIM = 0x03;
      MAMCR = 0x02;                    // Uključi MAM (potpuno aktivan)
   }
   else
      MAMCR = 0x0;                     // Isključi MAM

   for(delay=0;delay<1000000;delay++)  // Jednostavna petlja za kašnjenje...
      { ; }                            // ... milion prolazaka
   IO0PIN = FLASHer;                   // Promeni stanje izl. porta
   FLASHer = FLASHer <<1;              // Pomeri ulevo diodu koja svetli i ...
   if(FLASHer&0x01000000)              // ... ako je prošao poslednju...
      FLASHer = 0x00010000;            // ... vrati na prvu diodu.
   }
}
```

Programiranje fleš-memorije

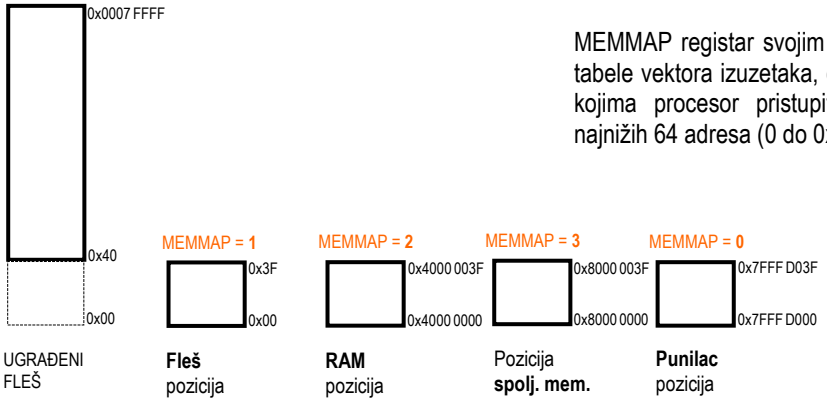
Iako je ugrađeni fleš uređen kao dva memorijska bloka sa podacima širine 128 bita, korisnik može da tretira memoriju kao kontinualni niz osmobitnih podataka i nisu potrebni nikakvi specijalni alati da bi se kôd prilagodio ovoj organizaciji prilikom za programiranja čipa. Tačnije, fleš-memorija je, sa gledišta programera, organizovana kao niz sektora veličine 4 ili 32 kilobajta koji se pojedinačno mogu brisati ili programirati. Postoji nekoliko metoda koje se mogu koristiti za programiranje ugrađene fleš-memorije. Najlakši način je pokretanje preprogramiranog punioca fleš-memorije (*bootloader*) koji omogućava da se kôd iz računara prenese preko UART0. Pri tom na računaru treba da se odvija program koji podatke iz specificirane .HEX datoteke šalje na serijski port (na primer „FlashMagic“ ili „Philips flash utility“), a mikrokontroler treba da izvrša program punioca koji podatke preko UART0 upisuje u fleš. Takođe, moguće je da se koristi JTAG razvojni alat za programiranje memorije pri čemu se podaci najpre upisuju u RAM, a potom prebacuju u fleš. Ovo je

korisno u toku razvoja zato što se punjenje programa može obaviti direktno iz razvojnog okruženja bez potrebe da se izlazi iz okruženje kako bi se pokrenuo poseban program za prenos. Takođe, JTAG komunikacija može da bude veoma brza, do 400KB/sec, tako da u slučaju velikih kodova, naročito ako se koristi spoljašnja fleš-memorija, to može biti najbolji način prenosa kôda. Konačno, moguće je u toku rada korisničkog programa pozvati program za programiranje fleša iz aplikacije (čip se fabrički isporučuje i sa ovim programom) i bilo koje podatke iz RAM-a prebaciti u fleš-memoriju.

Kontrola memorijske mape

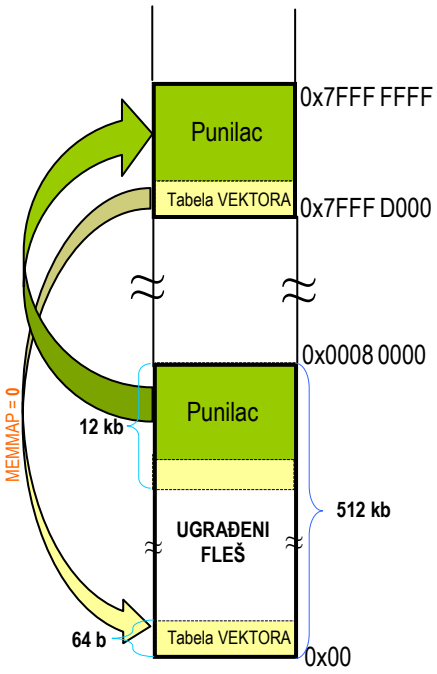
ARM7 tabela vektora izuzetaka (prvih 64 bajta memorijskog prostora, od 0x0000 0000 do 0x0000 003F) sadrži vektor reseta, vektore prekida, i vektore ostalih izuzetaka. Ova tabela se nalazi u delu adresnog prostora koji pripada ugrađenoj fleš-memoriji. Međutim, kako program ne mora da se izvršava jedino iz ugrađene fleš-memorije, već može da se izvršava i iz RAM-a, kao i iz spoljašnje memorije, bilo je nužno obezbediti mehanizam da se tabela vektora može prebaciti na početak dala adresnog prostora RAM-a ili spoljašnje memorije. Na taj način bi programi smešteni u RAM-u ili spoljašnjoj memoriji mogli da pristupaju ovoj tabeli. Sadržajem MEMMAP registra moguće promeniti (premapirati) poziciju tabele vektora.

MEMMAP registar određuje jednu od četiri moguće pozicije: fleš pozicija (*User Flash Mode*); RAM pozicija (*RAM Mode*); punilac pozicija (*Bootloader Mode*); i pozicija spoljne memorije (*External Flash Mode*). U slučaju da je odabrana RAM pozicija, vektori se dohvataju sa adresa od 0x4000 0000 do 0x4000 003F. Za poziciju spoljne memorije, vektori počinju od 0x8000 0000, a za poziciju punioca od 0x7FFF E000. Tako će se, ako je u MEMMAP registru odabrana RAM pozicija tabele vektora, kada se desi softerski prekid program skočiti na 0x4000 0008 umesto na 0x0000 0008, što bi bilo u „normalnom“ izvršavanju programa iz ugrađenog fleša. Tačnije, jezgro će se obratiti adresi 0x0000 0008 da bi pročitalo prvu instrukciju, ali će mu biti podmetnut pdatak sa 0x4000 0008 zbog remapiranja, definisanog u MEMMAP registru. U tabeli vektora, po pravilu se nalaze instrukcije bezuslovnog skoka.



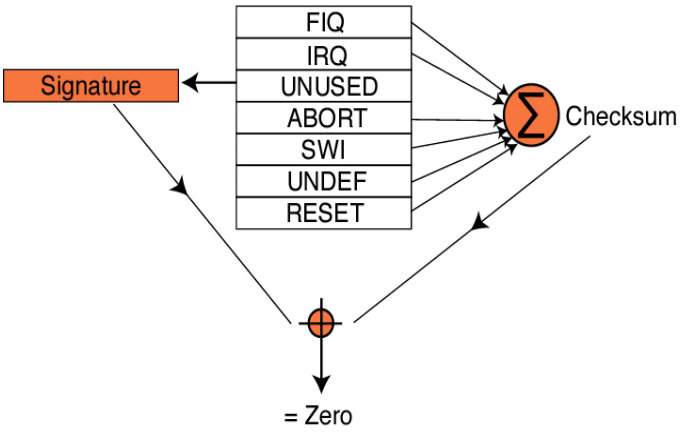
Punilac

Punilac je preprogramirani program (isporučuje se zajedno sa čipom) za punjenje ugrađenog fleša korisničkim programom. Njegov kôd zauzima 12 kb i originalno je smešten na vrhu ugrađene fleš-memorije ali je remapiran tako da mu procesor može pristupiti i u opsegu adresa 0x7FFF D000 do 0x7FFF DFFF. Posle **svakog** reseta aktivna je Punilac-pozicija tabele vektora (MEMMAP=0), tako da procesor, u slučajevima izuzetaka dok traje program za punjenje, pristupa vektorima koji pripadaju puniocu i koji su fabrički programirani zajedno sa njim. Reset vektor u ovoj tabeli sadrži naredbu skoka na prvu instrukciju punioca.



Deo memorijske mape čipa sa 512kb ugrađenog fleša: Punilac, zajedno sa pripadajućom tabelom vektora, originalno zauzima poslednjih 12 kilobajta ugrađene fleš-memorije. Posle reseta, ovaj deo mape je remapiran tako da zauzima prostor prostor od 0x7FFF D000 do 0x7FFF DFFF. Kako je tada MEMMAP=0, 64 bajta pripadajuće tabele vektora je remapirano na početak adresnog prostora pa je u toku izvršavanja punioca, aktivna tabela vektora izuzetaka koja pripada njemu.

Posle reseta uvek se pokreće punilac. On najpre proverava da li je uzrok reseta bio hardverski ili softverski (reset zbog isteklog *wachdog* brojača). U slučaju softverskog reseta program se očigledno već nalazi u flešu pa nije potrebno prenositi ga ponovo. Ako je reset harverski, neophodno je odlučiti da li trebada se nastavi prosedura punjenja fleša (prenos korisničkog programa iz računara gde je razvijen do mikrokontrolera) ili se korisnički program već nalazi u fleš-memoriji pa konrolu treba prepustiti njemu. Uloga da razlikuje ova dva slučaja poverena je GPIO portu P0.14. Logička nula na ovom ulazu znači da korisnički program još nije prenet i da treba nastaviti sa programom za prenos. Kada jednom napuni fleš-memoriju, korisnik mora (hardverski) da dovede logičku jedinicu (visok naponski nivo) na ovaj ulaz. Programu punioca, logička jedinica na P0.14, znači da treba da prekine sa radom i prepusti kontrolu korisničkom programu. Međutim pre prepuštanja kontrole korisničkom programu, punilac proverava još jednom da li se korinički program zaista nalazi u flešu. Ova provera je uvedena kako bi se izbegla mogućnost da je P0.14 postavljen (ako je, na primer, vezan za preklopnik koji prebacuje korisnik), a program ipak nije prenet. Od korisničkog programa se zahteva da ostavi „potpis“ (signature) kada se prebaci u fleš. Potpis je 32-bitni broj koji se upisuje na adresu 0x14 (što je neiskorišćeni vektor u tabeli vektora izuzetaka). Korisnik treba da upiše broj koji u zbiru sa ostalim 32-bitnim podacima iz tabele vektora (adrese od 0 do 0x1F) daje nulu.



Potpis (*signature*) upisan na mesto neiskorišćenog vektora na adresi 0x14, u zbiru sa sadržajima ostalih vektora mora da da nulu da bi se kontrola prepustila korisničkom programu.

Dakle, prilikom razvoja programa, kada su poznati podaci koji se upsuju u ostale memorijske lokacije tabele vektora (osim 0x14), potrebno je sabrati sve te 32-bitne podatke i drugi komplement zbira upisati kao konstantu na 0x14.

Punilac će sabrati osam 32-bitnih brojeva, počev od adrese 0, i ako je zbir nula, prepustiti kontrolu korisničkom programu tako što će najpre promeniti poziciju tabele vektora na „Fleš poziciju“ (upisom jedinice u MEMMAP), a zatim skočiti na reset vektor tako što upiše nulu u programski brojač.

Ako zbir nije nula, punilac smatra da se korisnički program još ne nalazi u flešu i nastavlja proceduru za punjenje fleša. Dakle, korisnički program se ne može pokrenuti bez potpisa. Programska okruženja (kao što je IAR-ovo) imaju alatke koje pored pripreme vektora izuzetaka pripremaju i potpis tako da programer ne mora da izračunava zbir niti da vodi računa o tome. Svaki put kada se iz okruženja napuni fleš, potpis se upisuje zajedno sa ostalim delovima programa.

Asemblerski kôd za popunu tabele vektora je dat ispod. Vetori su popunjeni instrukcijom bezuslovnog skoka na početnu adresu rutine za opsluživanje izuzetka.

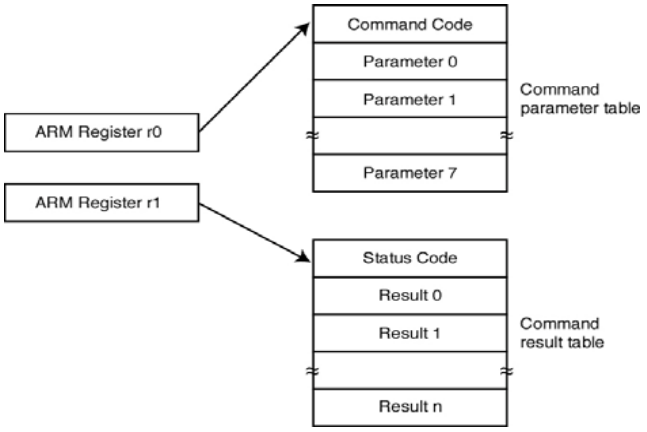
```
LDR   PC, Reset_Addr
LDR   PC, Undefined_Addr
LDR   PC, SWI_Addr
LDR   PC, Prefetch_Addr
LDR   PC, Abort_Addr
.long 0x1B28AC0F     // potpis:drugi komplement zbira 32-bitnih sadržaja ostalih vektora
LDR   PC, IRQ_Addr
LDR   PC, FIQ_Addr
```

Programiranje fleša u toku rada - IAP

Punilac ima isnovnu uloga da napuni fleš-memoriju podacima koje prima od nadređenog računara preko UART0 periferije, postoji i druga uloga ovog koji se isporučuje zajedno sa čipom. Naime, moguće je uz korišćenje IAP (*In Application Programing*) funkcije punioca prebaciti podatke iz RAM-a u fleš pozivom ove funkcije iz korisničkog programa. Komande IAP-a se sastoje iz kôda komade (*Command Code*) praćene parametrima čiji broj zavisi od komande i kreće se od 0 do 4.

Na primer, komanda za brisanje sektora ima kôd 52 i tri parametra: redni broj početnog sektora koji se briše (0 do 26), redni broj poslednjeg sektora koji se briše (0 do 26) i frekvenciju takta procesora u kHz. Tako bi komanda 52, 12, 14, 12000 prosleđena IAP funkciji obrisala sektore 12,13 i 14 u procesoru koji radi sa taktom 12 MHz. Redni brojevi sektora se mogu pronaći u prirukniku.

Da bi se komanda prosledila funkciji potrebno je niz brojeva koji predstavlja komandu upisati u negde RAM i početnu adresu niza (pokazivač na niz) ubaciti u registar R0 pre poziva IAP funkcije. Pre poziva treba takođe u registar R1 upisati početnu adresu RAM-a gde će komanda vratiti rezultat. Rezultat može da bude dugačak do 3 reči, a komanda do 5.



IAP funkciji treba (kroz registre R0 i R1) proslediti dva pokazivača: Prvi (nalazi se u R0) je pokazivač na RAM gde je korisnik, pre poziva, upisao kôd željene komande praćen parametrima; i drugi, (u R1) koji je pokazivač na slobodan prostor u RAM-u gde će funkcija vratiti status i rezultate.

Početna adresa IAP funkcije je 0x7FFF FFF1 i funkcija radi u THUMB režimu. Prilikom poziva, funkcija očekuje povratnu adresu u registru veze (*link register*) - R14. Za poziv funkcije iz C-jezika najjednostavnije je koristuti standardni C-poziv funkcije kojoj bi se asemblerskom instrukcijom izvršio skok na adresu početka IAP. Standardi C-poziv obezbeđuje korišćenje R14 registra a asembler skok na apsolutnu adresu datu kao broj.

Pozivom sledeće funkcije gde bi prvi parametar bila adrese u ramu gde je smeštena komanda, a treći parametar broj 0x7FFF FFF1 pozvala bi se IAP funkcija:

```
void IAP_fukcija (unsigned long *komanda, unsigned long *rezultat, unsigned entry)
{
      asm ("mov r15, r2");
}
```

Postoji dogovor (u formi standarda) u vezi sa prenosom parametara po kome se, u ARM arhitekturi, do 4 parametra prenose preko registara R0 do R3, redom, po redu navođenja u listi formalnih parametara u prototipu funkcije. Po tome bi prvi navedeni parametar (`unsigned long *komanda`) bio prenet preko R0, drugi (`unsigned long *rezultat`), preko R1, i treći (`unsigned entry`), preko R2. Kako je prilikom poziva treći parametar početna adresa IAP rutine, to će se ta adresa preneti funkciji preko registra R2. Jednostavna asemblerska instrukcija prebacuje sadržaj R2 u programski brojač (R15), to jest, izaziva bezuslovni skok na IAP rutinu. Po završetku odvijanje programa će se nastaviti od adrese sačuvane u registru veze R14 (*link register LR*), što jeste adresa odakle je pozvana `IAP_funkcija`.

Treba voditi računa o tome da IAP funkcija koristi 32 bajta sa dna (najviše adrese) memorijskog prostora memorije tipa RAM. Podatke smeštene na tim adresama (na tim adresama se obično može naći stek) treba sačuvati pre poziva i obnoviti po završetku.

Drugi pristup je definisanjem pokazivača na funkciju:

```
unsigned long komanda[5];      // rezervisanje prostora za komandu i rezultat
unsigned long rezultat[3];
```

Definicija tipa IAP. To je pokazivač na funkciju koja prima dva parametra a ne vraća ništa.

```
typedef void (*IAP)(unsigned long [],unsigned long[]);
IAP Poziv_iap;                 // definisanje promenljive tipa IAP (pokazivača na funkciju)
                               // Sada je promenljiva "Poziv_iap" pokazivač na funkciju
```

Dodela vrednosti pokazivaču i promena tipa pokazivača:

```
Poziv_iap =(IAP)0x7FFF FFF1;  // Početna adresa IAP se menja u tip pokazivača na funkciju
```

Funkcija se može pozvati instrukcijom:

```
Poziv_iap (komanda, rezultat);
```

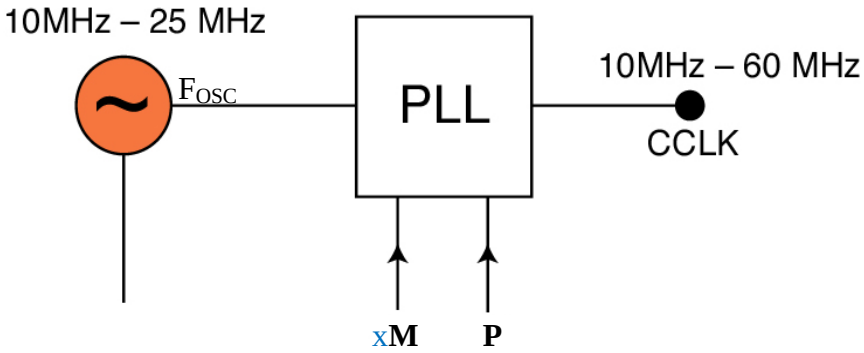
Pre poziva treba inicijalizovati komandu. Na primer:

```
komanda[0]=52; komanda[1]=13; komanda[2]=15; komanda[4]=12000;
Poziv_iap (komanda, rezultat);
```

Ovaj deo programa bi pozvao IAP komandu za brisanje više sektora, čiji je kôd 52 i koja ima tri parametra: prvi parametar je broj sektora odakle treba početi brisanje, drugi je broj poslednjeg sektora koji treba obrisati, i treći parametar je frekvencija CPU u kilohercima. Dakle, ova komanda bi obrisala sektore 13, 14 i 15 i status, da li je brisanje bilo uspešno (jednu od četiri moguće vrednosti koje su navedene u priručniku), vratila kroz promenljivu `rezultat[0]`.

Fazno zaključana petlja - PLL

Blok fazno zaključane petlje – PLL (*Phase Locked Loop*) služi da ušestanost takta dovedenog na ulaz uveća (umnoži M puta, gde je M celi broj) do maksimalnih 60 MHz. Na ulaz PLL bloka se dovodi takt ušestanosti 10 MHz – 25 MHz, bilo kao spoljni signal ili signal iz ugrađenog kristalnog oscilatora. Time se omogućava brži takt za ARM7 CPU i periferije (unutar čipa) u odnosu na učestanost sa kojom radi kristalni oscilator. Signali visoke učeatanosti sa kojom radi CPU se zbog toga ne pojavljuju van čipa. Ovakav pristup može značajno smanjiti elektro-magnetne smetnje koje izaziva ceo mikroračunar. Pored toga, izlazna učestanost PLL se moze menjati u toku izvršavanja programa čime se može uticati na potrošnju zavisno od potreba primene.



PLL se koristi da umnoži učestanost ulaznog takta do 60 MHz. Programira se unošenjem dve konstante: **M** i **P**

Da bi se definisao sistemski takt CCLK sa kojim radi CPU i AHB magistrala potrebno je definisati dve konstante označene sa **M** i **P**. Konstanta **M** je množilac ulazne učestanosti tako da je, ako obeležimo sa F_{OSC} ulaznu učestanost, a sa CCLK učestanost CCLK takta, učestanost itlaznog signala iz PLL bloka data sa:

$$CCLK = \mathbf{M} \times F_{OSC}$$

Konstanta **P** je u vezi sa principom rada PLL. Naime, PLL sadrži strujno-kontrolisani oscilator čija učestanost (označena sa Fcco) mora da bude u opsegu 156 MHz – 320 MHz kako bi ispravno radio. Ušestanost Fcco je vezana sa učestanošću CCLK relacijom:

$$Fcco = CCLK \times 2 \times \mathbf{P}$$

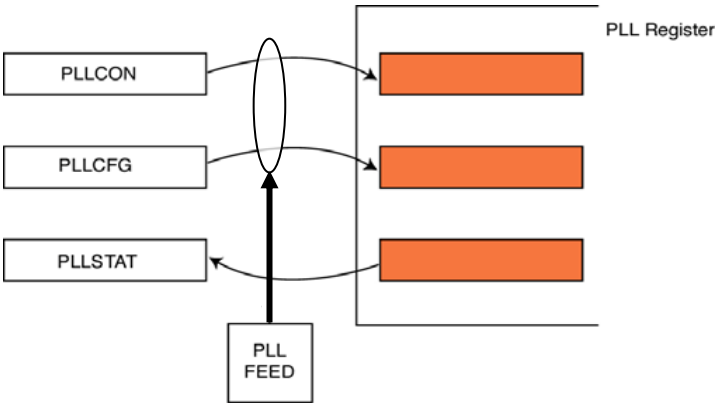
Korisnik treba najpre da odabere **M** (može biti ceo broj između 1 i 32, odnosno, zbog ograničenja najniže ulazne učestanosti, od 1 do 6) kako bi dobio željenu učestanost takta CPU. Potom, kada je ušestanost takta CCLK određena, treba odabrati **P** (može biti 1, 2, 4 ili 8) tako da (2**P** x CCLK) bude u traženom opsegu – između 156 i 320 MHz.

Na našoj razvojnoj pločici je oscilator učestanosti 12MHz i da bi se dobila maksimalna učestanost takta CPU od 60MHz, potrebno je da konstanta **M** bude 5 jer je 5x12 MHz = 60 MHz. Potrebna Vrednost **P** treba odrediti tako da 2x**P**x60MHz bude između 156 i 320 MHz. Vrednost koja zadovoljava ovaj zahtev je **P**=2.

Vrednosti M i P se definišu upisom u konfiguracioni registar PLLCFG. Konstantu P definišu dva bita ovog registra (PLLCFG.PSEL) a konstantu M pet bita (PLLCFG.MSEL). Odnos upisanih bita i vrednosti M i P opisan je u priručniku. Kontrolni registar PLLCON sadrži bit pomoću koga se može uključiti ili isključiti ceo PLL blok i bit pomoću koga se određuje da li je izlaz PLL koji stabilno radi, povezan za CCLK ili se takt CCLK dobija direktno iz oscilatora (a ne iz PLL).

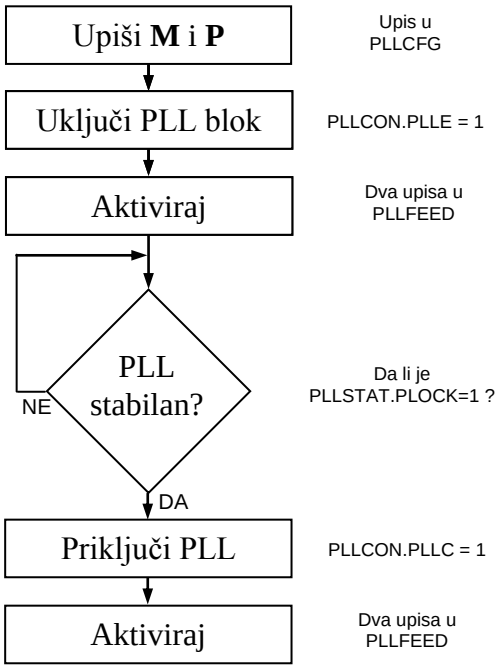
Upisom u konfiguracioni i kontrolni registar, novoupisane vrednosti ne postaju aktivne odmah, već tek posle upisa 0xAA i 0x55 u registar aktiviranja (PLLFEED) i to po zahtevanom redosledu. Dakle, korisnik mora najpre inicijalizovati PLLCFG i PLLCON, ali vrednosti M, P i kontrolnih bita

postaju „aktivne“, odnosno, imaju efekta na rad PLL, tek kada ih softver aktivira upisom 0xAA u PLLFEED registar i, odmah u sledećem ciklusu, upisom 0x55 u isti registar. Između ova dva upisa nužno je zabraniti prekide jer će nove vrednosti konfiguracionog i kontrolnog registra postati aktivne jedino ako su upisi 0xAA i 0x55 neposredno jedan za drugim.



Kontrolni i konfiguracioni registri PLL-a se mogu programirati bilo kada, ali će nove vrednosti imati efekta tek posle „aktiviranja“, odnosno, posle upisa odgovarajućih podataka u PLLFEED registar i to u zahtevanom redosledu.

Kada se upišu vrednosti P i M u komfiguracioni registar, uključi PLL blok, i ove vrednosti aktiviraju, potrebno je neko vreme da se oscilacije, definisane novim parametrima, na izlazu PLL stabilizuju. U literaturi se ova stabilizacija izlaznih oscilacija naziva „zaključavanjem“ (*lock*) petlje. Informacija da se fazna petlja zaključala, to jest da je signal na izlazu PLL stabilan, postoji u statusnom registru PLLSTAT (statusni bit PLLSTAT.PLOCK). Izlaz PLL se ne sme priključiti za CCLK dok izlazni signal nije stabilan.

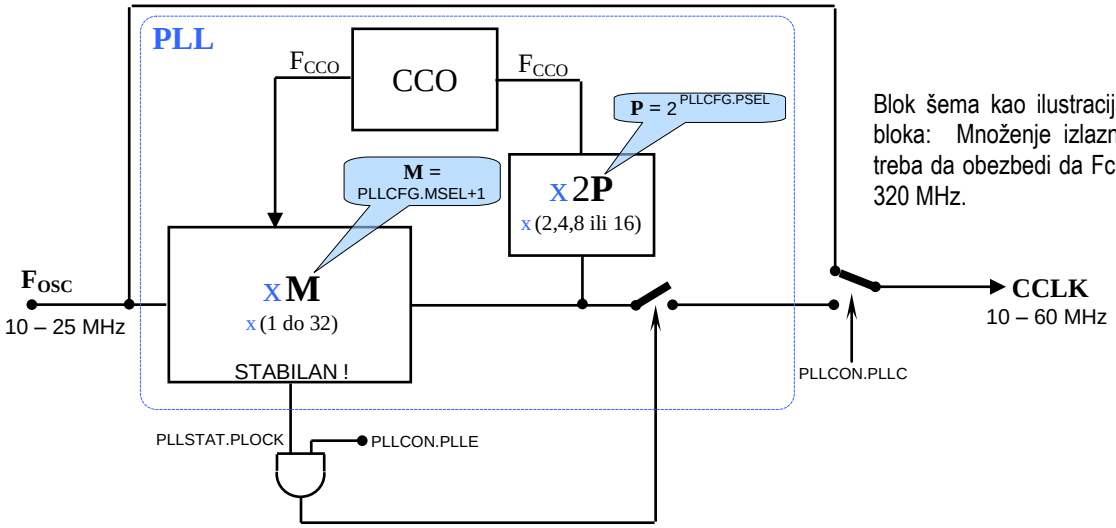


Redosled upisa parametara u PLL: Prilikom svake promene konstanti M i P nužno je slediti ovaj redosled upisa: Upisati konstante M i P i uključiti PLL blok, aktivirati ove promene, sačekati da se PLL stabilizuje, priključiti CCLK na izlaz PLL i aktivirati ovo priključenje.

Posebnu pažnju treba obratiti kod upisa vrednosti u PLLCFG registar. Pogrešnim određivanjem M i P moguće je uvesti PLL u nedefinisane i nestabilne režime rada pa samim tim i učiniti rad celog mikrokontrolera nepouzdanim. Greška može nastati i zbog činjenice da je zapis MSEL dužone pet bita (od b0 do b4), pa dva bita (b5 i b6) koji čine PSEL nisu u granicama polubajta. Zbog toga se iz heksadecimalnih cifara koje se upisuju u registar ne vidi direktno vrednost MSEL i PSEL već kombinacija ovih vrednosti. Da je MSEL dužine 4 bita, tada bi dve heksadecimalne cifrre upisane u PLLCGF bile baš PSEL i MSEL i greške pri upisivanju bi bile manje verovatne.

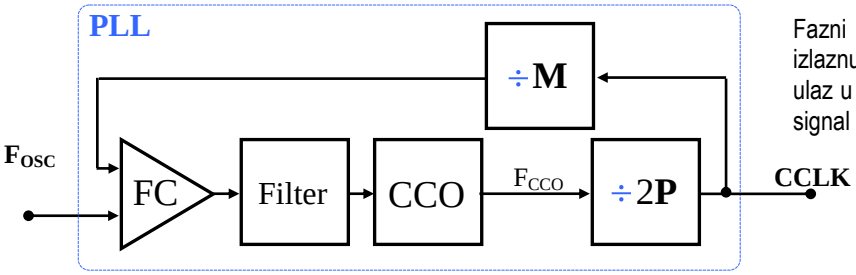
U režimima smanjene potrošnje, PLL blok se isključuje i odvaja od CCLK. Buđenje iz ovih režima neće automatski uključiti PLL tako da se opisani niz operacija uključenja PLL mora obaviti prilikom svakog izlaska iz režima smanjene potrošnje.

Sledeća slika ilustruje rad PLL i sa prethodnom, gde je naveden redosled operacija da bi se ostvarila funkcija PLL, predstavlja skraćeni kompletan opis rada ovog veoma značajnog bloka.



Blok šema kao ilustracija principa rada PLL bloka: Množenje izlazne učestanosti sa 2**P** treba da obezbedi da Fcco bude između 156 i 320 MHz.

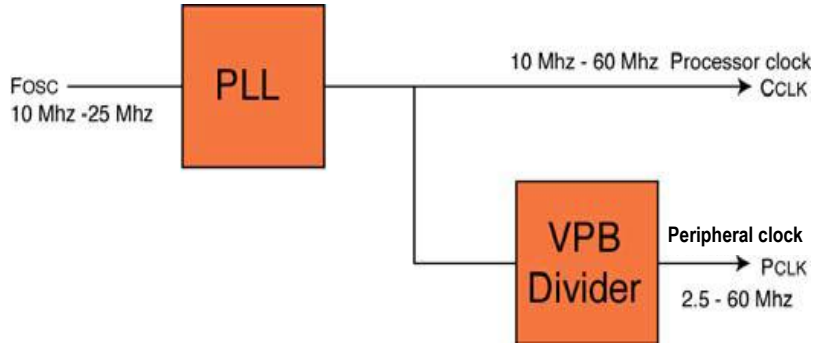
Zaključana fazna petlja sadrrži dva delitelja učestanosti (sa M i sa 2P). Delitelj sa M deli učestanost izlaznog signala PLL i poredi ga sa ulaznim. Delitelj sa 2P obezbeđuje da CCO radi na učestanostima koje moraju biti veće od 60 MHz, kolika je maksimalna učestanost CCLK. Princip rada PLL je prikazan na sledećoj slici:



Fazni komparator FC poredi ulaznu učestanost i izlaznu podeljenu sa M. Razlika se filtrira i predstavlja ulaz u strujno kontrolisani oscilator (CCO) čiji je izlazni signal učestanosti Fcco.

VPB delitilj (*VLSI Peripheral Bus divider*)

Izlaz iz PLL-a je CCLK – takt sa kojim rade ARM7 CPU i AHB magistrala. Periferije su na zasebnoj – VPB magistrali i koriste takt označen sa PCLK.



Signal CCLK služi kao takt za CPU i AHB magistralu (*Processor clock*) dok je PCLK takt kojim se napajaju periferije (*Peripheral clock*) i on se dobija deljenjem učestanosti CCLK signala sa 1,2 ili 4 u VPB delitelju (*VPB divider*)

Ovaj takt se dobija deljenjem učestanosti CCLK sa 1, 2 ili 4 VPB deliteljem (*VPB Divider*). Registar VPB delitelja se može promeniti u svakom trenutku, a posle reseta je postavljen 4. Trenutno, sve periferije na LPC2000 mikrokontrolerima mogu da rade na 60MHz.

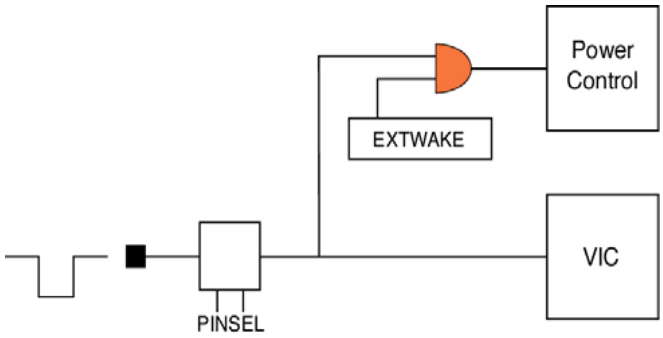
Primer koda: PLL i VPB konfiguracija

U nastavku se nalazi kôd inicijalizacije PLL za CCLK = 60 MHz i PCLK takt učestanosti 30 MHz ako je učestanost kristalnog oscilatora 12 MHz:

```
void init_PLL(void)
{
    PLLCFG = 0x00000024;                // M=5, P=2
    PLLCON = 0x00000001;                // Uključi PLL blok
    PLLFEED = 0x000000AA;               // Aktiviranje...
    PLLFEED = 0x00000055;               // ...poslednjih upisa
    while (!(PLLSTAT & 0x00000400));    // čekanje stabilizacije
    PLLCON = 0x00000003;                // Priključenje PLL
    PLLFEED = 0x000000AA;               // Aktiviranje...
    PLLFEED = 0x00000055;               // ...poslednjeg upisa
    VPBDIV = 0x00000002;                // VPB delitelj = 2
}
```

Ulazi za spoljašnji zahtev za prekid

Mikrokontroler sadrži do četiri ulazna signala EINT0 do EINT3 pomoću kojih se spolja može postaviti zahtev za prekid. Pomoću bloka za dodelu funkcija nožicama procesora (PINSEL blok) svaki od signala spoljašnjih zahteva se može postaviti bar na dve nožice mikrokontrolera. Na primer, funkcija EINT0 se može dodeliti P0.1 i P0.16, a funkciju EINT3 mogu da dobiju čak tri nožice (P0.9, P0.20 i P0.30). Detalji o tome kako radi zahtev za prekid kada postoji na više od jedne nožice mogu se pronaći u priručniku. Spoljašnjim zahtevima za prekid se upravlja pomoću tri registra. EXMODE i EXPOL registri definišu da li je ulaz osetljiv na logičko stanje signala (*level sensitive*) ili na ivicu (*edge sensitive*) i koji polaritet predstavlja zahtev (da li logička nula ili jedinica, odnosno, da li rastuća ili opadajuća ivica. EXWAKE registar definiše koji od spoljašnjih prekida može da vrati procesor iz režima smanjene potrošnje. Pored ovih kontrolnih registara postoji još i statusni registar koji sadrži informaciju preko kog od ulaznih signala je zahtevan prekid. Registri su opisani u priručniku



Ulazi za spoljašnji zahtev za prekid se jednostavno konfigurišu i mogu dobro poslužiti kao izvor prekida početnicima koji prvi put eksperimentišu prekidnom strukturom LPC2000 serije.