

# Dokumentacija za DSC

## TMS320F28335

### Sadržaj

UVOD.....	2
1. HARDVER .....	2
1.1 Tabela pinova.....	2
1.2 Opis džampera .....	2
1.3 Šema kontrolne kartice sa objašnjenjima.....	3
1.4 Šema <i>docking station-a</i> sa objašnjenjima .....	4
2. UPUTSTVO ZA KORIŠĆENJE .....	5
2.1 Generalno uputstvo za sve vežbe .....	5
2.2 Primeri.....	7
2.2.1 Vežba 1: GPIO .....	7
2.2.2 Vežba 2: Timer .....	8
2.2.3 Vežba 3: Interrupt .....	8
3. SOFTVER.....	8
3.1 Nalaženje softvera na DVD-u .....	8
3.2 Objašnjenje softvera u vežbama .....	9
3.2.1 Vežba 1:GPIO .....	9
3.2.2 Vežba 2: Timer .....	10
3.2.3 Vežba 3: Interrupt .....	13

## UVOD

Ovaj dokument je načinjen da korisniku olakša rad sa DSP-om F28335. Sastoji se od tri poglavlja. U prvom poglavlju je opisan hardver. U drugom poglavlju je dato je uputstvo za primenu primera koji su dati uz ovu dokumentaciju. U trećem poglavlju nalazi se softver, tj. dat je i detaljno objašnjen softver primera koji se koriste u drugom poglavlju.

## 1. HARDVER

### 1.1 Tabela pinova

Tabela pinova je data u folderu HARDVER kao *word* dokument pod nazivom "28335PinTable". Ona nije stavljena u ovaj dokument zbog prostora (preglednosti).

### 1.2 Opis džampera

DSC F28335 ima fabrički isprogramiran boot ROM. Boot ROM signali (signali koji zavise od rasporeda *jumper-a* ili *switch-eva*) služe da daju softveru u boot ROM-u informaciju koji boot mod da koristi pri startu (paljenje DSC-a ili restartovanje). U donjoj tabeli je prikazano kako korisnik treba da postavi *jumpere* da bi dobio određeni boot mod:

MODE	GPIO87/XA15	GPIO86/XA14	GPIO85/XA13	GPIO84/XA12	MODE <sup>(1)</sup>
F	1	1	1	1	Jump to Flash
E	1	1	1	0	SCI-A boot
D	1	1	0	1	SPI-A boot
C	1	1	0	0	I2C-A boot
B	1	0	1	1	eCAN-A boot
A	1	0	1	0	McBSP-A boot
9	1	0	0	1	Jump to XINTF x16
8	1	0	0	0	Jump to XINTF x32
7	0	1	1	1	Jump to OTP
6	0	1	1	0	Parallel GPIO I/O boot
5	0	1	0	1	Parallel XINTF boot
4	0	1	0	0	Jump to SARAM
3	0	0	1	1	Branch to check boot mode
2	0	0	1	0	Branch to Flash, skip ADC calibration
1	0	0	0	1	Branch to SARAM, skip ADC calibration
0	0	0	0	0	Branch to SCI, skip ADC calibration

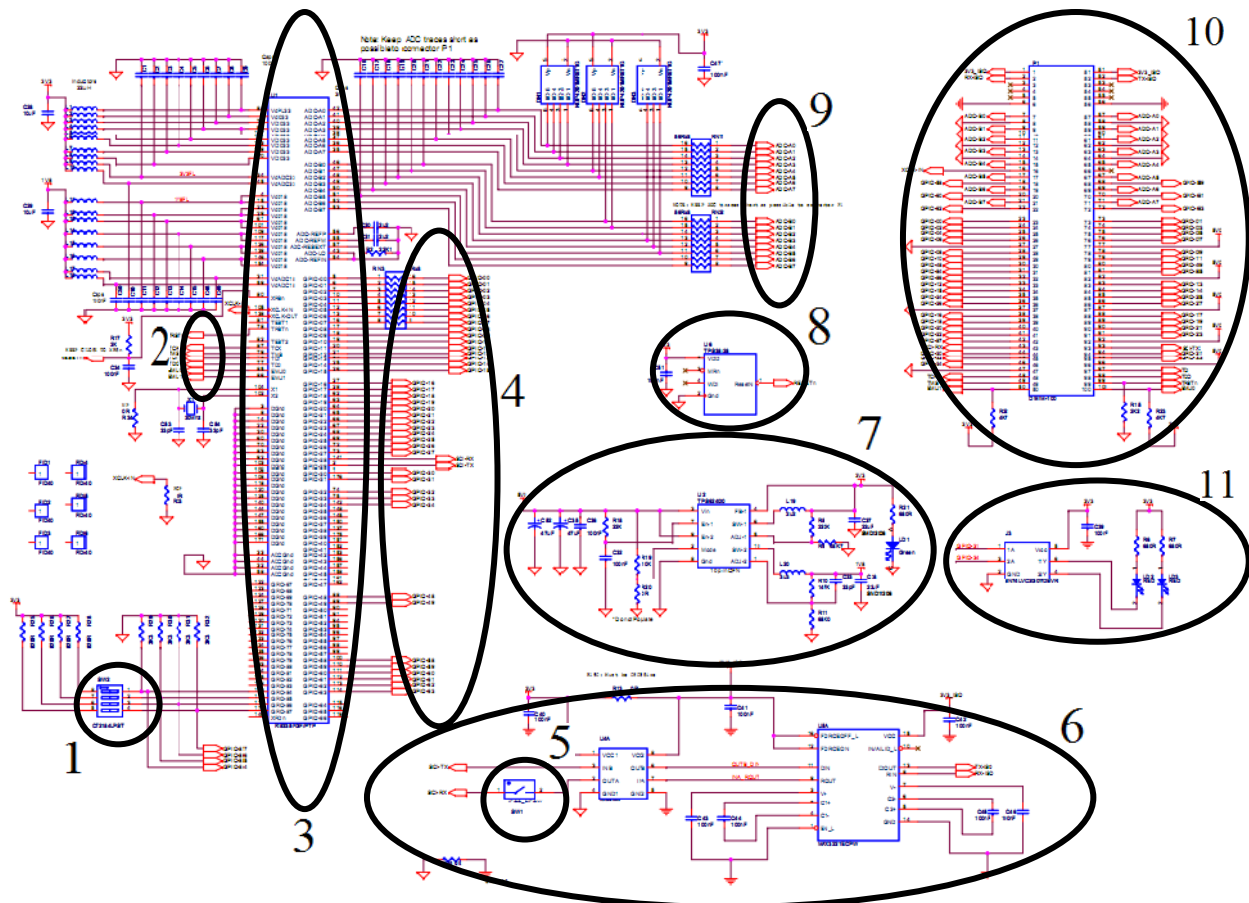
DSC F28335 ima 2 switch-a: SW1 i SW2. Prvi *switch*, SW1, kontroliše RS-232 vezu koja se nalazi na *control card-u*. On određuje da li je RS-232 veza dozvoljena ili zabranjena. U poziciji ON (*switch* pomeren naviše, u ovom je položaju *switch* kada se isporučuje iz fabrike) RS-232 komunikacija je dozvoljena.

Ako je SW1 poziciji OFF (*switch* spušten), dozvoljava se pinu GPIO-28 da se koristi kao GPIO. Komunikacija putem serijske veze je i dalje moguća, ali je sada potreban dodatni, eksterni transiver, kao što je FTDI – FT2232D čip. Ovo se primarno koristi za komunikaciju preko konvertera USB-serijska veza, koji je u sklopu *onboard XDS100 JTAG* emulacije na mnogim razvojnim pločicama serije C2000.

*Switch* SW2 kontroliše u kom se boot modu procesor nalazi pri startu. Ispod je prikazana tabela sa najčešće korišćenim položajima:

Position 1 (GPIO-84)	Position 2 (GPIO -85)	Position 3 (GPIO-86)	Position 4 (GPIO -87)	
0	1	1	1	SCI-A Boot
1	0	1	1	SPI-A Boot
1	1	1	1	(default) Get mode; the default get mode is boot from FLASH

### 1.3 Šema kontrolne kartice sa objašnjenjima



Na gornjoj šemi se mogu zapaziti:

1. SW2 koji je detaljno objašnjen u prethodnom poglavlju
2. Pinovi za emulaciju. Ove pinove koristi *onboard XDS100 JTAG* emulator ili neki drugi emulator, ako je povezan, da bi preuzeo kontrolu nad procesorom. Emulator se najčešće koristi da bi se u toku rada pročitalo stanje određenih registara procesora, bez prekidanja ili ometanja osnovnog programa koji se izvršava u tom trenutku.
3. Procesor DSP F28335. Procesor se sastoji od CPU (*Central processing unit*) i određenih periferija koje se koriste za izvršavanje određenih operacija bez oduzimanja procesorskog vremena CPU.
4. GPIO pinovi. Ovi pinovi su opisani u poglavlju "Tabela pinova".
5. SW1 koji je detaljno objašnjen u prethodnom poglavlju.

6. Transiver koji se koristi pri serijskoj komunikaciji da prevede signal sa pinova za serijsku vezu sa 3.3V (jer na 3.3V radi i procesor) na napon koji koristi serijska veza na računaru (računar koristi napon od 12V za serijsku komunikaciju).

7. Kolo za dobijanje naponskih nivoa od 1.8V i 3.3V (na koliko radi i procesor) koristeći napon od 5V koliko se dovodi na ulaz kontrolne kartice. Kontrolna dioda LD1 svetli zeleno uvek kada je dovedeno napajanje na kontrolnu karticu (*control card*).

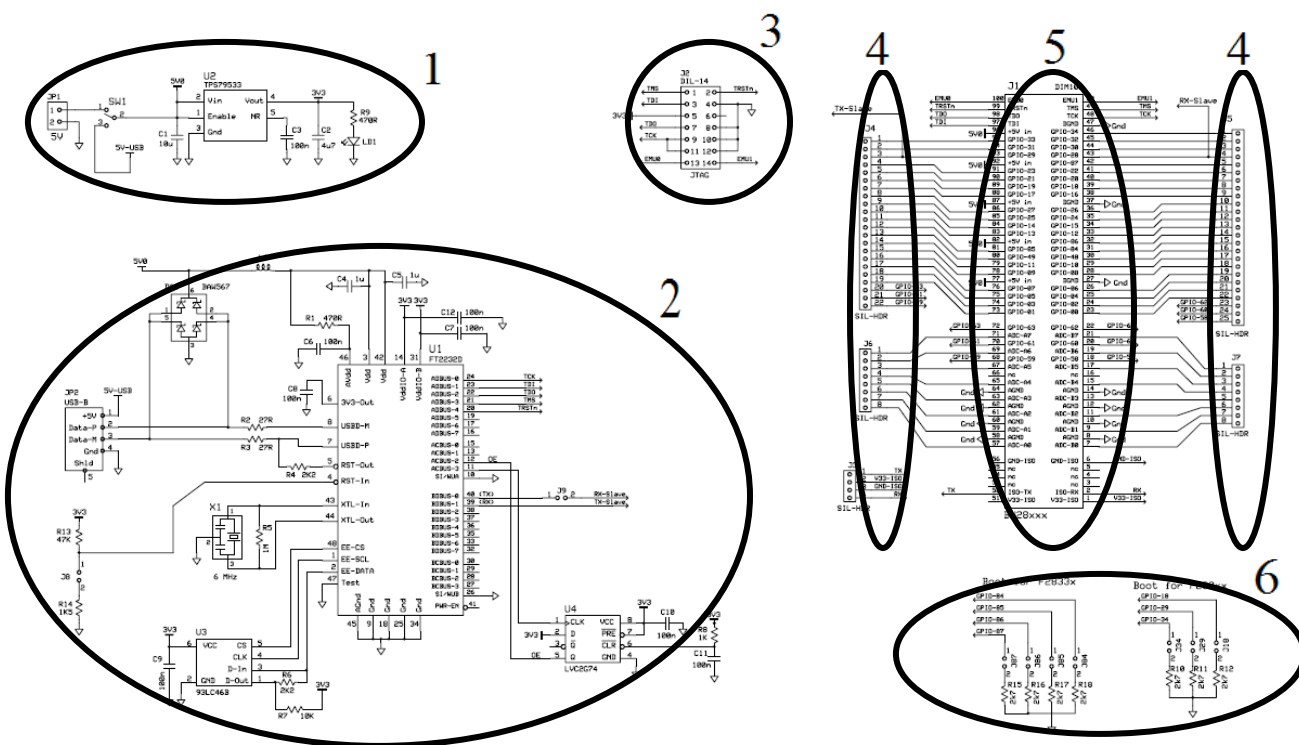
8. Kolo za RESETn pin koji se koristi za restartovanje.

9. ADC pinovi. Ovo su analogni pinovi i koriste se za analogno-digitalnu konverziju. Ulazni opseg im je 0-3.3V.

10. Signali koji sa kontrolne kartice ulaze u DIMM100 konektor. Treba obratiti pažnju da se ne dovode svi GPIO pinovi sa kontrolne kartice na DIMM100, što se može zaključiti i prostim brojanjem, tako da mnogi GPIO pinovi ostaju neiskorišćeni.

11. Kolo za diode LD2 i LD3. Prilikom rada sa DSC-om korisno je pored emulatora imati jos neku mogućnost za prikaz odziva sa kartice. U ovu svrhu se koriste diode LD2 i LD3 čije stanje korisnik softverski kontroliše.

#### 1.4 Šema docking station-a sa objašnjenjima



Na gornjoj šemi se mogu zapaziti:

1. Kolo za napajanje. U ovom kolu se može zapaziti prekidač SW1 (ovaj prekidač ne treba mešati sa switch-em SW1 koji se nalazi na kontrolnoj kartici) koji određuje da li će se DSC napajati preko ispravljača 5V DC, ili preko USB veze. Kada se DSC napaja, dioda LD1 svetli zeleno (ne treba je mešati sa diodom LD1 sa kontrolne kartice).

2. JTAG emulacija. Ovo kolo predstavlja onboard XDS100 JTAG emulator pomoću koga se eliminiše potreba za dodatnim, eksternim emulatorom. Emulator se najčešće koristi da bi se u toku rada pročitao stanje određenih registara procesora, bez prekidanja ili ometanja osnovnog programa koji se izvršava u tom trenutku.

3. Kontakti za eksterni JTAG emulator. Iako DSC F28335 već ima u sebi onboard XDS100 JTAG emulator, postoji mogućnost povezivanja eksternog emulatora koji bi preuzeo funkciju XDS100 (ovo naravno ima smisla ako je eksterni JTAG boljih karakteristika od XDS100).
4. Dostupni pinovi na docking station-u. Ovde su izvedeni pinovi sa procesora i odavde mogu lako da se povezuju sa drugim elementima jer su pinovi fizički dostupni.
5. DIMM100 konektor. Ovde su prikazani pinovi koji ulaze u DIMM100 konektor.

## 2. UPUTSTVO ZA KORIŠĆENJE

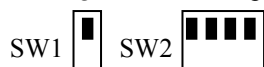
### 2.1 Generalno uputstvo za sve vežbe

U ovom poglavlju dato je uputstvo za pokretanje primera datih u ovom dokumentaciji. U dokumentaciji su data tri primera pod nazivima: vežba1 (GPIO), vežba2 (timer) i vežba3 (interrupt). Postupak pokretanja sva tri primera je identičan i biće objašnjen u ovom poglavlju.

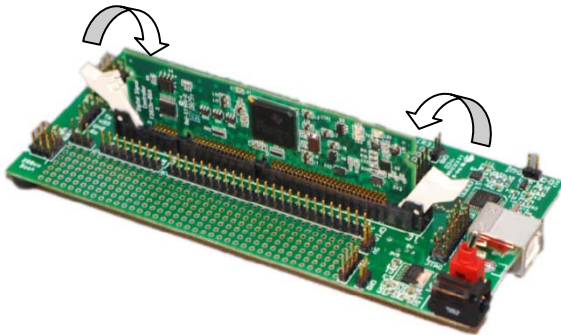
Od hardvera potrebno je imati: PC računar, dsp F28335 i USB kabl. Od softvera potrebno je imati CCSv3.3 instaliran na računaru i primere koji se nalaze na ovom DVD-u na putanji "SOFTVER / Primeri".

Postupak za pokretanje je sledeći:

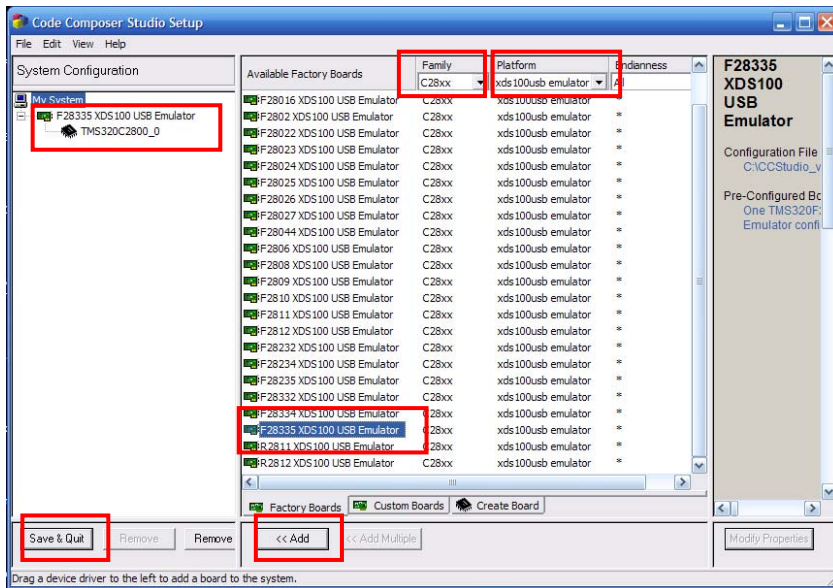
1. Uveriti se da su *switch-evi* na *control card-u* F28335 postavljeni kao na slici (svi na položaju ON) i da na *docking station-u* nisu postavljeni *jumperi*:



2. Ubaciti *control card* u *docking station*. Ovaj korak je prikazan na donjoj slici:



3. Povezati PC računar sa F28335 putem USB kabla
4. Postaviti prekidač na F28335 u položaj ON.
5. Pokrenuti "Setup CCStudio v3.3". Iz menija "Family" izabrati opciju "C28xx", a iz menija "Platform" opciju "xds 100 usb emulator". Sa liste "Available Factory Boards" izabrati "F28335 XDS100 USB emulator" i pritisnuti dugme "<<ADD". U ovom trenutku ekran treba da izgleda kao na donjoj slici:

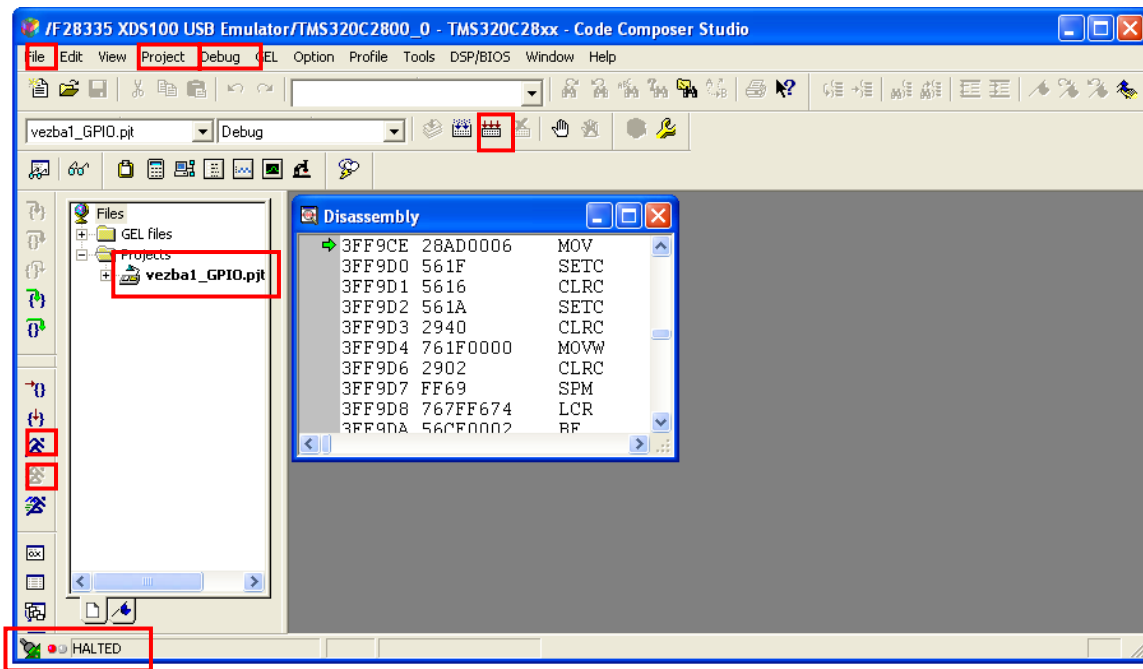



Kliknuti na dugme "Save and Quit". Na upit "Start Code Composer Studio on exit" kliknuti dugme "Yes". Ovaj korak se vrši samo prvi put kada se na računaru pokreće program za DSP F28335. Za svaki sledeći put ovaj korak glasi "Pokrenuti CCSv3.3".

6. Iz menija "Debug" odabrati opciju "Connect".


Ako je sve prošlo kako treba u ovom trenutku korisnik bi trebalo da dobije povratnu informaciju "The target is now connected". Ako ovo nije slučaj treba ugasiti prekidač na DSP-u, ugasiti CCSv3.3 i ponoviti korake 4-6.


7. Iz menija "Project" izabrati opciju "Open", izabrati folder "SOFTVER/Primeri/Vežba1,2 ili 3" i kliknuti na odgovarajući ".pjt" fajl. Sada bi ekran trebalo da izgleda kao na donjoj slici.



8. Kliknuti na dugme "Rebuild All" .

9. Iz menija "File" izabrati opciju "Load program". Izabrati odgovarajući ".out" fajl iz foldera "Debug".

10. Pritisnuti dugme "Run" .

11. Nakon izvršenog zadatka, koji je u svakom primeru različit, potrebno je zaustaviti program pritiskom na dugme "Halt" .

12. Ako korisnik želi da u ovom trenutku pokrene neki drugi primer potrebno je da ponovi korake 7-11. Ako postoji želja da se završi sa radom na F28335 potrebno je iz menija "Debug" izabrati opciju "Disconnect", ugasiti CCSv3.3, premestiti prekidač na DSP-u u položaj "OFF" i izvaditi USB kabl iz PC računara.

Ovim se uputstvo za korišćenje završava. Kao što je rečeno, uputstvo je identično za sva tri primera data u okviru ove dokumentacije. U nastavku će se ova tri primera (vežbe) opisati pojedinačno.

## 2.2 Primeri

Sledi opis tri primera datih u ovoj dokumentaciji. Za svaki od primera će se dati namena, procesi koje korisnik treba da posmatra i postupak (zadatak) koji korisnik treba da uradi.

### 2.2.1 Vežba 1: GPIO

Ova vežba je namenjena savladavanju inicijalizacije pinova DSP-a. U okviru inicijalizacije je demonstrirano kako se podešava a) namena pina, b) da li je pin ulazni ili izlazni, c) da li je početo stanje pina "PullUp ili PullDown". U okviru vežbe su takođe prikazana tri načina kako je moguće promeniti stanje GPIO pinova. Za ovu vežbu je potreban osciloskop koji se koristiti za prikaz stanja pinova.

Ova vežba ima u sebi pet primera. Prva dva vrše inicijalizaciju pinova na različite načine, dok treći, četvrti i peti demonstriraju tri načina menjanja vrednosti pina. Potrebno je u projektu u okviru odeljka "source" otvoriti fajl "vezba1\_GPIO.c". Željeni primer se bira tako što se u ovom fajlu ispred njega stavi vrednosti 1, dok se ispred ostalih primera stavi 0 (linije 76-80). U toku rada je dozvoljeno da samo jedan primer bude aktiviran (jedna jedinica), a nikako više.

U prva dva primera se može videti kako se inicijalizuju pinovi za različite funkcije. Međutim, pošto ne postoji nikakva manifestacija koja se izvršava i koja može da se prati, a kod koji je korišćen se objašnjava tek u trećem poglavlju, ovde će se odmah preći na treći, četvrti i peti primer. Za sva tri ova primera postupak koji korisnik treba da uradi je identičan: treba pomoću osciloskopa da prati kako se stanje na pinovima menja. Ovo se vrši tako što se masa sonde osciloskopa poveže na bilo koji pin na kartici F28335 na kome piše GND, a sama sonda se poveže na pin čije se stanje posmatra. Izbor pina za posmatranje je ostavljen korisniku. Zanimljivo je u isto vreme pratiti dva pina čija su stanja komplementarna (npr. pinovi 31 i 34). Ako je sve dobro povezano trebalo bi da se vidi kako se na ovim pinovima stanje naizmenično menja. Diode LD2 i LD3 će naizgled obe bez prekida svetleti, ali to je samo iz razloga što ljudsko oko ne može da prati veliku frekvenciju treperenja. Ovo se može proveriti tako što se smanji učestanosti promene stanja. To je najlakše uraditi tako što se u funkciji "delay\_loop" jako poveća broj do koga se broji (linija koda br.171). Na ovaj način je moguće videti promene stanja pinova i pomoću osciloskopa i pomoću dioda. Ovim je vežba 1 završena.

Za objašnjenje koda vežbe pogledati poglavlje 3.2.1.

### 2.2.2 Vežba 2: Timer

Ova vežba je namenjena savladavanju rada sa brojačima (*timer-ima*). U okviru vežbe je objašnjeno a) šta su *prescaler-i* b) kako se *timer-i* inicijalizuju, c) kako se dovodi signal dozvole (napajanje) na periferije. Signal sa dioda LD2 i LD3 se koristiti za prikaz stanja brojača.

Po pokretanju vežbe potrebno je posmatrati diode LD2 i LD3. Period sva tri *timer-a* je podešen da traje jednu sekundu. *Timer1* je podešen da kontroliše diode, pa će period paljenja dioda odgovarati *timeru1* i biti 1 sekunda. Dioda će se paliti i gasiti u zavisnosti od toga da li je stanje brojača *timer-a* manje ili veće od polovine perioda do kog treba da broji. Npr. ako se broji do 150 000 000, ako je stanje brojača između 0-75 000 000 dioda će biti upaljena, a ako je između 75 000 000 – 150 000 000 dioda će biti ugašena. Korisnik može da proveriti kako se menja brzina paljenja i gašenja dioda (period *timer-a*) kada se menja period brojanja brojača. To se postiže direktnim upisom u registar "*CpuTimer1Regs.PRD.all*". Kada korisnik isproba promenu brzine paljenja dioda i vizuelno verifikuje da je uspeo ova vežba je završena.

Za objašnjenje koda vežbe pogledati poglavlje 3.2.2.

### 2.2.3 Vežba 3: Interrupt

Ova vežba je namenjena savladavanju rada sa prekidima (*interrupt-ima*). U okviru vežbe je prikazana inicijalizacija 3 *interrupta* koji odgovaraju *timer-ima* 0, 1 i 2. U ovoj vežbi se koristi "*watch window*" i "*real time debugging*" za prikaz načina izvršavanja *interrupta*.

U ovoj vežbi pokrenuta su tri *interrupta*. Njih kontrolišu *timer-i* 0, 1 i 2. Kada brojač *timer-a* dodje do broja 0 (završi period brojanja) on pokreće *interrupt*. Interrupt se ne poziva nigde u kodu već ga *timer* automatski generiše. Da bi se videlo kako se *interrupti* izvršavaju koristiće se "*onboard XDS100 JTAG emulator*" koji već postoji na *docking station-u* i opcija "*RealTimeMode*" koja se koristi za prikazivanje promenljivih u realnom vremenu. Pre nego što se pokrene program (dugme "*Run*") potrebno je registar "*CpuTimer0.InterruptCount*" dodati u "*watch window*". Ovo se postiže tako što se ovaj registar zacrni (linija koda br.193), a zatim posle desnog klika na njega izabere opcija "*Add to Watch Window*". Da bi se u realnom vremenu pratilo stanje ovog registra potrebno je iz menija "*Debug*" izabrati opciju "*Real-Time Debug*", a zatim iz menija "*View*" opciju "*Real Time Refresh Options*" i u okviru nje podesiti period osvežavanja stanja npr. 100ms i prihvatiti opciju "*Global Continuous Refresh*". Sada će emulator svakih 100ms proveravati stanje registara iz *watch window-a* i ako se ono promeni crvenim brojevima prikazati novo stanje na istom mestu. Sada korisnik treba da pokrene program (dugme "*Run*") i posmatra kako se menja registar iz *watch window-a*. Može se posmatrati i više registara u isto vreme, pa je zanimljivo pored "*CpuTimer0.InterruptCount*" dodati u *watch window* "*CpuTimer1.InterruptCount*" i "*CpuTimer2.InterruptCount*". Sva tri registra će se menjati u isto vreme i biće jednaki jer su *timeri* isto podešeni. Sada korisnik može da proba da menja periode brojanja *timer-a* i da gleda u realnom vremenu kako se to odražava na brzinu izvršavanja *interrupta* 0,1 i 2. Kada se ovo izvrši vežba 3 je završena.

Za objašnjenje koda vežbe pogledati poglavlje 3.2.3.

## 3. SOFTVER

### 3.1 Nalaženje softvera na DVD-u

Sav softver koji se koristi u okviru ove dokumentacije nalazi se u folderu "SOFTVER". U ovom folderu postoje folderi: "Primeri" (u njemu se nalazi kod za sve tri vežbe), "DSP2833x\_common" i "DSP2833x\_headers"(to su pomoćni folderi koji moraju tu da stoje jer su neophodni za funkcionisanje glavnog koda) i "IZVORNI KOD.rar"(to je ".rar" arhiva sa TI kodom od koga se krenulo).



## 3.2 Objašnjenje softvera u vežbama

U ovom delu objašnjen je kod koji se koristi u vežbama. Za svaku vežbu objašnjeno je uopšteno kako se postiže cilj vežbe: za vežbu 1 to je podešavanje pinova, za vežbu 2 to je podešavanje timera i za vežbu 3 to je podešavanje interrupta. Sve izmene u odnosu na originalni TI kod su obeležene sa „//xxx“.

### 3.2.1 Vežba 1:GPIO

U ovoj vežbi biće opisane stavke koje je potrebno odraditi da bi se izvršila inicijalizacija pinova.

Pojedinačno, pinovi na F28335 mogu biti izabrani da rade kao digitalni ulazi/izlazi (funkcija pina koja se obično naziva GPIO – *General Purpose Input Output*), ili da rade povezani na jednu od tri (za svaki pin ponuđene) periferije kao njihovi ulazi ili izlazi. Pri opredeljivanju koji pin će imati koju funkciju *word* dokument “28335\_PinTable” može da bude od koristi (on se nalazi u folderu “HARDVER”).

Za inicijalizaciju pina potrebno je uraditi tri stvari. Pored gore opisanog izbora funkcije pina, to su izbor da li je on ulazni ili izlazni i da li je na početku postavljen na logičku nulu ili na logičku jedinicu. Sve tri ove stvari se podešavaju u GPIO kontrolnim registrima (*GpioCtrlRegs - GPIO control registers*).

U okviru kontrolnih registara potrebno je obratiti pažnju na sledeća tri registra od značaja:

1. GPxMUXn – *Multiplexer* registar pomoću koga se određuje funkcija pina, što je objašnjeno gore u tekstu.
2. GPxDIR – *Direction* registar, koji služi za izbor da li je pin ulazni ili izlazni
3. GPxPUD – *Pull Up Dissable* registar, koji služi za izbor početnog stanja pina

Svi pinovi na uređaju su raspoređeni u tri porta: PortA (GPIO0-GPIO31), PortB (GPIO32-GPIO63) i PortC (GPIO64-87). Gornja tri registra su data kao uopštena za sve pinove, dok za konkretni pin, za nalaženje ispravnog registra prvo treba videti kom portu pin pripada, a zatim umesto “x” staviti ime porta (A,B ili C). Isto važi i za “n”, samo što on služi za izbor grupe od 16 pinova koji pripadaju istom portu. Njega treba zameniti sa 1 ako je pin u prvoj polovini porta ili sa 2, ukoliko je pin u drugoj polovini porta.

Treba napomenuti da su nakon *reseta* pinovi već inicijalizovani iako to nije uradio korisnik. Naime, reč je o programu koji se pokreće uvek nakon restartovanja procesora, koji je fabrički programiran, nije podložan brisanju i izvršava se iz boot ROM-a. Taj program inicijalizuje pinove svaki put i to na sledeći način: svi pinovi imaju funkciju GPIO, svima je početno stanje nula i svi su podešeni kao ulazi. Ipak, čak iako postavka nakon *resteta* odgovara korisniku, preporučljivo je da on ponovo inicijalizovati pinove jer će tako lakše znati šta je podešeno prostim uvidom u kod.

Sada je korisno pogledati kod vežbe 1, primere 1 i 2 (*example 1, example 2*). Treba posmatrati funkcije “*Gpio\_setup1*” i “*Gpio\_setup2*” i obratiti pažnju na primere kako je sve moguće inicijalizovati različite pinove i to korišćenjem samo tri gore pomenuta registra. Jasno je da postoji skoro beskonačno različitih mogućnosti za inicijalizaciju. Ipak, kada je korisnik upoznat sa tri gore navedena registra postupak inicijalizacije pinova postaje jako jednostavan.

Ovim je objašnjavanje inicijalizacije pinova završeno i sada će se pažnja usmeriti na načine postavljanja i menjanja stanja na već inicijalizovanim pinovima. Ovo je obrađeno u primerima 3,4 i 5 (*example 3, example 4, example 5*).

Postavljanje i promene stanja registra se vrše iz registara podataka (*GpioDataRegs*). Stanje pinova je moguće promeniti na više načina. U okviru registara podataka, registri od značaja su:

1. GPxDAT – bitovi u ovom registru direktno korespondiraju (odgovaraju) stanjima na pinovima.
2. GPxSET i GPxCLEAR – postavljanje stanja pina na jedinicu odnosno nulu, respektivno.
3. GPxTOGGLE – promena stanja na pinu (ako je bilo 0 biće 1 ,a ako je bilo 1 biće 0).

Sada je korisno pogledati kod vežbe za primere 3,4 i 5 (*example 3, example 4, example 5*). Treba posmatrati funkcije “*Gpio\_setup3*”, “*Gpio\_setup4*” i “*Gpio\_setup5*” i uvideti na koji način se u njima menja stanje na pinovima. Treba zaključiti da svaki od gore navedenih registara (1,2 ili 3) može da obavlja istu funkciju. Ovo oslikava i način rada DSC-a, gde se često isti posao može obaviti na više različitih načina. Ne postoji univerzalno najbolji način, a izbor optimalnog zavisi samo od konkretne situacije. Naravno, pre promene stanja pinovi se u ovim funkcijama prvo inicijalizuju ali to je već gore objašnjeno.

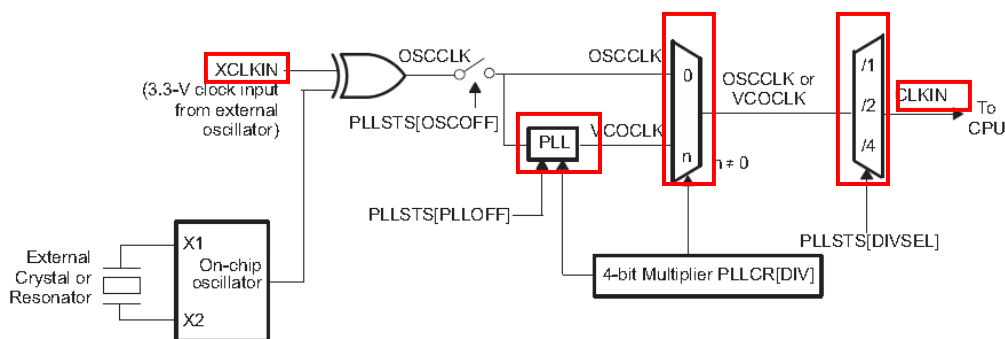
Za dublje zalaženje u temu pogledati dokument na sledećem linku (strane 60-102) :

<http://focus.ti.com/lit/ug/sprufb0d/sprufb0d.pdf> ili identičan dokument naći u folderu “PDF” pod nazivom “SysCtrl\_and\_interrupts.pdf”.

### 3.2.2 Vežba 2: Timer

U ovom delu će biti reči o tome šta je *prescaler*, kako se podešavaju *timeri* i kako se dovodi signal dozvole (napajanje) na periferije. Da bi se ovo objasnilo kreneće se od takta oscilatora.

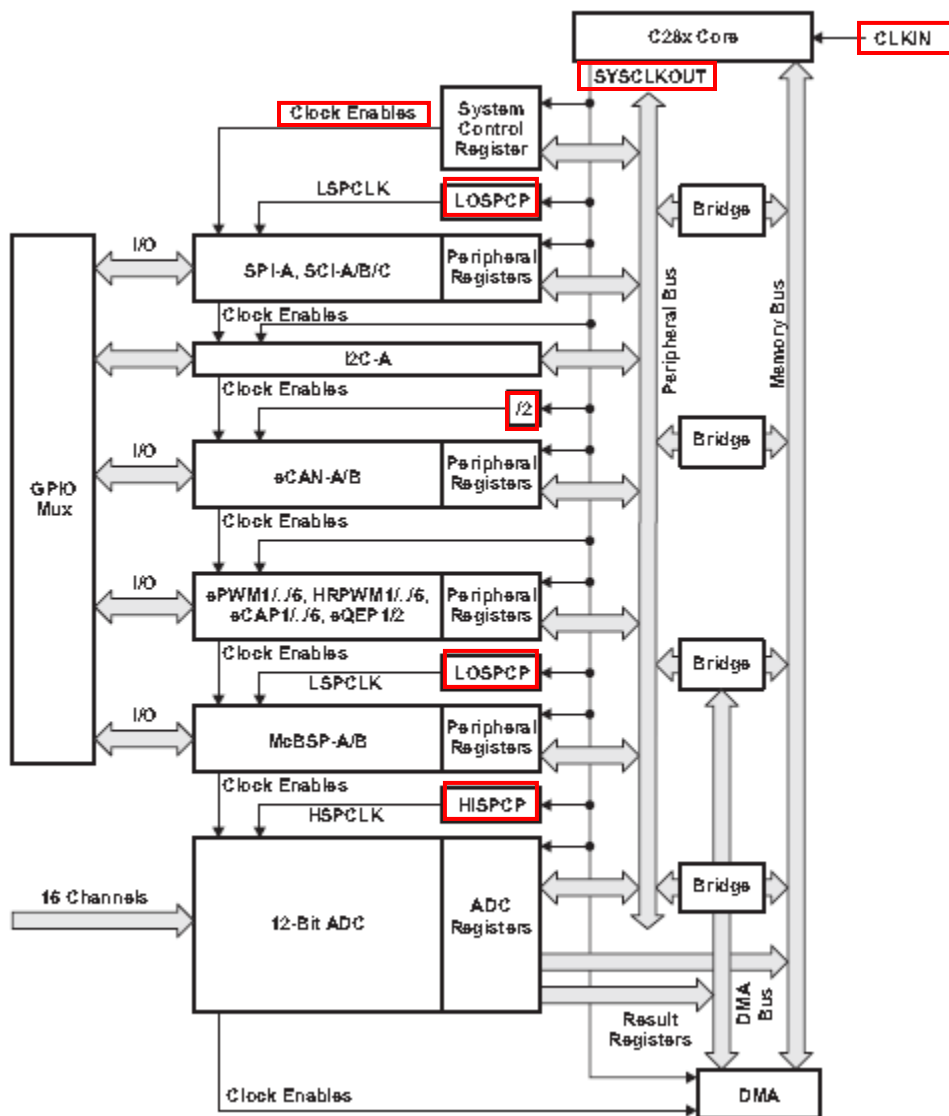
Oscilator i PLL blok (*Phase Locked Loop*), koji se nalaze na čipu, proizvode takt koji se dovodi na uređaje. Oscilator se može posmatrati kao glavni proizvođač takta (koji je na F28335 jednak 30MHz), dok se PLL blok može posmatrati kao množač tog takta. PLL blok ima 4-bitnu kontrolu pojačanja. Kada je takt pomnožen, on se deli sa brojem koji je upisan u statusni registar PLL-a. Najzad, dobija se takt izlaza ka centralnoj procesorskoj jedinici (*CPU - Central Processing Unit*). Ovaj takt se naziva CLKIN. Najveći takt koji može da bude izabran se postiže množenjem sa deset i deljenjem sa 2. Na ovaj način se dobija maksimalni takt od 150MHz ( $30\text{MHz} * 10 / 2 = 150\text{MHz}$ ). Ovaj proces je prikazan na donjoj slici:



CLKIN je takt CPU-a, tj. takt koji ulazi u CPU i na kome CPU radi. CPU na svom izlazu daje takt SYSCLKOUT koji je uvek iste frekvencije kao i CLKIN, samo se drugačije zove. Ovaj takt se zatim vodi ka periferijama.

Da bi periferije radile (bile uključene na napajanje) potrebno im je dovesti dve stvari. To su signal dozvole “*Clok Enable*”(mora se dovesti svakoj periferiji pojedinačno) i signal takta na kome periferija treba da radi. Ovaj signal takta koji se dovodi na periferiju može biti SYSCLKOUT ili SYSCLKOUT podeljen sa nekim faktorom. Ovaj faktor se naziva *prescaler*. Postoje HISPCP (*High-Speed Peripheral Clock Prescaler*) i LOSPCP (*Low-Speed Peripheral Clock Prescaler*). Za svaku periferiju je unapred definisano koji će *prescaler* ona koristiti (HISPCP, LOSPCP ili nijedan) i korisnik nije u mogućnosti to da menja. Ipak, korisnik može da određuje i menja vrednosti

ova dva *prescaler-a* i tako utiče na brzinu rada periferija. Dovođenje signala dozvole i signala takta na periferije je prikazano na donjoj slici:



Sa slike se vidi da LOSPCP *prescaler* koriste periferije: SPI, SCI i McBSP, da HISPCP *prescaler* koristi periferija ADC, da eCAN uvek koristi SYSCLKOUT/2, dok ostale periferije (DMA, I2C, ePWM, HRPWM, eCAP, eQEP) koriste direktno SYSCLKOUT.

Registri od značaja, pomenuti u gornjem izlaganju, su:

1. PLLCR - PLL kontrolni registar, on množi takt sa oscilatora (maksimalno sa 10)
2. PLLSTS - PLL statusni registar, on posle množenja deli takt sa oscilatora (sa 1, 2 ili 4)
3. PCLKCRn - *Peripheral Clock Control Register*, bitovi u njemu predstavljaju signal dozvole za periferije
4. HISPCP - *High-Speed Peripheral Clock Prescaler*

## 5. LOSPCP - Low-Speed Peripheral Clock Prescaler

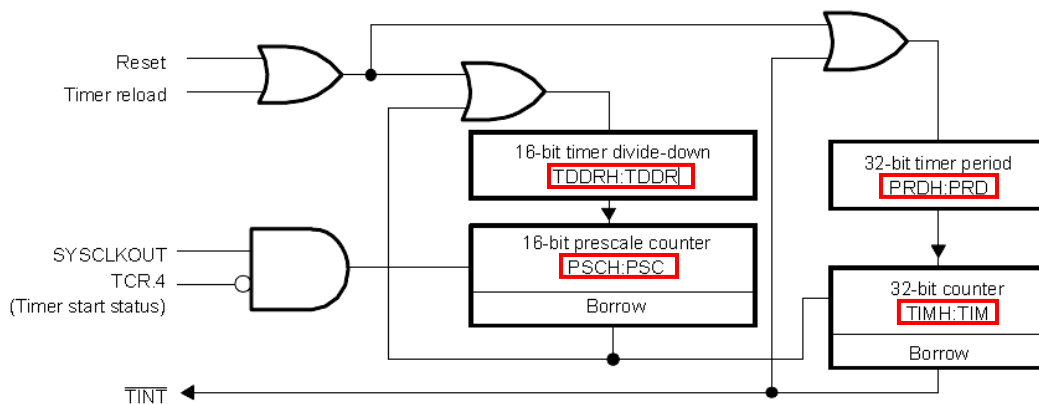
Ovim je završeno objašnjavanje dovođenja signala dozvole i takta na periferije, pa će se pažnja u nastavku usmeriti na *timere*.

Mikrokontroler F28335 ima u sebi tri *timera* (*timer 0/1/2*).

Registri od interesa, koje treba posmatrati su:

1. TIMERnTPR - *prescale* registar
2. TIMERnPRD - *period* registar
3. TIMERnTIM - broječki (*counter*) registar
4. TIMERnTCR - kontrolni registar vezan za interapte, pa će više reći o njemu biti u vežbi br.3

Pomenuti registri se mogu uočiti na donjoj slici (TDDR i PSC zajedno formiraju TIMERnTPR):



Da bi se inicijalizovao *timer* potrebno mu je podesiti 3 stvari:

- 1) do koliko broji - ovo se podešava u registru period: TIMERnPRD
- 2) kojom brzinom broji - ovo se podešava u *prescale* registru: TIMERnTPR
- 3) način na koji broji - ovo se podešava u kontrolnom registru: TIMERnTCR (objašnjen u sl. poglavlju)

Registri TIMERnPRDH i TIMERnPRD zajedno formiraju jedan 32-bitni registar perioda brojanja. Ovaj 32-bitni registar sadrži vrednost od koje će se brojati unazad.

Registri TIMERnTIMH i TIMERnTIM zajedno formiraju jedan 32-bitni brojački (*counter*) registar. Ovaj 32-bitni registar sadrži trenutnu vrednost brojača. Ova vrednost se dekrementira za po jedan na onoj učestanosti na koju je podešen *timer* (SYSCLKOUT/prescaler). Kada vrednost brojača padne na nulu registar se puni sa vrednošću perioda brojanja i dekrementiranje se nastavlja (u ovom trenutku se generiše *flag* za *interrupt*, ali o tome će biti reči u sledećem poglavlju).

Registri TIMERnTPRH i TIMERnTPR zajedno formiraju jedan 32-bitni *prescale* registar. Ovaj 32-bitni registar određuje na kojoj frekvenciji će se brojač dekrementirati. On u stvari određuje sa kojim brojem će se podeliti učestanost SYSCLKOUT (što je učestanost na kojoj radi *timer*). Ovaj 32-bitni registar je pomalo specifičan jer se bitovi (23-16):(7-0) posmatraju zajedno i oni predstavljaju *prescale* za *timer*, dok se registri (31-24):(15-8)

takođe posmatraju zajedno i predstavljaju brojač za *prescale* (*prescale counter*). *Prescale* funkcija se izvršava tako što se vrednost u brojaču za *prescale* dekrementira na svaki otkucaj *clock*-a timera (na frekvenciji SYSCLKOUT). Jedan otkucaj *clock*-a pošto ovaj registar dobije vrednost jednaku nuli on se puni sa vrednošću *prescale*-a za *timer* (bitovi (23-16):(7-0) ), a brojač *timer*-a se dekrementira za jedan. Na ovaj način se brojač timera dekrementira na učestanosti SYSCLKOUT / (*prescale* za *timer*).

Sada je korisno pogledati kod vežbe2 i podesiti *timer1* na drugi način i verifikovati način brojanja vizuelnom metodom pomoću dioda. Takođe, korisno je ubrzati ili usporiti učestanost paljenje dioda podešavanjem perioda brojanja i *prescaler*-a. Zanimljivo je i probati da se zadrži ista (početna) vrednost učestanosti paljenja promenom *prescale* registra ako se promeni period brojanja i obrnuto.

Za dublje zalaženje u temu pogledati dokument na sledećem linku (strane 32-59) :

<http://focus.ti.com/lit/ug/sprufb0d/sprufb0d.pdf> ili identičan dokument naći u folderu “PDF” pod nazivom “SysCtrl\_and\_interrupts.pdf”.

### 3.2.3 Vežba 3: Interrupt

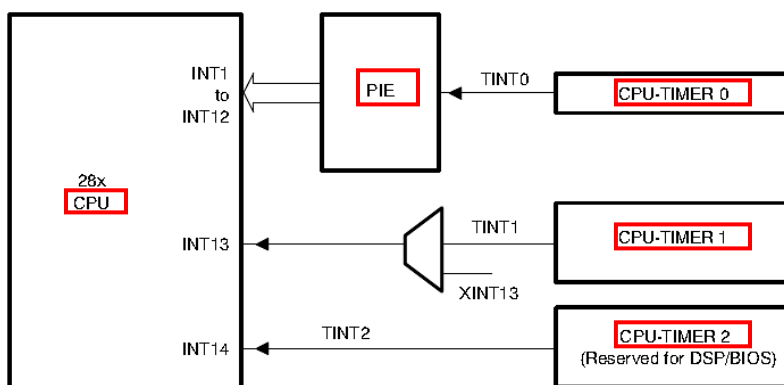
U ovom delu će biti reči o tome kako *timeri* generišu *interrupt-e* i biće objašnjen PIE (*Peripheral Interrupt Expansion*) blok.

Ovo poglavlje se direktno nadovezuje na prethodno, pa čitaocu koji se prvi put susreće sa DSC-om nije preporučljivo dalje čitanje ovog poglavlja ako nije pročitao prethodno.

Kao što je u prethodnom poglavlju napomenuto, kada vrednost brojača *timer*-a opadne na nulu taj *timer* generiše *flag* (početni signal) za *interrupt* ako postoji signal dozvole za *interrupt*. Kontrolni registar *timer*-a “TIMERnTCR”, koji je vezan za *interrupt-e* je namerno preskočen u prethodnom poglavlju. Sada je vreme da se on obradi u okviru objašnjavanja načina generisanja *interrupt-a*. Bit 14 ovog registra ima jako važnu funkciju, a to je da dozvoli ili zabrani generisanje *flag*-a (početnog signala) za *interrupt* u trenutku kada vrednost brojača *timer*-a opadne na nulu. Ako je ovaj bit podešen na jedinicu generisanje *flag*-a će biti dozvoljeno.

*Flag* se može posmatrati kao jedan običan bit i to bit koji se nalazi na mestu br.15 gore pomenutog registra. Kada *timer* generiše *flag* on u stvari samo podesi bit 15 registra TIMERnTCR na jedinicu. Ako brojač još nije dostigao nulu, vrednost ovog bita će biti 0 (flag se neće generisati).

*Flag* za *interrupt* se obično ne dovodi direktno na procesor, već prolazi kroz razne blokove koji ga propuštaju, zadržavaju ili poništavaju. Kako izgleda putanja početnog signala sa različitih *timer*-a do CPU-a može se videti na donjoj slici:

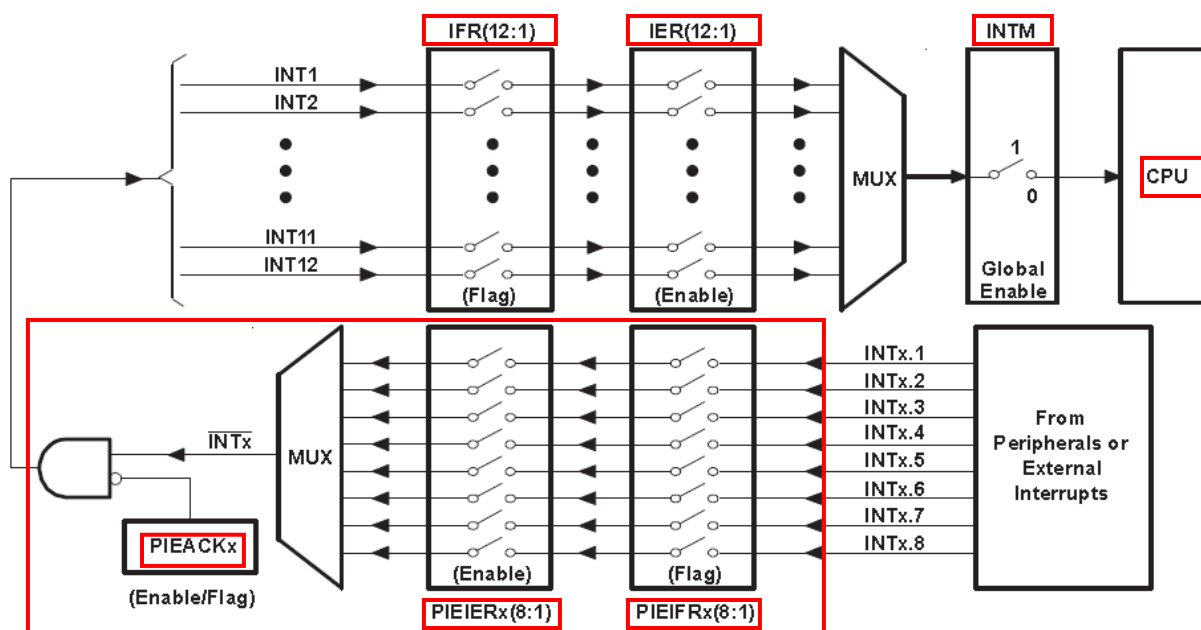


Uvidom u gornju sliku može se zapaziti da *flag* sa timer-a 1 i 2 mnogo lakše nalazi put do CPU-a, dok flag sa timer-a 0 mora da prođe kroz PIE (Peripheral Interrupt Expansion) blok.

U nastavku će biti više reči o PIE bloku.

Zašto se uopšte ukazala potreba za PIE blokom? Mikrokontroler F28335 ima puno periferija i svaka od njih ima mogućnost da generiše jedan ili više *interrupt-a* koji su određeni raznim događajima na nivou periferije. Pošto CPU nema dovoljno kapaciteta da uskladi sve zahteve za *interrupt* koji dolaze sa periferija tu funkciju preuzima PIE blok, koji će prosledivati procesoru samo onoliko *interrupt-a* koliko je procesor u stanju da primi. PIE blok podržava (multipleksira) do 96 individualnih *interrupt-a*. On omogućava procesoru da umesto 96 prima samo 12 zahteva za *interrupt* sa periferija (ili eksternih zahteva).

Na donjoj slici je prikazana šema PIE bloka (PIE blok je uokviren na slici):



PIE vektor tabela (*vector table*) se koristi da sačuva adrese (vektore) svake pojedinačno ISR- *interrupt* rutine (ISR- interrupt service routine) u okviru sistema.

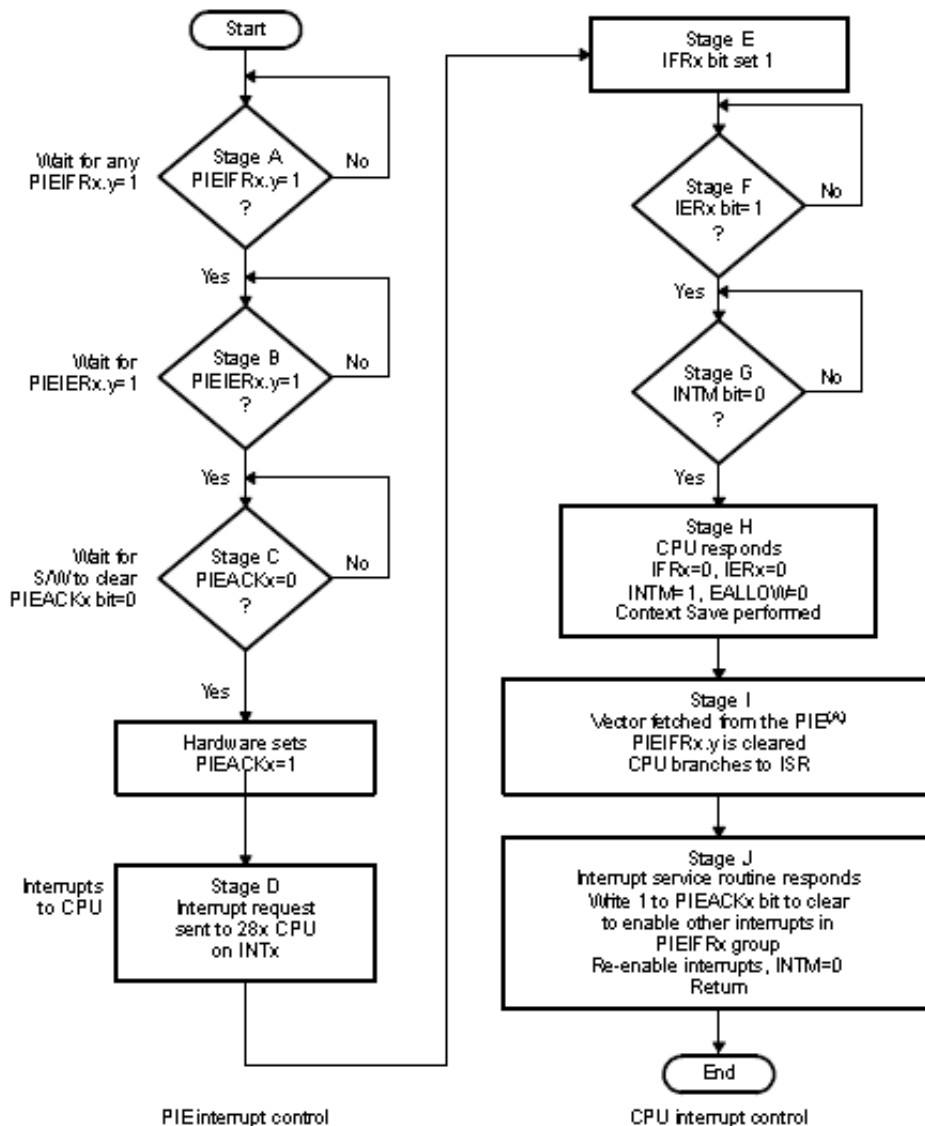
Kao što se na gornjoj slici vidi, PIE blok multipleksira 8 signala za interrupt u 1 signal koji se dovodi do CPU. Ovi signali koji se dovode do CPU su podeljeni u 12 grupa (*PIE group 1- PIE group 12*). Svi signali (*interrupt-i*) u okviru jedne grupe se multipleksiraju u jedan *interrupt* koji se vodi na CPU.

Svaka PIE grupa (*PIE group 1- PIE group 12*) ima svoj *flag* registar (PIEIFRx) i svoj registar za dozvolu (*enable* registar- PIEIERx), gde x može biti od 1 do 12. Svaki signal unutar grupe ima svoj PIEIFRx.y i PIEIERx.y registar. Npr. registar PIEIFR11.6 se odnosi na interrupt br. 6 iz PIE grupe 11.

Kada *interrupt* dođe na ulaz PIE bloka dešava se sledeće: *PIE interrupt flag* bit (PIEIFRx.y) za taj *interrupt* se postavlja na jedinicu. Ako je odgovarajući *enable* bit (PIEIERx.y) takođe postavljen na jedinicu, onda PIE proverava odgovarajući PIEACKx bit da bi proverio da li je CPU spreman za *interrupt* iz grupe iz koje dolazi ovaj *interrupt*. Ako je ovaj bit na nuli za ovu grupu, onda će PIE poslati zahtev za *interrupt* na CPU. Ako je PIEACKx bit na jedinici, onda će PIE čekati da se ovaj bit izbriše, pa tek onda poslati zahtev za *interrupt* na CPU.

Ovim se završava tok signala kroz PIE i signal ulazi u CPU. Na nivou CPU-a dešava se sledeće: kada stigne zahtev za *interrupt* odgovarajući IFR bit, koji odgovara broju *interrupt-a* (IFR1- IFR12) se postavlja na jedinicu. *Interrupt* će se izvršiti ako je odgovarajući IER bit na jedinici (IER1- IER12) i ako je postavljena globalna dozvola za *interrupt-e* (INTM).

Upravo opisani algoritam se može uočiti na donjoj slici:



Leva strana slike se odnosi na PIE blok, dok se desna odnosi na CPU. Pošto se iz algoritma vidi da je nakon završavanja *interrupt-a* iz jedne grupe odgovarajući PIEACKx bit postavljen na jedinicu (onemogućeno primanje sledećih *interrupt-a* iz te grupe) potrebno je unutar same ISR ovaj bit postaviti na nulu.

Sada je korisno pogledati kod vežbe 3 i uočiti kako se za *interrupt* sa *timer-a 1* postavlja PIEACKx bit na nulu da bi se omogućilo primanje sledećih *interrupt-a* iz te grupe. Korisno je verifikovati da li će se *interrupt* samo jednom izvršiti ako se ovaj bit ne izbriše u okviru samog *interrupt-a*. Isto tako treba uočiti da *timeri 1* i *2* nemaju u svojim ISR komandu za brisanje PIEACKx bita zato što *interrupt-i* koje oni generišu ne ulaze u PIE blok (pogledati šemu sa početka poglavlja).

Za dublje zalaženje u temu pogledati dokumente na sledećim linkovima:

<http://focus.ti.com/lit/ug/sprufb0d/sprufb0d.pdf> i <http://focus.ti.com/lit/ug/spru430e/spru430e.pdf> ili identične dokumente naći u folderu "PDF" pod nazivima "SysCtrl\_and\_interrupts.pdf" i "CPU\_and\_InstructionSet.pdf".