Univerzitet u Beogradu

Elektrotehnički fakultet

Laboratorija za mikroprocesorsko upravljanje

elektromotornim pogonima

Implementacija *Space-Vector* modulacije na procesoru *DSP56F801*

- Seminarski rad -

Student

Goran Mandić

Br. Indeksa

155/04

Mentor

prof. dr Slobodan Vukosavić

Beograd, 2007

Sadržaj:

1.Uvod

Ovaj seminarski rad se bavi ispitivanjem mogućnosti primene procesora *DSP56F801* proizvođača *Freescale/Motorola* u upravljanju energetskim pretvaračima. Iako naslov glasi "Implementacija Space-Vector modulacije na procesoru *DSP56F801*", najveći deo teksta je posvećen opisu samog procesora, njegovih periferijskih jedinica kao i praktičnim detaljima vezanim za programiranje ovog procesora. To se pre svega odnosi na opis tzv. *hardware abstraction layer*-a, koji je razvijen sa ciljem da značajno olakša rad sa periferijskim jedinicama ovog procesora i time skrati vreme potrebno za razvoj softvera za neku konkretnu aplikaciju. Takođe, pored Space-Vector modulacije, opisano je i nekoliko veoma jednostavnih primera koji ilustruju programiranje ovog procesora i praktičnu upotrebu *hardware abstraction layer*-a.

U poglavlju 2. dat je kratak opis osnovnih karakteristika procesora *DSP56F801*.

U poglavlju 3 su ukratko opisane periferijske jedinice ovog procesora koje su korišćene u primerima iz poglavlja 7.

Poglavlje 4 se bavi opisom razvojnog okruženja *Code Warrior,* odnosno osnovnim koracima pri započinjanju novog projekta kao i opis *start-up* koda koji razvojno okruženje generiše pri kreiranju novog projekta.

Poglavlje 5 se bavi problemom pristupa registrima periferijskih jedinica iz C koda. Ovde je opisan jedan koncept to jest *framework* zasnovan na upotrebi struktura i unija kojima se predstavljaju periferijski registri i njihovi pojedinačni bitovi. Upotrebom struktura i unija i njihovim lociranjem na odgovarajuće adrese u memorijskom prostoru dobijen je tzv. sloj za apstrakciju hardvera – *hardware abstraction layer.*

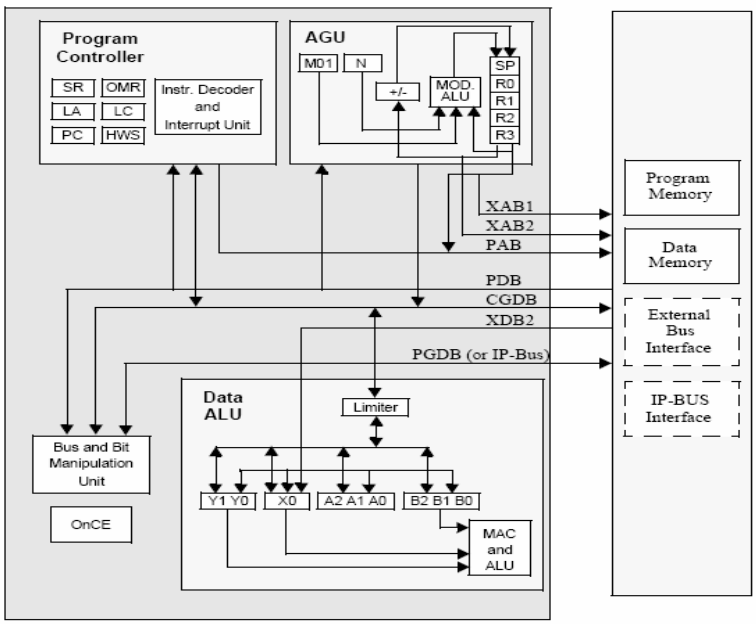
U poglavlju 6. se nalazi kratak opis demonstracionog modula CSM-56F801, koji se može koristiti za testiranje primera opisanih u poglavlju 7.

Poglavlje 7 sadrži četiri primera koji ilustruju rad sa procesorom *DSP56F801*. U prvom primeru je opisan program koji pali i gasi LED diodu na jednom pinu procesora. U drugom primeru je ilustrovana upotreba A/D konvertora dok se treći primer bavi upotrebom serijske veze između procesora *DSP56F801* i nekog drugog uređaja. Na kraju, u četvrtom primeru je opisan Space-Vector algoritam koji se koristi za upravljanje trofaznim tranzistorkim invertorima. Prikazan je jedan način implementacije ovog algoritma, koji ne zahteva intenzivne matematičke operacije.

Na kraju, u prilogu A, data je šema JTAG adaptera koji omogućava programiranje i debagovanje softvera na procesorima familije *DSP56F80x*. U prilogu B je prikazano povezivanje CSM-56F801 modula sa potenciometrom, LED diodom i RC filtrom za testiranje primera iz poglavlja 7.

2. Arhitektura digitalnih signalnih kontrolera familije DSP56F80x

Procesori familije *DSP56F80x,* proizvođača *Freescale/Motorola,* pripadaju grupi tzv. digitalnih signalnih kontrolera. Procesorsko jezgro ove familije, čija je oznaka *DSP56F800*, je 16-bitno i poseduje karakteristike digitalnih signalnih procesora u pogledu izvršavanja aritmetičko-logičkih operacija, brzine rada kao i paralelizma u radu različitih blokova procesorskog jezgra. Sa druge strane, instrukcijski set je sličan instrukcijskom setu mikrokontrolera opšte namene. Druga karakteristika ovih procesora je i harvardska arhitektura, što znači da su memorijski prostori za program i podatke odvojeni. Pored procesorskog jezgra i memorije ovi procesori poseduju i različite periferijske jedinice namenjene za izvršavanje komunikacionih ili upravljačkih zadataka u sistemu sa procesorom *DSP56F80x* . To su I/O portovi opšte namene, tajmersko-brojački moduli, PWM moduli, SPI moduli, serijski interfejsi, interfejsi za eksternu memoriju, itd. *DSP56F80x* procesori poseduju i tzv. *On-Chip Emulation* modul koji zajedno sa integrisanim JTAG portom omogućava kontrolu izvršavanja programa u procesoru od strane razvojnog okruženja nakon ugradnje procesora u sistem, u fazi pronalaženja i otkrivanja grešaka u softveru *(debugging)*. Preko JTAG porta se takođe može programirati interna FLASH memorija procesora. Na slici 2.1 je data blok šema procesorskog jezgra *DSP56F800* koje je zajedničko za sve procesore ove familije.



Slika 2.1. DSP56F800 procesorsko jezgro.

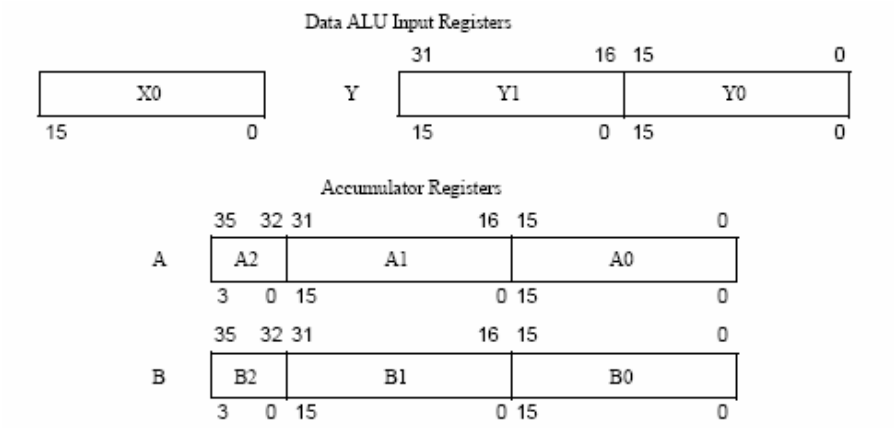
Kao što se sa slike vidi, postoje tri osnovne celine koje rade paralelno. To su :

Aritmetičko logička jedinica, *(Data ALU)*

Adresna jedinica, *(AGU)*

Upravljčka jedinica, *(Program Controller)*

Aritmetičko logička jedinica obavlja operacije sabiranja, oduzimanja, množenja, množenja sa akumulacijom, pomeranje bitova, zaokruživanje itd. ALU posedije dva 36-bitna akumulatorska registra A i B. Ovim registrima se može pristupati u celini ili njihovim delovima. Tako je registar A podeljen na dva 16-bitna registra A0 i A1 i jedan 4-bitni *extension* registar A2. Na isti način je podeljen i B registar. Pored akumulatora A i B ALU sadrži i jedan 32-bitni registar Y kao i jedan 16-bitni registar X0. Registar Y se sastoji od dva 16-bitna registra Y0 i Y1. Za DSP algoritme se koriste celi akumulatorski registri, dok se za izvršavanje zadataka opšte namene koriste 16-bitni delovi ovih registara. Ovim se procesori familije svrstavaju i u DSP procesore kao i u mikrokonrolere opšte namene. Ove vrste procesora se nazivaju i hibridni kontroleri. Na slici 2.2 su prikazani registri aritmetičko logičke jedinice koji su dostupni programeru.



2.2. Registri aritmetičko logičke jedinice

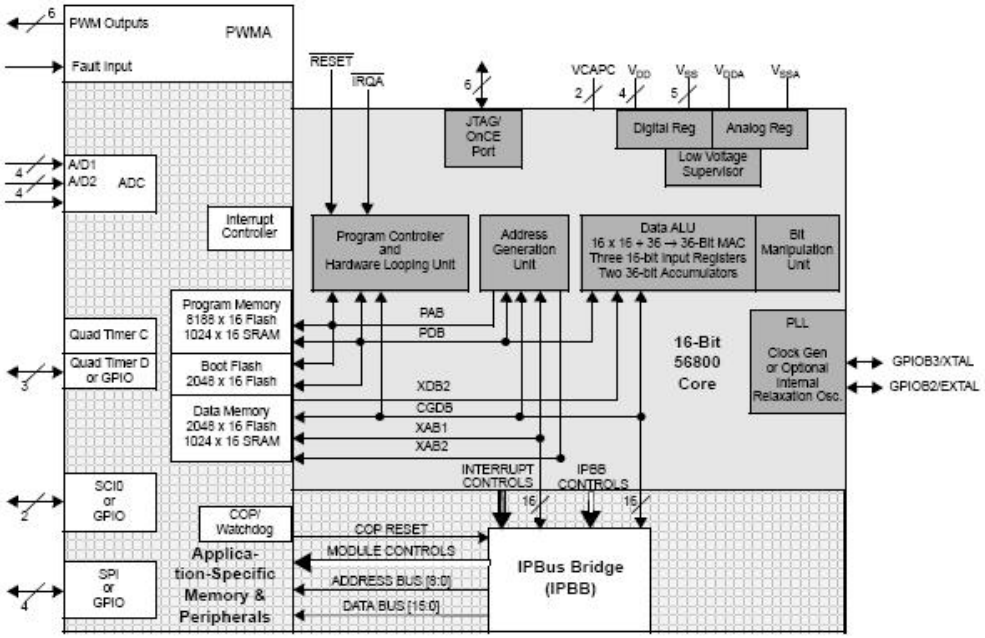
Adresna jedinica obavlja izračunavanje adresa paralelno sa radom ostalih modula u procesoru. Zahvaljujući adresnoj jedinici moguć je veliki broj načina adresiranja kod procesora *DSP56F80x.* Veliki broj načina adresiranja je još jedna od karakteristika procesora opšte namene koja nije postojala kod DSP procesora.

Upravljačka jedinica je zadužena za upravljanje radom procesora. Tu spada čitanje instrukcija iz programske memorije, njihovo dekodovanje i izvršavanje, obrada prekida, itd.

Procesori *DSP56F80x* poseduju više magistrala preko kojih se komunicira sa programskom ili *data* memorijom i periferijskim jedinicama. Ovaj sistem višestrukih magistrala omogućava paralelno pristupanje programskoj i *data* memoriji ili periferijskim jedinicama. Registri periferijskih jedinica su memorijski mapirani u prostor *data* RAM memorije pa se njima pristupa

kao i ostalim lokacijama *data* RAM-a. Hardverski, komunikacija sa periferijskim jedinicama, odnosno njihovim I/O registrima, se obavalja preko *IP-Bus* interfejsa.

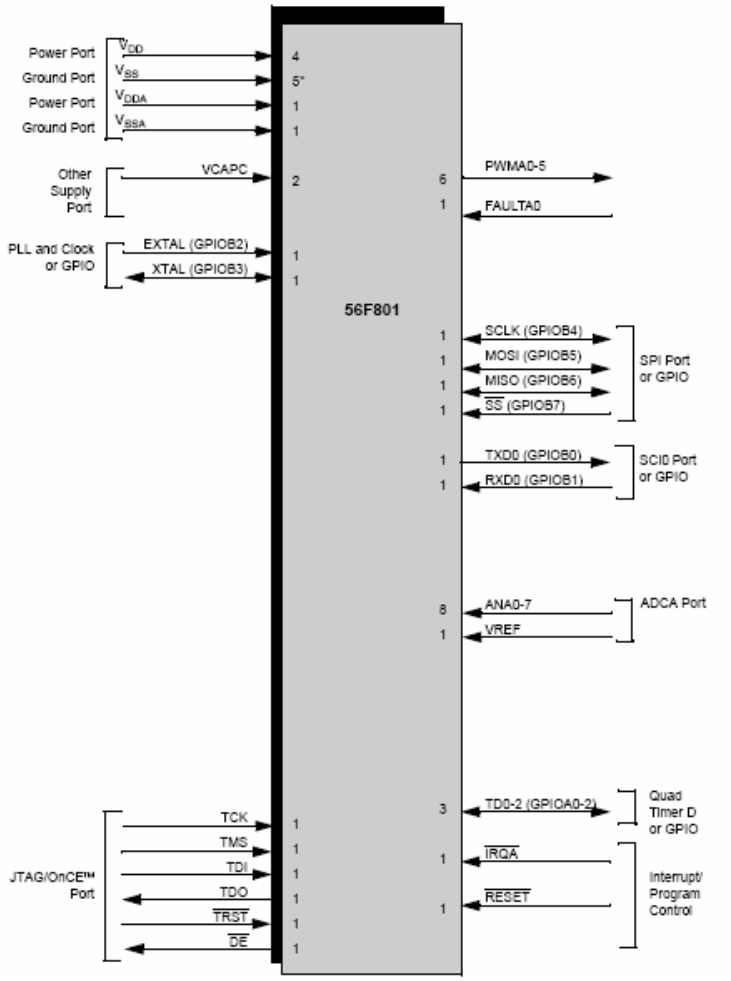
U zavisnosti od količine memorije i izbora periferijskih jedinica, proizvodi se nekoliko varijanti *DSP56F80x* procesora. To su procesori DSP56F801, DSP56F802, DSP56F803, DSP56F805 i DSP56F807. Na slici 2.3 je data blok šema procesora *DSP56F801.*



Slika 2.3. Blok šema procesora DSP56F801

Ostali procesori iz familije *DSP56F80x* se razlikuju od procesora *DSP56F801* samo po količini memorije i broju integrisanih periferijskih jedinica. Procesor *DSP56F801* poseduje nešto manje od 16KB programske FLASH memorije. Postoji i 2KB programske RAM memorije kao I 4KB Boot-FLASH memorije u koju može da se smesti *boot loader.* Za podatke je namenjeno 2KB RAM memorije kao i 4KB FLASH memorije za čuvanje konstanti ili tabela.Od periferijskih jedinica tu su PWM modul sa 6 PWM izlaza, 12-bitni AD konvertor (ADCA), dva tajmersko brojačka modula C i D, serijski interfejs i SPI modul. Za generisanje takta procesor poseduje PLL modul koji može programski da podešava učestanost takta u sistemu. Procesori *DSP56F80x* se u zavisnosti od učestanosti takta izrađuju u verziji od 60MHz ili u verziji od 80MHz. Ova učestanost se dobija iz PLL-a na osnovu takta internog relaksacionog oscilatora ili takt generatora koji koristi spoljašnji kvarcni oscilator koji se povezuje na pinove XTAL i EXTAL. Pri taktu od 80MHz procesorsko jezgro ostvaruje brzinu od 40MIPS, dok je pri taktu od 60MHz brzina jezgra 30MIPS. Periferijske jedinice koriste signal takta koji se dobija deljenjem takta procesorskog jezgra sa dva. Na ovoj učestanosti radi *IPBus* interfejs preko koga se obavlja komunikacija između procesorskog jezgra i I/O registara periferijskih jedinica. Zbog toga se u daljem tekstu učestanost rada *IP-Bus* interfejsa naziva učestanost *IP-Bus* takta. Procesor *DSP56F801* se pakuje u LQFP

kućište sa 48 pinova. Radi na 3.3V pri čemu su I/O pinovi tolerantni na 5V. Na slici 2.4. su prikazani pinovi procesora DSP56F801.



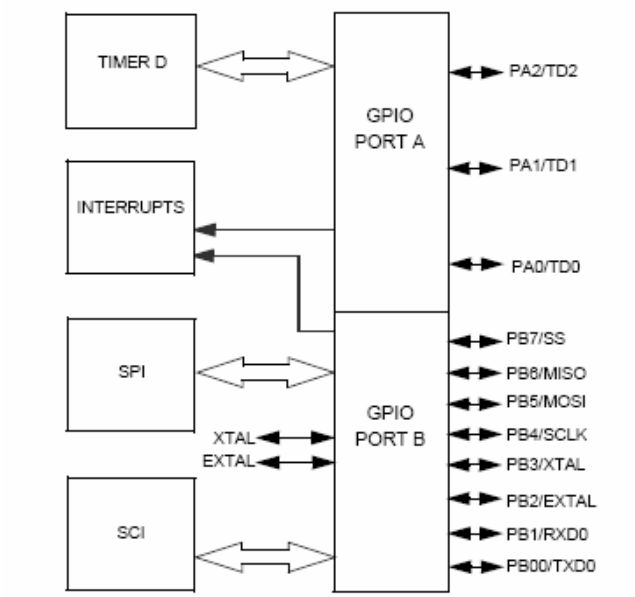
Slika 2.4.Pinovi procesora DSP56F801

U narednim poglavljima su ukratko opisane periferijske jedinice procesora *DSP56F801.*

3. Periferijske jedinice procesora DSP56F801

3.1. Ulazno/Izlazni portovi opšte namane (GPIO)

Procesori familije *DSP56F80x* poseduju grupu I/O pinova koji osim što su dodeljeni pojedinim periferijama, mogu da se koriste i kao I/O pinovi opšte namene. Ovi pinovi mogu da se koriste u slučaju da se odgovarajuća periferijska jedinica ne koristi a namena im može biti digitalni ulaz, digitalni izlaz ili eksterni zahtev za prekid. Procesor *DSP56F801* poseduje 11 ovakvih pinova koji su grupisani u dva porta GPIOA (port A) i GPIOB (port B). Na slici 3.1 je prikazana blok šema veza GPIO portova i periferijskih jedinica koje koriste ove portove.



Slika 3.1. Veze između GPIO portova i odgovarajućih periferijskih jednica.

Kao što se vidi, port A poseduje samo tri pina. PA0, PA1 I PA2. Ovi pinovi su dodeljeni jednoj od tajmersko-brojačkih jedinica opšte namene i to tajmerima D0 (TD0), D1 (TD1) i D2(TD2). O ovim tajmerima će biti više reči u narednom poglavlju.

Port B poseduje osam pinova od kojih su dva dodeljena interfejsu serijske veze (SCI), četiri su dodeljena SPI interfejsu dok su dva pina priključci za kristalni oscilator. Ova dva pina mogu da se koriste ukoliko procesor koristi interni relaksacioni oscilator za generisanje takta.

Da li će se neki pin GPIO porta koristiti kao I/O pin opšte namene ili će biti dodeljen periferijskoj jedinici zavisi od stanja odgovarajućeg bita PER registra *(Peripheral Enable)*. Naime, svaki GPIO port poseduje grupu od po osam registara kojima se kontroliše rad portova. Svaki

GPIO port (u daljem tekstu modul) poseduje sopstveni PER registar. PER registar je šesnaestobitni, ali se koriste samo donjih osam bitova. Ako je neki bit setovan, tada je odgovarajući pin dodeljen periferijskoj jedinici. Na primer ako je setovan bit 0 PER registra GPIOA modula tada je pin PA0 dodeljen tajmeru D0 i ne može da se koristi kao I/O pin opšte namene. Ako je neki bit PER registra resetovan tada se odgovarajući pin može koristiti kao I/O pin opšte namene. U tom slučaju su za kontrolu pina od značaja registri PUR *(Pull-up Enable),* DDR *(Data Direction register)* i DR *(Data register).* Bitovi registra PUR određuju da li su uključeni ili isključeni *pull-up* otpornici na odgovarajućim pinovima. Ovaj registar ima efekta samo na one pinove koji nisu dodeljeni svojim periferijskim jedinicama, dakle samo na one pinove koji se koriste kao I/O pinovi opšte namene. Ako je neki pin konfigurisan kao ulazni, setovanje odgovarajućeg bita PUR registra uključuje *pull-up* otpornik na tom pinu. Ako je bit PUR registra resetovan *pull-up* otpornik je isključen. Na pinove koji su konfigurisani kao izlazni, registar PUR nema uticaj. Da li će neki pin biti ulazni ili izlazni zavisi od odgovarajućeg bita DDR registra. Setovan bit DDR registra znači da je odgovarajući pin izlazni dok resetovan DDR bit znači da je pin ulazni. GPIO pinovima se upravlje preko registra DR. Ako je neki pin konfigurisan kao izlazni, upisom logičke jedinice na njegovu poziciju u DR registru dobija se logička jedinica na tom pinu. Isto tako, stanje na ulaznom pinu se može očitati proverom stanja odgovarajućeg bita DR registra.

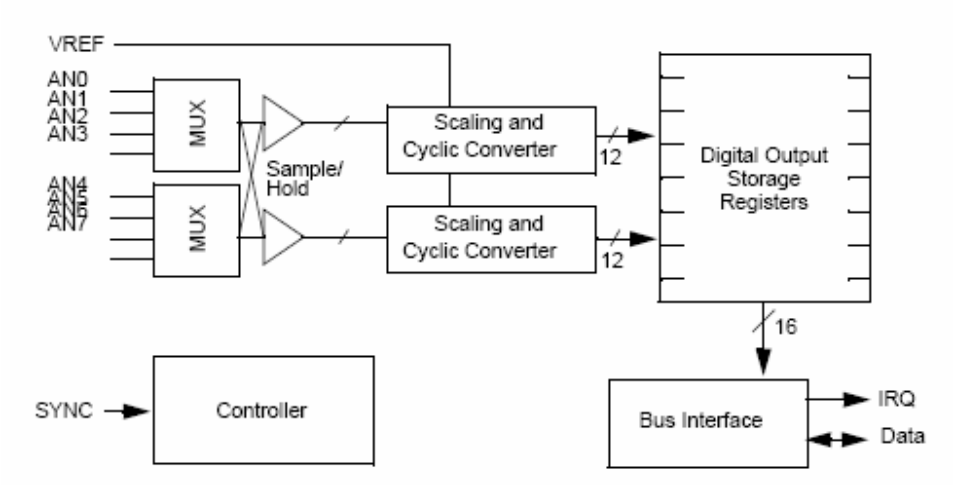
GPIO pinovi mogu takođe da se koriste kao ulazi za eksterne zahteve za prekid. Osim eksternim signalima, ovi zahtevi za prekid mogu da se generišu i softverski setovanjem odgovarajućih bitova IAR *(Interrupt Assert)* registra. Da bi zahtevi za prekid bili prihvaćeni potrebno je da budu omogućeni setovanjem odgovarajućih bitova IENR registra. Da li se zahtev za prekid javlja dovođenjem logičke jedinice, nule ili detekcijom ivice zavisi od stanja registara IPOLR i IESR. Registar IPR sadrži flegove koji signaliziraju da li je do zahteva za prekid došlo.
U tablei 3.1 su prikazani periferijski registari GPIO modula.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	PUR	R	0	0	0	0	0	0	0	0	PUR[7:0]							
		W																
$1	DR	R	0	0	0	0	0	0	0	0	DR[7:0]							
		W																
$2	DDR	R	0	0	0	0	0	0	0	0	DDR[7:0]							
		W																
$3	PER	R	0	0	0	0	0	0	0	0	PER[7:0]							
		W																
$4	IAR	R	0	0	0	0	0	0	0	0	IAR[7:0]							
		W																
$5	IENR	R	0	0	0	0	0	0	0	0	IENR[7:0]							
		W																
$6	IPOLR	R	0	0	0	0	0	0	0	0	IPOLR[7:0]							
		W																
$7	IPR	R	0	0	0	0	0	0	0	0	IPR[7:0]							
		W																
$8	IESR	R	0	0	0	0	0	0	0	0	IESR[7:0]							
		W																

Tabela 3.1. Periferijski registri GPIO modula

3.2. Analogno-digitalni konvertor

Procesori *DSP56F801, DSP56F803* i *DSP56F805* poseduju dvokanalni 12-bitni analogno digitalni (u daljem tekstu A/D) konvertor, sa po četiri analogna ulaza po kanalu. Ukupan broj analognih ulaza na ovim procesorima je osam. Procesori *DSP56F807* poseduju po dva ovakva A/D konvertora. Svaki kanal A/D konvertora poseduje sopstveno *sample and hold* kolo, što omogućava istovremeno uzimanje po dva odbirka sa analognih ulaza. Na slici 3.2. je prikazana blok šema A/D konverora.



Slika 3.2. Blok šema A/D konvertora

Startovanje A/D konverzije pokreće u stvari sekvencu konverzija *(scan).* Ova sekvenca može da se izvrši tako što se simultano obavi A/D konverzija signala sa svih osam ulaza ili tako što se parovi analognih ulaza posmatraju kao jedan diferencijalni analogni ulaz. U tom slučaju se izvršavaju četiri konverzije. Naravno, moguće su i razne kombinacije ova dva načina. Takođe, moguće je zadati sa kojih ulaza se uzimaju odbirci tokom sekvence pa se tako može zadati način rada u kojem se sekvenca sastoji od samo jedne A/D konverzije signala sa jednog ulaza. Nova sekvenca se startuje setovanjem bita START registra ADCR1 ili putem SYNC ulaza na koji se može delovati izlaznim signalom nekog od tajmera iz *quad-timer* modula. Zahvaljujući postojanju SYNC ulaza moguće je veoma jednostavno izvesti sinhronizovanje između A/D konverzije i PWM signala, što je veoma korisno kod upravljanja električnim motorima.

A/D konvertor može da radi u jednom od tri moda, u zavisnosti od stanja SMODE bitova registra ADCR1. Nakon prvog startovanja sekvence A/D konverzija (scan), A/D konvertor može da, u zavisnosti od moda rada, automatski započne novu sekvencu ili da čeka na novi start sekvence. Redosled po kome se uzimaju odbirci može da se programira putem ADLST1 i ADLST2 registara.

ADC modul može da generiše dva različita zahteva za prekid, tako da svaki ADC modul poseduje po dva prekidna vektora. Prvi tip prekida je *End of Scan Interrupt* koji se javlja nakon što se završi *scan* sekvenca. Drugi tip zahteva za prekid se javlja ukoliko je neki od rezultata konverzije van zadatih limita određenih sadržajem ADRLLMTx ili ADHLMTx registara. Takođe, ovaj zahtev za prekid se javlja i ako se detektuje da je semplovani signal promenio znak ukoliko se koriste diferencijalni analogni ulazi.

U tabeli 3.2. su prikazani kontrolni i statusni registri ADC modula.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	ADCR1	R	0	STOP	0	SYNC	EOSIE	ZCIE	LLMTIE	HLMTIE	CHNCRG				0	SMODE		
		W			START													
$1	ADCR2	R	0	0	0	0	0	0	0	0	0	0	0	0	DIV			
		W																
$2	ADZCC	R	ZCE7		ZCE6		ZCE5		ZCE4		ZCE3		ZCE2		ZCE1		ZCE0	
		W																
$3	ADLST1	R	0	SAMPLE3			0	SAMPLE2			0	SAMPLE1			0	SAMPLE0		
		W																
$4	ADLST2	R	0	SAMPLE7			0	SAMPLE6			0	SAMPLE5			0	SAMPLE4		
		W																
$5	ADSDIS	R	TEST		0	0	0	0	0	0	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
		W																
$6	ADSTAT	R	CIP	0	0	0	EOSI	ZCI	LLMTI	HLMT	RDY7	RDY6	RDY5	RDY4	RDY3	RDY2	RDY1	RDY0
		W																
$7	ADLSTAT	R	HLS7	HLS6	HLS5	HLS4	HLS3	HLS2	HLS1	HLS0	LLS7	LLS6	LLS5	LLS4	LLS3	LLS2	LLS1	LLS0
		W																
$8	ADZCSTAT	R	0	0	0	0	0	0	0	0	ZCS7	ZCS6	ZCS5	ZCS4	ZCS3	ZCS2	ZCS1	ZCS0
		W																

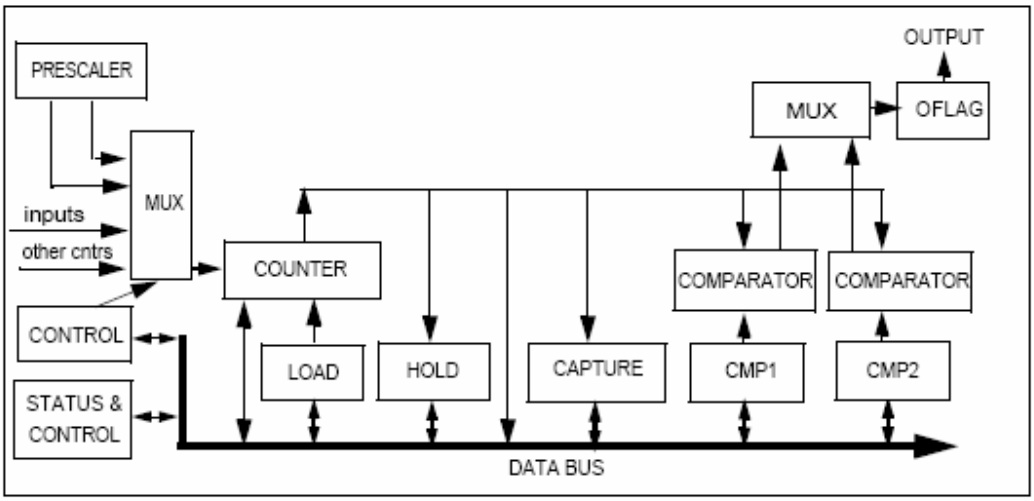
Tabela 3.2. Periferijski registri ADC modula.

Pored ovih registara ADC modul poseduje osam ADRSLTx registara za smeštanje rezultata konverzije, po osam ADRLLMTx i ADHLMTx registara u koje se upisuje gornji i donji limit za semplovane signale i osam offset registara ADOFSx u koje se upisuje offset vrednost koja će se oduzeti od rezultata A/D konverzije pre smeštanja u ADRSLT registar.

Ovde je dat kratak opis ADC modula. Detaljan opis rada ovog modula i njegovih registara se nalazi u uputstvu DSP56F801-FM. U primerima 7.2 i 7.4 je ilustrovana upotreba ADC modula.

3.3. Quad timer moduli

Quad timer modul je tajmersko brojačka periferijska jedinica procesora *DSP56F80x* koja može da se koristi za brojanje internih ili eksternih događaja, merenje trajanja nekog događaja, generisanje signala određenog trajanja na pinovima procesora, generisnje periodičnih prekida, itd. Svaki *quad timer* modul se sastoji od četiri identične tajmersko brojačke jedinice koje će se u daljem tekstu nazivati kanali. Blok šema jednog tajmersko brojačkog kanala je prikazana na slici 3.3.



Slika 3.3. Blok šema tajmersko brojačke jedinice.

Svaki kanal poseduje sopstveni 16-bitni brojač, na slici označen kao *COUNTER*, preskaler, *LOAD* registar, *HOLD* registar, *CAPTURE* registar, dva *COMPARE* registra (CMP1 i CMP2), kontrolni registar i kontrolno-statusni registar. Pošto je brojač šesnaestobitni, gornji limit brojanja je 65535. Izlaz tajmersko-brojačkog kanala je signal OFLAG. Način delovanja na OFLAG signal zavisi od režima rada tajmersko-brojačke jedinice. Registar *LOAD* sadrži inicijalnu vrednost koja se upisuje u brojački registar u trenutku kad brojač završi prethodni brojački ciklus. Registar *HOLD* preuzima stanje brojača u trenutku čitanja sadržaja brojačkog registra nekog drugog kanala u okviru istog *quad timer* modula. Ovaj registar se koristi kada je potrebno očitati sadržaj kaskadno spregnutih brojača. *CAPTURE* registar preuzima stanje brojača u trenutku delovanja eksternog signala. Registri *COMPARE* sadrže vrednosti sa kojima se poredi sadržaj brojača. U trenutku izjednačavanja sadržaja brojačkog registra sa sadržajem nekog od ova dva registra signal OFLAG se setuje, resetuje ili invertuje u zavisnosti od stanja *Output Mode (OM)* bitova kontrolnog registra CTRL.

Procesor *DSP56F801* sadrži dva *quad timer* modula koji se označavaju kao quad timer moduli C i D. Kanalima 0, 1 i 2 modula D, tj. njihovim OFLAG signalima, su dodeljeni pinovi TD0, TD1 i TD2. Ovi pinovi su multipleksirani sa I/O pinovima opšte namene PA0, PA1 i PA2. Svakom kanalu u oba modula je dodeljen vektor prekidne rutine a svaki kanal može da generiše tri različita zahteva za prekid.

Upotreba *COMPARE* registara

Svaki kanal sadrži dva registra za poređenje, CMP1 i CMP2. Ovim registrima se mogu definisati limiti brojanja sa gornje i donje strane, pri čemu se CMP1 koristi za limitiranje sa gornje a CMP2 sa donje strane. Prilikom promene sadržaja ovih registara tokom rada brojača, treba voditi računa da je nova vrednost za upis u CMP1 veća, a vrednost za upis u CMP2 manja od trenutnog stanja brojača. U suprotnom će brojač nastaviti da broji do 65535 ukoliko je smer brojanja bio na gore, odnosno do 0 ako je smer brojanja bio na dole.

Upotreba *CAPTURE* registra

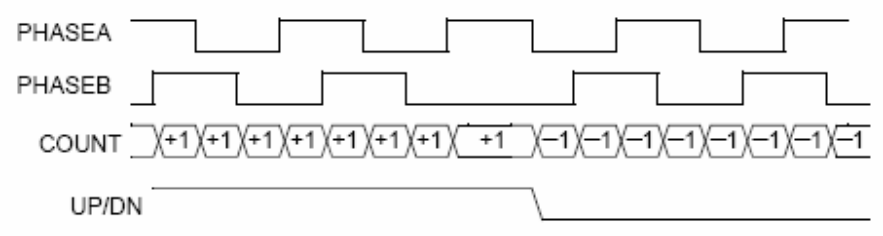
U ovaj registar se upisuje stanje brojača u trenutku tranzicije ulaznog signala. Da bi registar mogao da prihvati novu vrednost brojačkog registra, potrebno je obrisati IEF fleg SCR registra upisom logičke nule na njegovo mesto.

Modovi rada *quad timer* modula

Brojači *quad timer* modula mogu da broje u oba smera. Moduo brojanja može da se podešava proizvoljno u zavisnosti od sadržaja *COMPARE* registara, kao i sadržaja *LOAD* registra koji sadrži početnu vrednost brojača u trenutku otpočinjanja novog ciklusa. Brojači mogu da broje neprekidno, započinjući novi ciklus nakon završetka prethodnog ili da se zaustave nakon završetka ciklusa. Postoji ukupno 14 modova rada *quad timer* modula, koji se biraju podešavanjem stanja *Count Mode* (CM) bitova CTRL registra.

1. ***Stop mode.*** CM=000. U ovom modu brojač je isključen.
2. ***Count mode.*** CM=001. Brojač broji rastuće ivice odabranog signala takta. Ovaj mod se koristi kada je potrebno generisanje periodičnih zahteva za prekid ili za brojanje eksternih događaja.
3. ***Edge-Count mode.*** CM=010. Ovaj mod se razlikuje od prethodnog po tome što se u ovom slučaju broje obe ivice odabranog signala takta.

4. ***Gated-Count mode.*** CM=011. Brojač je aktivan samo dok je sekundarni ulazni signal tajmersko-brojačke jedinice aktivan. Ovaj mod se koristi kada se meri trajanje nekog eksternog događaja.
5. ***Quad-Count mode.*** CM=100. Ovaj mod rada je namenjen za prihvat signala sa inkrementarnog enkodera pozicije. Brojač prihvata i dekoduje signale sa primarnog i sekundarnog ulaza i na osnovu njihovog faznog stava i učestanosti određuje smer i takt brojanja kao što je ilustrovano na slici.



6. ***Signed-Count mode.*** CM=101. Brojač broji signale sa primarnog ulaza pri čemu smer brojanja zavisi od stanja na sekundarnom ulazu. Ako je signal na sekundarnom ulazu na visokom nivou, smer brojanja je na dole, dok je u suprotnom slučaju smer brojanja na gore.
7. ***Triggered-Count mode.*** CM=110. Brojač započinje brojanje rastućih ivica signala na primarnom ulazu nakon detektovanja rastuće ivice na sekundarnom ulazu. Brojanje se nastavlja do trenutka nove detekcije rastuće ivice sekundarnog ulaza ili dok brojač ne dostigne gornju granicu brojanja. Naredna rastuća ivica sekundarnog ulaza ponovo startuje brojanje.
8. ***One-Shot mode.*** Ovaj mod rada je podvarijanta *Triggered-Count* moda. Osim što su CM bitovi postavljeni na 110, setovan je i bit LENGTH, bitovi *Output Mode* (OM) su postavljeni na 101 i setovan je bit ONCE. Svi ovi bitovi pripadaju CTRL registru. Kao i u *Triggered-count* modu, brojač započinje brojanje nakon pozitivne tranzicije signala na primarnom ulazu. Nakon dostizanja gornjeg limita brojanja setuje se OFLAG signal. Ovaj signal se resetuje narednom pozitivnom tranzicijom signala na primarnom ulazu. Ovaj mod generiše kašnjenje između primarnog ulaza i OFLAG izlaza i može da se koristi za sinhronizaciju između PWM modula i A/D konvertora.
9. ***Cascade-Count mode.*** CM=111. U ovom režimu primarni kao ulaz brojača se bira izlaz (OFLAG) drugog brojača. Na ovaj način se povećava rezolucija brojanja. Detaljniji opis ovog moda može se naći u uputstvu za familiju procesora *DSP56F80x*.
10. ***Pulse-Output mode.*** Ovaj mod se aktivira ako su CM bitovi postavljeni u stanje 001 *(Count mode)* a OM bitovi na 111 i ako je ONCE bit CTRL registra setovan. Tada se na izlazu tajmersko-brojačkog kanala (signal OFLAG) dobija niz impulsa iste učestanosti kao na primarnom ulazu pri čemu je njihov broj jednak razlici broja upisanog u CMP1 registar

i inicijalizacione vrednosti (*LOAD* registar). Ovaj mod rada je koristan za upravljanje step motorima.

11. ***Fixed-Frequency PWM mode.*** Ovaj mod je podvarijanta *Count* moda (CM=001). Bitovi LENGTH i ONCE registra CTRL su resetovani a OM bitovi su postavljeni na 110. Učestanost generisanog PWM signala je jednaka učestanosti signala na primarnom ulazu podeljenoj sa 65536, dok je širina impulsa jednaka vrednosti upisanoj u *compare* registar pomnoženoj sa periodom primarnog takta.
12. ***Variable-Frequency PWM mode.*** Ovaj mod je takođe podvarijanta *Count* moda (CM=001) pri čemu su bitovi LENGTH i ONCE setovani a OM bitovi postavljeni na 100. Ovaj mod se razlikuje od prethodnog po tome što se učestanost i širina PWM impulsa određuje sadržajem CMP1 i CMP2 registara.

Registri *Quad-Timer* modula

Svaki tajmersko-brojački kanal poseduje po osam I/O registara. Svaki modul poseduje po četiri kanala pa je ukupan broj registara po modulu 32. U tabeli 3.3 su prikazani svi registri jednog tajmersko-brojačkog kanala, kao i njihova unutrašnja struktura.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TMRxn	CTRL	R	COUNT MODE			PRIMARY COUNT SOURCE				SECONDARY SOURCE		ONCE	LENGTH	DIR	Co Init	OUTPUT MODE		
		W																
TMRxn	SCR	R	TCF	TCFIE	TOF	TOFIE	IEF	IEFIE	IPS	INPUT	CAPTURE MODE		MSTR	EEOF	VAL		OPS	OEN
		W														FORCE		
TMRxn	CMP1	R	COMPARISON VALUE[15:0]															
		W																
TMRxn	CMP2	R	COMPARISON VALUE[15:0]															
		W																
TMRxn	CAP	R	CAPTURE[15:0]															
		W																
TMRxn	LOAD	R	LOAD[15:0]															
		W																
TMRxn	HOLD	R	HOLD VALUE[15:0]															
		W																
TMRxn	CNTR	R	COUNTER[15:0]															
		W																

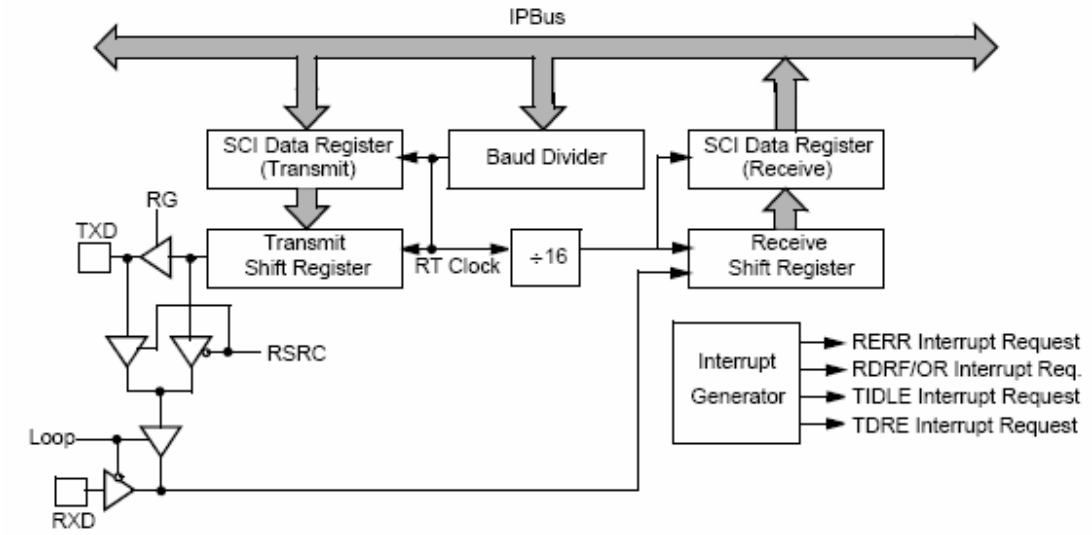
x = A,B,C, or D n = 0,1,2, 3

Tabela 3.3. Periferijski registri *Quad-Timer* modula

Sa TMRxn je označen adresni ofset u odnosu na baznu adresu *quad timer* modula. Procesor *DSP56F801* poseduje module C i D, što je u tabeli označeno sa x a sa n je označen redni broj kanala koji može biti od 0 do 3. Ovi registri su ukratko opisani u prethodnom tekstu, a detaljnije informacije se mogu naći u dokumentu *DSP56F801_UM.pdf* koji se nalazi na CD-u.

3.4. Serijski komunikacioni interfejs (SCI)

Preko serijskog komunikacionog interfejsa (SCI) se obavlja asinhrona serijska komuinikacija sa drugim uređajima. SCI interfejsu su dodeljena dva procesorska pina, TxD i RxD, preko kojih se odvija slanje i prijem podataka. Na slici 3.4 je prikazana blok šema SCI interfejsa.



3.4. SCI interfejs

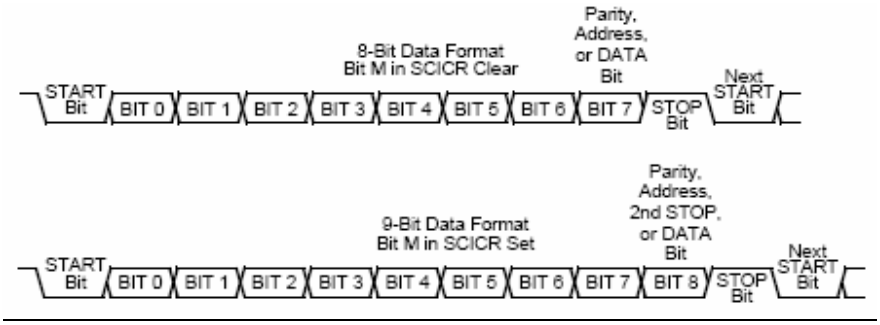
Kao što se vidi sa slike, SCI interfejs poseduje dva *data* registra koji su mapirani na istoj memorijskoj adresi. Procesor pristupa ovim registrima kao jedinstvenom SCIDR registru. Pri slanju, podatak se iz SCI *data transmit* registra kopira u pomerački registar gde mu se dodaju start, stop i bit parnosti. Za prijem podataka se koristi prijemni pomerački registar. Nakon što se podatak prekopira iz SCI *data transmit* registra, setuje se TDRE fleg statusnog registra SCISR. Ukoliko je setovan TEIE bit kontrolnog registra SCICR, generiše se zahtev za prekid *SCI Transmitter Ready*. TDRE fleg se briše čitanjem sadržaja SCISR registra i upisom novog podatka u SCIDR registar.

Kada se završi prijem podatka preko prijemnog pomeračkog registra, podatak se kopira u SCI *data receive* registar i setuje se RDRF fleg statusnog registra SCISR. Ako je setovan RFIE bit kontrolnog registra SCICR generiše se zahtev za prekid *SCI Receiver Full*. RDRF fleg se briše upisom logičke jedinice na njegovo mesto.

Pored prethodno opisanih zahteva za prekid, SCI modul može da generiše i dodatna dva zahteva za prekid. Jedan od njih, *SCI Transmit Complete,* se javlja nakon što se isprazni predajni pomerački registar, pri čemu se setuje TIDLE flag SCISR registra. Ovaj prekid se maskira bitom TIIE registra SCICR. Ako se prilikom prijema podatka detektuje greška setuje se odgovarajući

fleg u statusnom registru SCISR i generiše se *SCI Receiver Error* zahtev za prekid. Ovaj prekid se maskira REIE bitom registra SCICR.

Podaci koji se šalju i primaju SCI interfejsom mogu biti osmobitni ili devetobitni u zavisnosti od stanja bita M ragistra SCICR. Na slici 3.5. je prikazan format podataka koje se šalju ili primaju putem serijskog interfejsa.



3.5. Format podataka pri serijskoj komunikaciji

Sa slike se vidi da svaki novi podatak započinje start bitom, a završava stop bitom. Pre stop bita može da se nalazi bit parnosti.

SCI interfejs poseduje programabilni delitelj učestanosti IPbus takta kojim se kontroliše brzina prenosa podataka serijskom vezom. Brzina prenosa zavisi od sadržaja SCIBR registra prema sledećoj formuli:

$$SCI\ baud\ rate = \frac{IPBus\ Clock}{16 \cdot SCIBR}$$

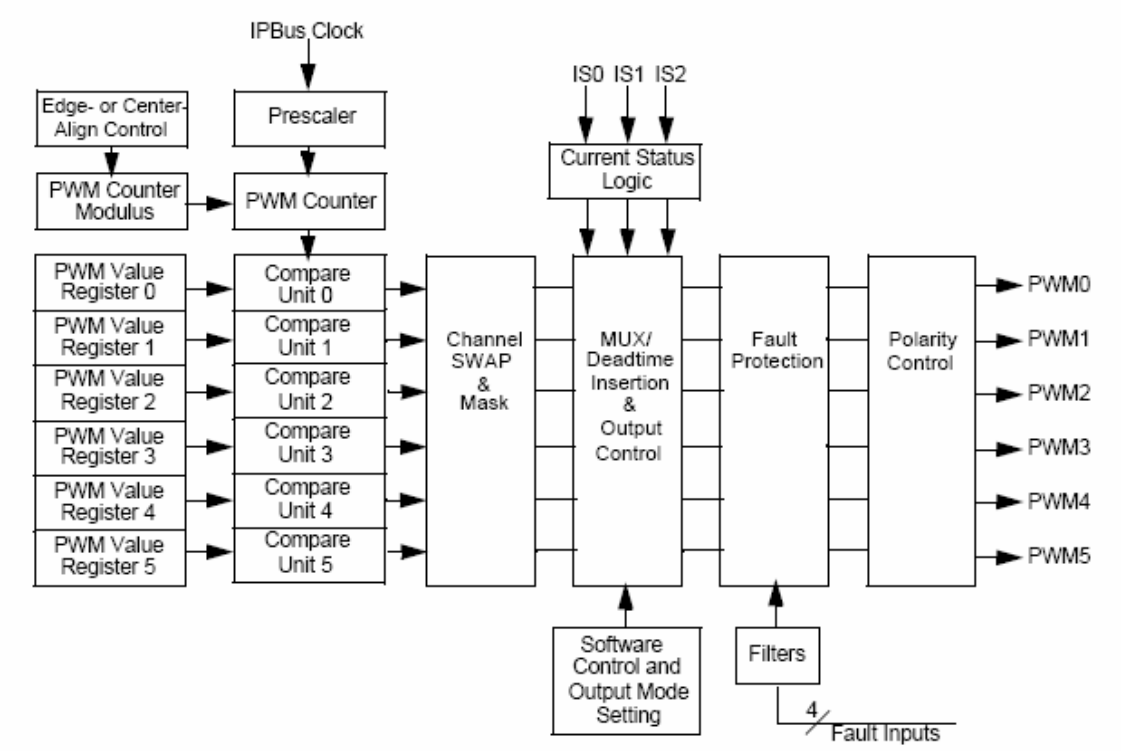
SCI baud rate je brzina prenosa podataka u *bps*. Na brzinu prenosa veliki uticaj ima tačnost učestanosti takta procesora. Ukoliko se koristi interni relaksacioni oscilator, učestanost takta može da varira od procesora do procesora. Zbog toga se preporučuje upotreba kvarcnog oscilatora. Ako se ipak koristi relaksacioni oscilator, neophodno je fino podesiti njegovu učestanost upisom odgovarajuće vrednosti u IOSCTL registar. Ovo je pokazano u primerima 3 i 4. U sledećoj tabeli je dat pregled I/O registara koji pripadaju SCI interfejsu.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	SCIBR	R	0	0	0	SBR												
		W																
$1	SCICR	R	LOOP	SWAI	RSRC	M	WAKE	POL	PE	PT	TEIE	TIIE	RFIE	REIE	TE	RE	RWU	SBK
		W																
$2	SCISR	R	TDRE	TIDLE	RDRF	RIDLE	OR	NF	FE	PF	0	0	0	0	0	0	0	RAF
		W																
$3	SCIDR	R	0	0	0	0	0	0	0	RECEIVE DATA								
		W								TRANSMIT DATA								

Tabela 3.4. Periferijski registri SCI interfejsa

3.5. PWM modul

Procesor DSP56F801 poseduje jedan PWM modul, čija je blok šema prikazana na slici 3.6. PWM modul može da generiše do šest PWM signala koji su dostupni na pinovima procesora. Ovi signali mogu da se generišu kao 3 komplementarna para ili 6 nezavisnih signala. PWM modul poseduje zajednički 15-bitni brojač za svih 6 kanala (*PWM counter*). Takt PWM brojača se dobija od *IPBus* takta nakon deljenja učestanosti programabilnim preskalerom. U režimu generisanja komplementarnih parova PWM signala, PWM modul omogućava automatsko ubacivanje mrtvog vremena *(deadtime)* između gašenja jednog i paljenja drugog tranzistora u istoj grani invertorskog mosta. PWM signali su izvedeni na pinove DSP-a, a ovim pinovima je moguće upravljati i softverski.

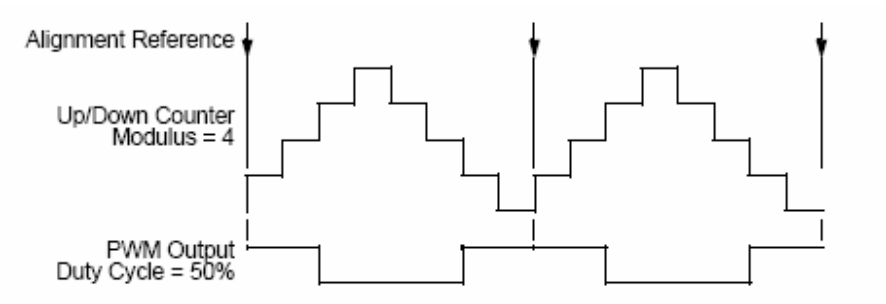


3.6. PWM modul

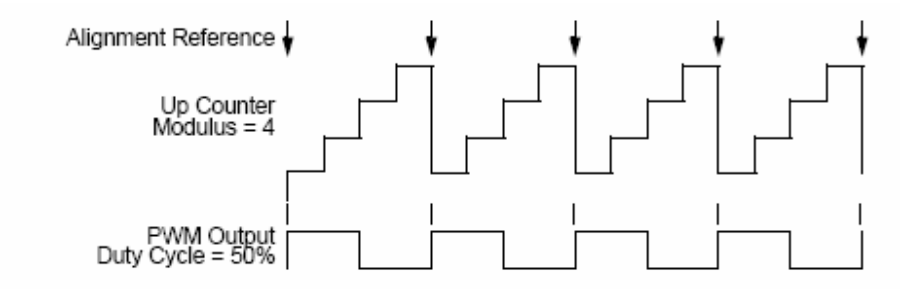
Pored PWM pinova, PWM modul koristi i *Fault* pinove. Na procesoru DSP56F801 postoji samo jedan *Fault* pin. Dovođenjem signala logičke jedinice na ovaj pin, PWM pinovi prelaze u stanje visoke impedanse.

Perioda PWM signala zavisi od učestanosti takta i od radnog režima PWM modula. Postoje dva radna režima i to su simetrični i asimetrični PWM. Kod simetričnog režima rada

brojač broji naviše dok ne dostigne zadatu maksimalnu vrednost, odnosno moduo brojanja. U tom trenutku brojač kreće da broji naniže dok ne dostigne nulu, a potom opet kreće da broji naviše itd. Na sledećim slikama su ilustrovana oba režima rada PWM modula.



3.7. Simetrični PWM mod



3.8. Asimetrični PWM mod

Na slikama 3.7 i 3.8 se vidi razlika između simetričnog i asimetričnog PWM režima. Pri radu u simetičnom režimu perioda PWM signala iznosi :

$$T_{PWM} = PWM\text{mod} \times T_{clk\ PWM} \times 2 \qquad (3.1)$$

$T_{clk\ PWM}$ je perioda signala takta PWM brojača. Ovaj takt se dobija iz preskalera koji deli učestanost periferijskog takta *IPBus.* $PWM\text{mod}$ je modul brojanja. Ova vrednost se upisuje u *PWMCM* registar. Tokom rada u asimetričnom režimu perioda PWM signala se izračunava po sledećoj formuli :

$$T_{PWM} = PWM\text{mod} \times T_{clk\ PWM} \qquad (3.2)$$

Vidi se da je kod asimetričnog režima rada perioda duplo manja, tj. PWM učestanost je duplo veća u odnosu na simetrični mod rada. Takođe, pri asimetričnom modu rada je veći sadržaj viših harmonika u izlaznom signalu.

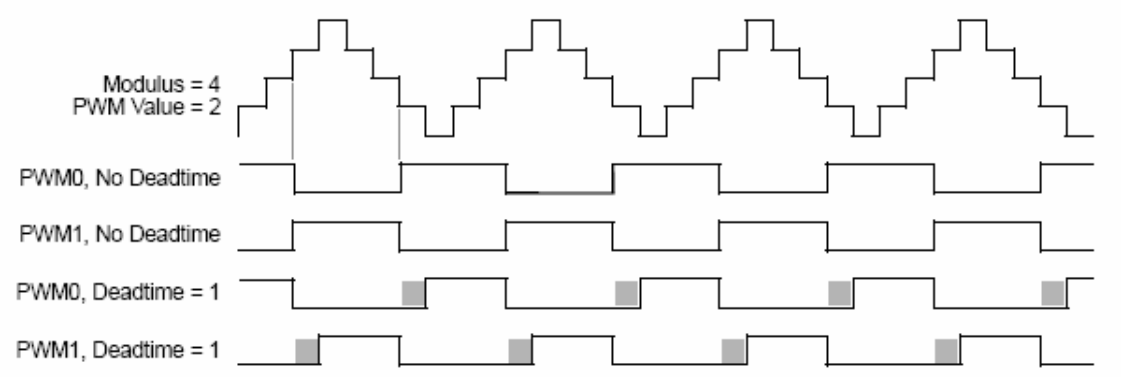
Širina impulsa tokom jedne periode PWM signala se može izračunati po sledećim formulama :

$$Pulse\ width = PWMvalue \times T_{clk\ PWM} \times 2 \quad (3.3)$$

$$Pulse\ width = PWMvalue \times T_{clk\ PWM} \quad (3.4)$$

Formula 3.3 važi u simetričnom režimu rada, dok je formula 3.4 za rad u asimetričnom režimu rada. $PWMvalue$ predstavalja sadržaj odgovarajućeg PWMVAL registra. Ovih registara ima šest, za svaki PWM pin po jedan.

PWM modul poseduje mogućnost ubacivanja mrtvog vremena između dva komplementarna izlazna signala. Mrtvo vreme se zadaje kao celi broj taktova *PWM* brojača. Ovo je iluistrovano na slici 3.9.



Slika 3.9. Dodavanje mrtvog vemena komplementarno paru PWM signala

PWM modul poseduje i mogućnost automatske kompenzacije mrtvog vremena. Za ovu svrhu se koriste posebni *Current Status* pinovi, na slici 3.6. označeni kao IS0 Ii1 i IS2. Na ove pinove je potrebno dovesti logičke signale koji reprezentuju trenutni smer struje na priključcima potrošača. Procesor *DSP56F801* ne poseduje *Current Status* pinove, ali poseduje flegove koji pripadaju ovim pinovima i koji se nalaze u registru *PMPORT.* Zahvaljujući ovim flegovima, kompenzacija mrtvog vremena se može obavljati softverski.

PWM modul može da generiše dve vrste zahteva za prekid. Jedan od njih je *PWM Reload* koji se generiše nakon završetka određenog broja ciklusa PWM brojača. Ovaj broj ciklusa se zadaje upisom odgovarajuće vrednosti u LDFQ polje registra PMCTL. Prekidna rutina treba da obezbedi nove vrednosti za PWMVALx registre, i eventualno PWMCM. Nakon što se završi zadati broj ciklusa, setuje se PWMF fleg registra PMCTL. Prekidna rutina se poziva ako je setovan PWMRIE bit istog registra.

Drugi tip prekida, *PWM Fault*, se javlja kad se na nekom od *FAULT* pinova pojavi logička jedinica. Ako je setovan odgovarajući FIEx bit PMFCTL registra, poziva se prekidna rutina. Procesor DSP56F8001 poseduje samo jedan FAULT pin, FAULTA0.

Registri PWM modula su dati u tabeli 3.5.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	PMCTL	R	LDFQ				HALF	IPOL2	IPOL1	IPOL0	PRSC		PWM RIE	PWMF	ISENS		LDOK	PWM EN
		W																
$1	PMFCTL	R	0	0	0	0	0	0	0	0	FIE3	FMODE3	FIE2	FMODE2	FIE1	FMODE1	FIE0	FMODE0
		W																
$2	PMFSA	R	FPIN3	FFLAG3	FPIN2	FFLAG2	FPIN1	FFLAG1	FPIN0	FFLAG0	0	0	DT5	DT4	DT3	DT2	DT1	DT0
		W										FTACK3		FTACK2		FTACK1		FTACK0
$3	PMOUT	R	PAD_ EN	0	OUT CTL5	OUT CTL4	OUT CTL3	OUT CTL2	OUT CTL1	OUT CTL0	0	0	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0
		W																
$4	PMCNT	R	0	CNT														
		W																
$5	PWMCM	R	0	CM														
		W																
$6-$B	PWMVAL0-5	R	VAL															
		W																
$C	PMDEADTM	R	0	0	0	0	0	0	0	0	PWMDT							
		W																
$D	PMDISMAP1	R	DISMAP[15:0]															
		W																
$E	PMDISMAP2	R	0	0	0	0	0	0	0	0	DISMAP[23:16]							
		W																
$F	PMCFG	R	0	0	0	EDG	0	TOP NEG 45	TOP NED 23	TOP NEG 01	0	BOT NEG 45	BOT NEG 23	BOT NEG 01	INDEP 45	INDEP 23	INDEP 01	WP
		W																
$10	PMCCR	R	ENHA	0	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	0	0	VLMODE		0	SWP 45	SWP 23	SWP 01
		W																
$11	PMPORT	R	0	0	0	0	0	0	0	0	0	IS2	IS1	IS0	FAULT 3	FAULT 2	FAULT 1	FAULT 0
		W																

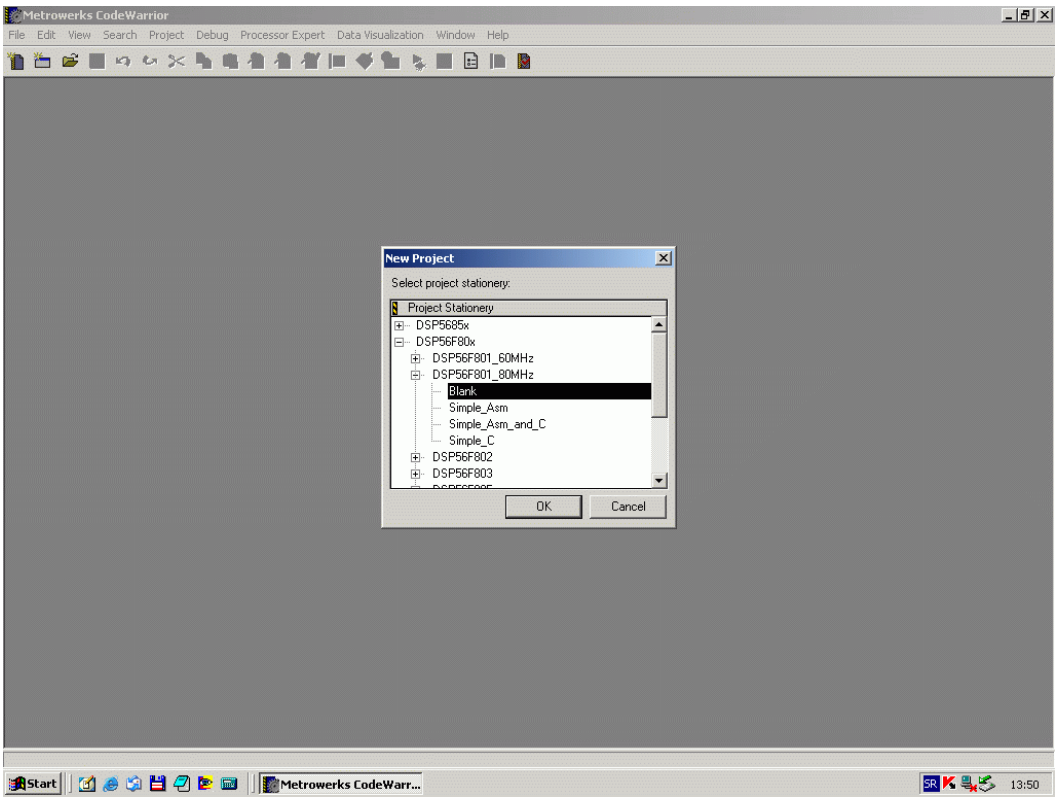
Tabela 3.5. Periferijski registri PWM modula

U primeru 4 je prikazana realizacija trofazne Space-Vector modulacije upotrebom PWM modula.

4.Code Warrior razvojno okruženje

4.1. Započinjanje novog projekta

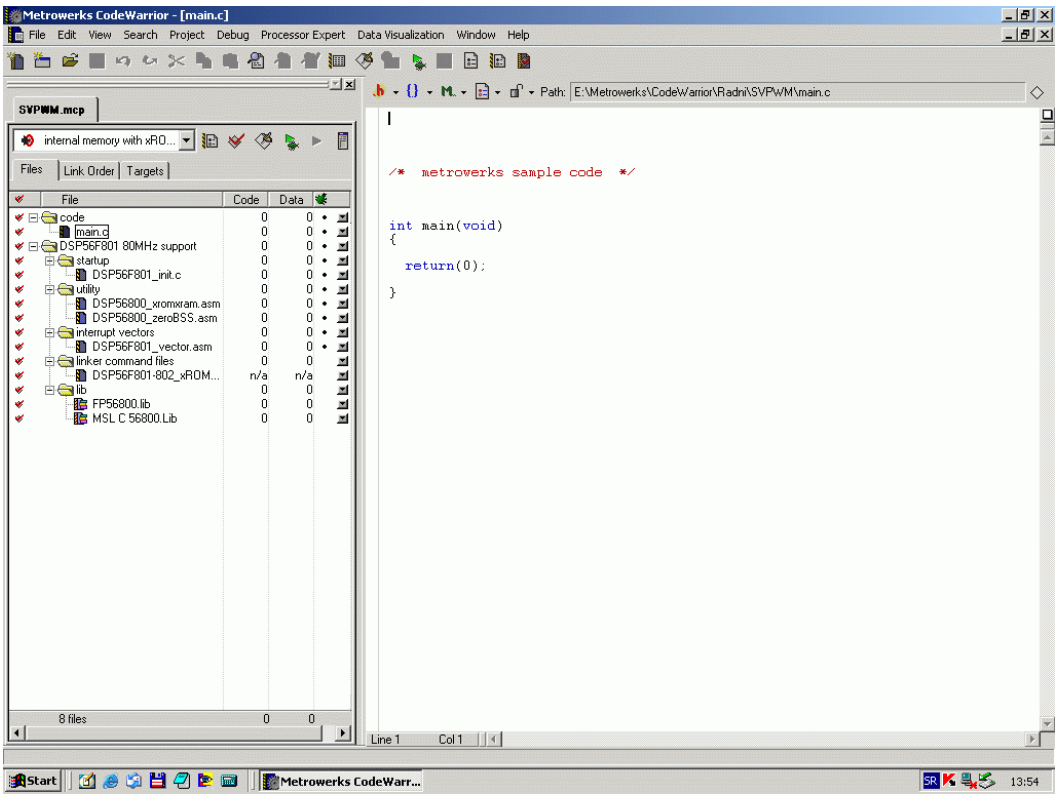
Novi projekat se započinje izborom opcije *New* koja se nalazi u meniju *File*. Ovim se otvara *new project wizard* koji omogućava izbor između nekoliko različitih vrsta projekata. Svi primeri koji su opisani u ovom radu su kreirani kao *DSP56800x EABI Stationery* projekti, pa će ovde biti reči samo o ovoj vrsti projekata. Pre nego što se pređe na naredni korak potrebno je uneti ime projekta i njegovu lokaciju na disku. Nakon što se klikne na *OK* otvara se prozor u kome je potrebno izabrati tip procesora za koji se razvija kod kao i vrstu koda koja se razvija. Ovde je najbolje odabrati opciju *Blank,* kao što je prikazano na slici 4.1.



Slika 4.1. Započinjanje novog projekta

Izborom *DSP56800x EABI Stationery* tipa, kreira se projekat u kome će razvojno okruženje generisati sve neophodne fajlove sa *start-up* kodom za inicijalizaciju. Pod *start-up* kodom se podrazumeva tabela vektora prekidnih rutina, asemblerske funkcije za inicijalizaciju

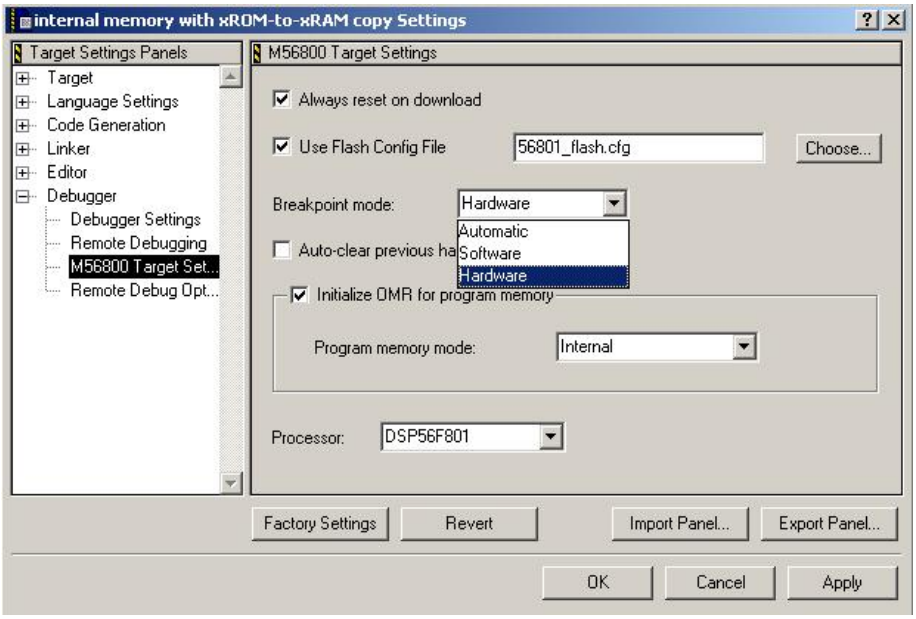
procesora, komandni fajl za povezivanje *(linker command file)* kao i prazna *main* funkcija. Dakle, nakon do sada opisanih koraka dobija se novi projekat i ekran prikazan na slici 4.2.



Slika 4.2. Novi *DSP56800x EABI Stationery* projekat

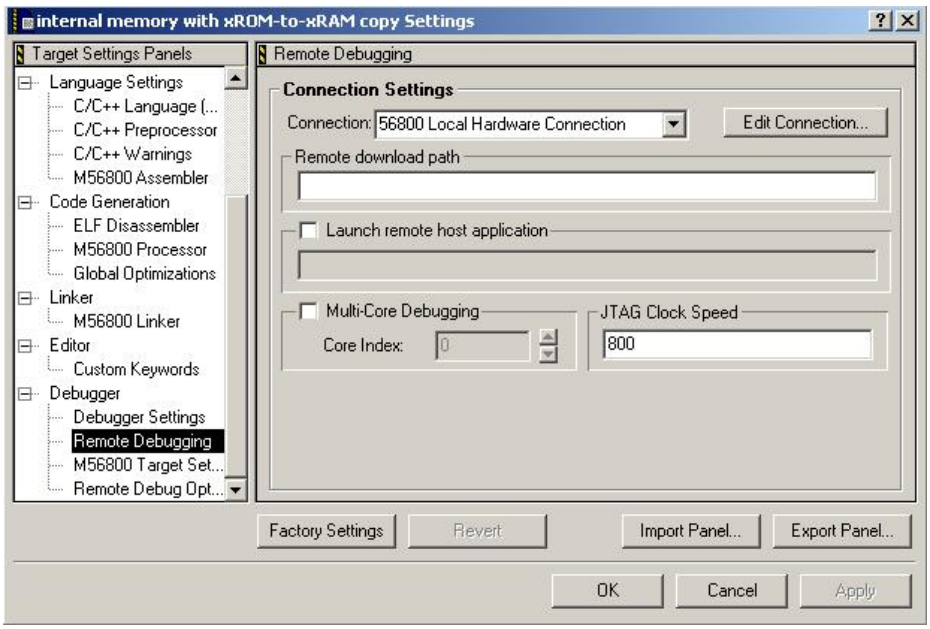
Na levom kraju ekrana se nalazi *Project Manager* u kome se nalaze izlistani svi fajlovi koji pripadaju datom projektu. *Project Manager* omogućava i rad sa više projekata istovremeno. Takođe, iz *Project Manager-a* je moguće konfigurisati svaki projekat pojedinačno. Kao što se vidi u *Project Manager-u*, razvojno okruženje je generisalo nekoliko fajlova koji su pridruženi projektu, kao i *main.c* fajl u kome se nalazi prazna *main()* funkcija.

Pritiskom na tastere Alt+F7 otvara se prozor za konfigurisanje novog projekta. Ovaj prozor može da se otvori i izborom opcija *Edit→internal memory with xROM-to-xRAM copy Settings...* Na levoj strani prozora se nalazi polje *Target Settings Panels* u kome je potrebno pronaći opciju *Debugger* u okviru koje treba kliknuti na podopciju *M56800 Target Settings.* Na desnoj strani prozora se pojavljuju parametri za podešavanje. Parametar *Breakpoint mode* treba da bude podešen na *Hardware,* kao što je prikazano na slici 4.3.



Slika 4.3. Podešavanje *Breakpoint* moda

U okviru opcije *Debugger,* klikom na *Remote Debugging* se otvara polje za podešavanje načina na koji će se obavljati debagovanje programa. Ako se želi hardversko debagovanje putem JTAG porta potrebno je u polju *Connection Settings* parametar *Connection* podesiti da bude *56800 Local Hardweare Connection*. U polju *JTAG Clock Speed* treba da piše 800. Ova podešavanja su prikazana na slici 4.4.



Slika 4.4. Podešavanje *Remote Debugging* parametara

4.2. DSP56800x EABI Stationery

U prethodnom tekstu je već navedeno da *DSP56800x EABI Stationery* projekat sadrži sav neophodan *start-up* kod za novi projekat. Ovde je dat detaljniji pregled svih funkcija od kojih se sastoji *start-up* kod.

U *Project Manager-u* postoji direktorijum *DSP56F80x xxMHz support* u koji su smešteni fajlovi sa *start-up* kodom. Prvi od njih je fajl *DSP56F801_init.c*. Unutar ovog fajla se nalazi definicija funkcije *init_M56801_()*. Funkcija *init_M56801_()* je napisana u asembleru i obavlja sledeći niz koraka :

1) Brisanje hardverskog steka i izbor linearnog nacina adresiranja
2) Inicijalizacija *IPR* i *COPCTL* registara
3) Inicijalizacija stek pointera *SP*
4) Inicijalizacija generator takta tako da se koristi interni relaksacioni oscilator
5) Kopiranje konstanti iz ROM u RAM memoriju
6) Brisanje *bss* zone u RAM memoriji
7) Pozivanje *main* funkcije

U direktorijumu *utility* su smeštena dva fajla sa asemblerskim funkcijama za kopiranje konstanti iz ROM-a u RAM, i sa brisanje *bss* sektora u RAM memoriji.

U direktorijumu *interrupt vectors* se nalazi fajl *DSP56F801_vector.asm* koji sadrži tabelu vektora prekidnih rutina. Inicijalno su svi vektori isti i upućuju na *M56801_intRoutine* koja je definisana na početku fajla. U slučaju da se koristi neki od prekida, neophodno je na odgovarajuću poziciju u tabeli ubaciti vektor prekidne rutine za taj prekid. Ovo je detaljnije objašnjeno kroz primere.

Naredni direktorijum *linker command files* sadrži komandni fajl kojim se definiše način smeštanja koda i podataka u memorijski prostor procesora. Detaljno objašnjenje sintakse komandnog fajla može se pronaći u dokumentu *Targeting_56F800.pdf.*

U direktorijumu *lib* nalaze se programske biblioteke koje se koriste u projektu.

U ovako generisan osnovni projekat se lako dadaju fajlovi sa C ili asemblerskim kodom. *DSP56800x EABI Stationery* projekat oslobađa programera od pisanja osnovnih rutina za inicijalizaciju čime se olakšava razvoj softvera za ovu klasu digitalnih signalnih kontrolera.

5. Pristup registrima periferijskih jedinica iz C koda

Periferijske jedinice procesora iz familije DSP56F80x poseduju memorijski mapirane ulazno izlazne (I/O) registre. Ovi registri su smešteni u RAM memoriji na lokacijama od 0C00h do 0FFFh, za procesore DSP56F801 do DSP56F805. Kod procesora DSP56F807 memorijski mapirani registri periferijskih jedinica su smešteni na lokacije od 1000h do 17FFh. Kako se ovi registri nalaze u DATA RAM memoriji, njima se može pristupiti na isti način kao što se pristupa i podacima smeštenim u RAM memoriju dakle korišćenjem *mov* instrukcija i odgovarajućeg načina adresiranja. Dakle jedan od načina za pristup ovim registrima iz C koda je korišćenjem *inline* asemblerskih instrukcija. Ovakav način rada sa I/O registrima ima ozbiljan nedostatak u veoma lošoj preglednosti koda koji se na ovakav način dobija. Takođe, samo pisanje je veoma otežano i usporeno jer programer treba da poznaje tačne adrese pojedinačnih registara kao i pozicije pojedinačnih statusnih i kontrolnih bitova u svakom registru. Pisanje i održavanje koda bi bilo mnogo jednostavnije ukoliko bi se periferijskim registrima moglo pristupati kao i svakoj drugoj promenljivoj u C programu. U ovu svrhu je, u okviru ovog rada, razvijen tzv. sloj za apstrakciju hardvera *(hardware abstraction layer)* koji zahvaljujući korišćenju registarskih fajlova i polja bitova omogućava jednostavan pristup kako I/O registrima periferijskih jedinica, tako i pojedinačnim bitovima ili grupama bitova u okviru ovih registara. U narednim poglavljima je opisana realizacija sloja za apstrakciju hardvera i njegovo korišćenje u projektima.

5.1. Hardware abstraction layer

Kao što je već napomenuto, sloj za apstrakciju hardvera se formira korišćenjem registarskih fajlova i polja bitova. Svaka periferijska jedinica poseduje grupu I/O registara. Registarski fajl je C/C++ struktura koja kao članove sadrži sve I/O registre koji pripadaju nekoj određenoj periferijskoj jedinici. Ovi registarski fajlovi se smeštaju u RAM memoriju na odgovarajuće memorijske lokacije tako da se svaki član strukture nalazi na memorijskoj lokaciji na kojoj je fizički mapiran I/O registar predstavljen tim članom strukture. Na ovaj način je omogućeno da se registrima periferijskh jedinica pristupa kao promenljivima, članovima strukture, iz C/C++ koda. Da bi pristup I/O registrima bio još fleksibilniji, koristi se još jedna mogućnost jezika C/C++. Naime, programski jezik C/C++ dozvoljava kreiranje struktura čiji se elementi (polja) mogu sastojati od svega nekoliko bitova ili čak samo jednog bita. Zahvaljujući toj činjenici moguće je kreirati strukturu koja se sastoji od pojedinačnih bitova tako da ovakva struktura predstavlja neki periferijski registar. Tako je moguće kreirati odgovarajuću strukturu za svaki I/O registar koji pripada jednoj periferijskoj jedinici. Na kraju se formira struktura – registarski fajl čiji

su elementi prethodno definisane strukture – registri. Ovim je omogućen pristup pojedinačnim bitovima svakog I/O registra za koji je definisana struktura sastavljena od pojedinačnih bitova. U daljnjem tekstu je naveden postupak formiranja registarskog fajla za PWM modul kao i njegovo smeštanje počev od odgovarajuće memorijske lokacije.

Primer 1. Struktura kojom se definišu bitovi PWM Control registra (PMCTL)

```
// PWM individual register bit definitions :

struct PMCTL_BITS                   // bits   description
{
        Uint16 PWMEN:1;             // 0      PWM enable
        Uint16 LDOK:1;              // 1      Load okay
        Uint16 ISENS:2;             // 3:2    Current status
        Uint16 PWMF:1;              // 4      PWM reload flag
        Uint16 PWMRIE:1;            // 5      PWM reload interrupt enable
        Uint16 PRSC:2;              // 7:6    Prescaler
        Uint16 IPOL0:1;             // 8      Current polarity 0
        Uint16 IPOL1:1;             // 9      Current polarity 1
        Uint16 IPOL2:1;             // 10     Current polarity 2
        Uint16 HALF:1;              // 11     Half cycle reload
        Uint16 LDFQ:4;              // 15:12  Load frequency bits
};
```

Ovom strukturom su definisani svi bitovi *PWM Control* registra. *Uint16* je identifikator tipa promenljivih *unsigned int* koji su širine 16 bita. Pojedina polja su široka po nekoliko bita kao što je ISENS dva bita ili LDQF od 4 bita. Da bi se omogućio pristup čitavom registru potrebno je definisati uniju prethodno definisane strukture i celobrojne šesnaestobitne promenljive koja će predstavljati ceo registar.

Primer 2. Unija prethodno definisane strukture i celobrojne šesnaestobitne promenljive

```
union PWM_PMCTL_REG
{
        Uint16 all;
        struct PMCTL_BITS bit;
};
```

Oba elementa ove unije se smeštaju u isti memorijski prostor od jedne reči (16 bita). Na isti način se definišu strukture bitova za ostale registre PWM modula kao i unije iz primera 2 za svaki pojedinačni registar osim za one registre čiji pojedinačni bitovi nemaju nikakvu dodeljenu funkciju. Radi se o registrima kojima ima smisla pristupati samo kao celini. Na kraju je potrebno definisati strukturu koja predstavlja registarski fajl za konkretnu periferijsku jedinicu, u ovom slučaju PWM modul.

Primer 3. Struktura koja čini registarski fajl PWM modula

```
struct PWM_REGS
{
        union PWM_PMCTL_REG PMCTL;       // PWM Control register
        union PWM_PMFCTL_REG PMFCTL;// PWM Fault control register
        union PWM_PMFSA_REG PMFSA;       // PWM Fault status & acknowledge register
        union PWM_PMOUT_REG PMOUT;       // PWM Output control register
        Uint16 PMCNT;                    // PWM Counter register
        Uint16 PWMCM;                    // PWM Counter modulo register
        Uint16 PWMVAL0;                  // PWM Value register 0
        Uint16 PWMVAL1;                  // PWM Value register 1
        Uint16 PWMVAL2;                  // PWM Value register 2
        Uint16 PWMVAL3;                  // PWM Value register 3
        Uint16 PWMVAL4;                  // PWM Value register 4
        Uint16 PWMVAL5;                  // PWM Value register 5
        Uint16 PMDEADTM;                 // PWM Deadtime register
        Uint16 PMDISMAP1;                // Dissable mapping register 1
        Uint16 PMDISMAP2;                // Dissable mapping register 2
        union PWM_PMCFG_REG PMCFG;       // PWM Configure register
        union PWM_PMCCR_REG PMCCR;       // PWM Chanell control register
        union PWM_PMPORT_REG PMPORT;  // PWM Port register
};
```

Kao što se vidi u primeru 3, prva četiri člana strukture su unije. Prva u nizu je unija PMCTL koja je tipa PWM_PMCTL_REG definisanog u primeru 2. Ova unija predstavlja PMCTL registar PWM modula. Na isti način sledeća unija predstavlja PMFCTL registar, itd. Elementi tipa *Uint16* su registri kojima se može pristupiti samo kao celini : PMCNT, PWMCM, PWMVAL0-5, PMDEADTM, PMDISMAP1-2. Preostala tri elementa su ponovo unije, odnosno registri čijim se pojedinačnim bitovima može pristupati. Kao što je opisano, svaki element strukture iz primera 3 predstavlja pojedinačan registar PWM modula. Elementi strukture su složeni onim redosledom kojim su pojedinačni registri poređani u memoriji. Ova struktura sama po sebi nije varijabla koja može da se smesti u neki memorijski prostor. Da bi pristup I/O registrima bio moguć potrebno je formirati promenljivu tipa *struct PWM_REGS* i smestiti je u odgovarajući memorijski prostor. Familija *DSP56F80x* generalno poseduje dva PWM modula A i B pa je potrebno definisati po jednu promenljivu za svaki modul.

Primer 4. Definicije promenljivih PWMA i PWMB

```
volatile struct PWM_REGS PWMA;
volatile struct PWM_REGS PWMB;
```

Promenljiva PWMA predstavlja PWM modul A, a PWMB predstavlja PWM modul B. Sada su definisane promenljive koje linker može da pozicionira na odgovarajuće lokacije u memoriji. Na početku definicije dodat je modifikator *volatile* kojim se prevodiocu saopštava da se sadržaj ovih promenljivih može promeniti mimo kontrole programa, to jest od strane samih periferijskih

jedinica. Dodavanjem modifikatora *volatile* sprečava se mogućnost da prevodilac izvrši optimizaciju koda na mestu gde se pristupa I/O registru.

Da bi se registarski fajlovi PWMA i PWMB smestili na odgovarajuće pozicije u memoriji poterbno je formirati sekcije odnosno elemente koje linker može da pozicionira tačno na zahtevano mesto u memoriji. Da bi se kreirala sekcija poterbno je definicijama promenljivih iz primera 4 dodati odgovarajuće #pragma direktive. U primeru 5 je data kompletna definicija promenljive PWMA tipa *struct PWM_REGS* sa definicijom sekcije kojoj ta promenljiva pripada. Slična definicija je potreban i za promenljivu PWMB.

Primer 5. Defininicija promenjljive PWMA koja pripada sekciji PWMA_reg

```
#pragma define_section PWMA_reg "PWMA" RW
#pragma section PWMA_reg begin

volatile struct PWM_REGS PWMA;

#pragma section PWMA_reg end
```

U prvom redu je direktivom *#pragma define_section* definisana sekcija PWMA_reg kojoj se dodeljuje identifikator “PWMA”. U drugom redu se direktivom *#pragma section* označava početak definicije PWMA_reg sekcije. U trećem redu je definisana promenjliva PWMA, dok se u četvrtom redu završava definicija sekcije PWMA_reg. Na isti način su definisane odgovarajuće promenljive odnosno registarski fajlovi i njima pridružene sekcije za svaku periferijsku jedinicu procesora *DSP56F80x.* Sve ove definicije su smeštene unutar fajla pod imenom *DSP56F801_GlobalVariables.c.* Poslednji korak je da se u linker komandnom fajlu svaka prethodno definisana sekcija smesti počev od memorijske lokacije na kojoj se nalazi prvi registar odgovarajuće periferijske jedinice.

Pristup I/O registrima

U primeru 6 je dato nekoliko primera pristupanja I/O registrima i njihovim bitovima korišćenjem prethodno opisanog *hardware abstraction layer-a*.

Primer 6. Pristup I/O registrima upotrebom hardware abstraction layer-a

```
PWMA.PMCTL.bit.LDOK=1;
PWMA.PMCTL.bit.PWMEN=1;

GPIOB.DDR.bit.DDR5=1;
GPIOB.DR.bit.DR5=0;

ADCA.ADCR1.all=0x0800;
ADCA.ADCR2.all=0x0004;
```

PWMA, GPIOB i ADCA su podaci strukturnog tipa čije se definicije nalaze u fajlu *DSP56F801_GlobalVariables.c.* PWMA je struktura tipa *struct PWM_REGS* čiji elementi predstavljaju pojedinačne registre PWM modula A. Ovi elementi mogu biti promenljive tipa *unsigned int* ili unije koje se sastoje od promenljive *all* tipa *unsigned int* i strukture pod imenom *bit* sastavljene od bitova kao što je pokazano u primerima 2 i 3. Elementima strukture se pristupa pomoću operatora tačka (.). Dakle, u prvom redu primera 6 pristupa se registru PMCTL koji pripada PWM modulu A. Pošto se registru PMCTL može pristupiti kao celini, ili samo nekom od njegovih bitova poterbno je navesti kom elementu unije PMCTL se pristupa. Ako se pristupa nekom od bitova potrebno je navesti identifikator *.bit* i nakon njega ime bita kojem se pristupa. U ovom slučaju je to bit Load Okay (LDOK) kome se dodeljuje vrednost 1. Ovde je potrebno napomenuti da se pojedinačnim bitovima može dodeliti samo vrednost 0 ili 1, dok se grupama bitova mogu dodeliti vrednosti u zavisnosi od broja bitova koji čine grupu. U slučaju da se bitu dodeli vrednost različita od 0 ili 1 ili ako se grupi bitova dodeli vrednost koja se ne može predstaviti tom grupom bitova, prevodilac neće javiti da je došlo do greške pa je o tome potrebno voditi računa. U naredna tri reda primera 6 je pokazano setovanje bitova PWM enable (PWMEN) registra PMCTL kao i bita 5 data direction (DDR) registra modula GPIOB. U poslednja dva reda je prikazano kako se pristupa registrima ADCR1 i ADCR2 kao celini. Dakle navođenjem polja *.all* nakon imena registra. Registri ADCR1 i ADCR2 pripadaju analogno-digitalnom konvertoru ADCA.

5.2. Upotreba hardware abstraction layer-a u novim projektima

Hardware abstraction layer se sastoji od više *header* fajlova sa definicijama struktura i unija opisanih u prethodnom poglavlju, fajla *DSP56F801_GlobalVariables.c* sa definicijama globalnih podataka i komandnog fajla za linker *DSP56F801_linker.cmd.* Za svaku periferijsku jedinicu postoji zaseban *header* fajl. Svi *header* fajlovi sa definicijama registarskih fajlova periferijskih jedinica se uključuju u izvorni kod programa pomoću fajla *DSP56F80x_device.h* čiji je sadržaj izlistan u primeru 7. U svaki .c fajl novog projekta, u kome se nalazi kod koji pristupa registrima periferijskih jedinica, potrebno je na početku ubaciti direktivu :

#include “DSP56F80x_device.h”

Osim ubacivanja ove direktive, neophodno je u projekat uključiti i fajl *DSP56F801_GlobalVariables.c.* Na kraju, u *linker command files* direktorijum *project manager-a* treba dodati fajl *DSP56F801_linker.cmd* umesto .lcf fajla koji se dobija kreiranjem novog *EABI Stationery* projekta. Sve fajlove koji se pozivaju u *DSP56F80x_device.h* kao i sam fajla *DSP56F80x_device.h* zajedno sa *DSP56F801_GlobalVariables.c* treba fizički iskopirati u direktorijum novog projekta, dok je fajl *DSP56F801_linker.cmd* je potrebno prekopirati u *lcf* poddirektorijum unutar direktorijuma novog projekta. Nakon toga je *hardware abstraction layer* spreman za korišćenje u novom projektu.

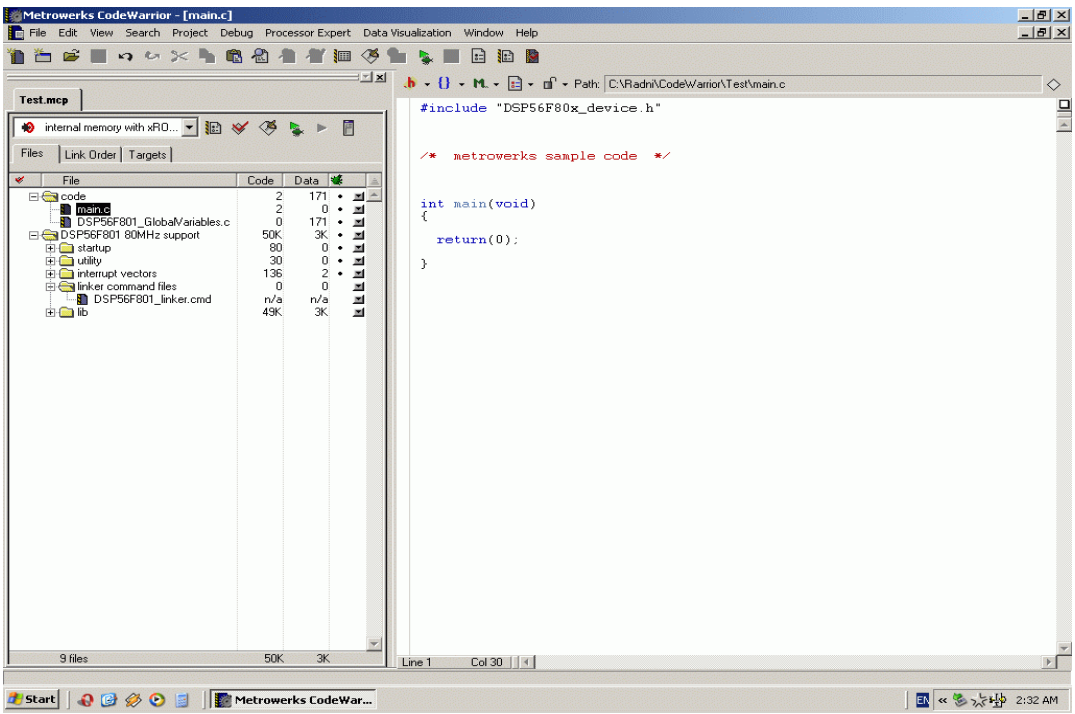
Primer 7. DSP56F80x_device.h

```
#ifndef _IO_REG_H
#define _IO_REG_H

#ifndef _UINT16
#define _UINT16
#define Uint16 unsigned int
#endif // _UINT16

#include "DSP56F80x_OCCS.h"
#include "DSP56F80x_PWM.h"
#include "DSP56F80x_SCI.h"
#include "DSP56F80x_SPI.h"
#include "DSP56F80x_QuadTimer.h"
#include "DSP56F80x_ADC.h"
#include "DSP56F80x_GPIO.h"
#include "DSP56F80x_ITCN.h"

#define IPR                0xFFFB          // Interrupt priority register

#endif _IO_REG_H
```

Dakle, uključivanje *hardware abstraction layer*-a u novi projekat se sastoji od sledećih koraka:

1) Pritiskom na Alt+F7 otvoriti prozor *Settings.* U *target settings panels* odabrati polje *Access Paths* i klikom na dugme *Add…* dodati putanjedo direktorijuma *..IO_support\Headers, ..IO_support\Code* i *..IO_support\cmd.*
2) Desnim klikom miša na poddirektorijum *source* direktorijuma *code* unutar *project manager*-a otvoriti meni i izabrati opciju *Add Files...*
3) U prozoru *Select files to add* odabrati fajl *DSP56F801_GlobalVariables.c* iz direktorijuma *IO_support\Code*.
4) Desnim klikom miša na poddirektorijum *linker command files* direktorijuma *DSP56F801 60-80MHz support* unutar *project manager*-a otvoriti meni i izabrati opciju *Add Files...*
5) U prozoru *Select files to add* odabrati fajl *DSP56F801_linker.cmd* iz direktorijuma *..IO_support\cmd*
6) Iz poddirektorijuma *linker command files* izbaciti stari komandni fajl *DSP56F801-802_xROM-xRAM_linker.cmd* desnim klikom na ovaj fajl i izborom opcije *Remove.*
7) Dodati direktivu ***#include "DSP56F80x_device.h"*** na početku svakog .c fajla novog projekta.

Na slici 5.1. je prikazan izgled ekrana sa otvorenim novim projektom prilagođenim po prethodno opisanim koracima.



Slika 5.1. Izgled ekrana nako podešavanja po tačkama 1-7

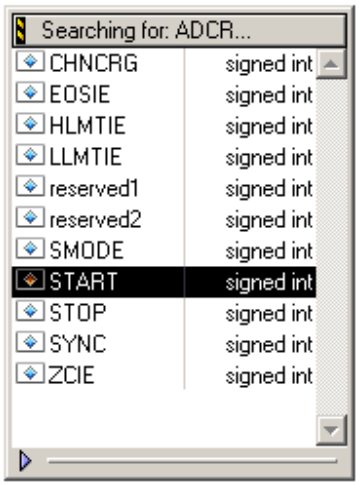
5.3. Prednosti upotrebe hardware abstraction layer-a

Osnovna prednost primene *hardware abstraction layer*-a se ogleda u jednostavnijem pisanju pregledu i održavanju koda. Registrima i pojedinačnim bitovima, ili poljima bitova, se pristupa navođenjem njihovog naziva iz dokumentacije. Nije potrebno poznavanje adresa registara u memoriji ili izračunavanje maski da bi se određeni bitovi setovali ili resetovali.

Primena ovog koncepta dodatno olakšava pisanje koda zahvaljujući *auto complete* funkciji CodeWarrior razvojnog okruženja. Kao što se može zaključiti iz prethodnog poglavalja, primena koncepta struktura i polja bitova u pristupu registrima periferijskih jedinica dovodi do pojave složenih imena elemenata kojima se želi pristupiti. Upotrebom *auto complete* funkcije, razvojno okruženje nudi listu mogućih imena struktura ili polja bitova tokom samog pisanja imena elementa kome se pristupa. U primeru 1 je ilustrovan pristup bitu *START* registra *ADCR1* analogno-digitalnog konvertora *ADCA.*

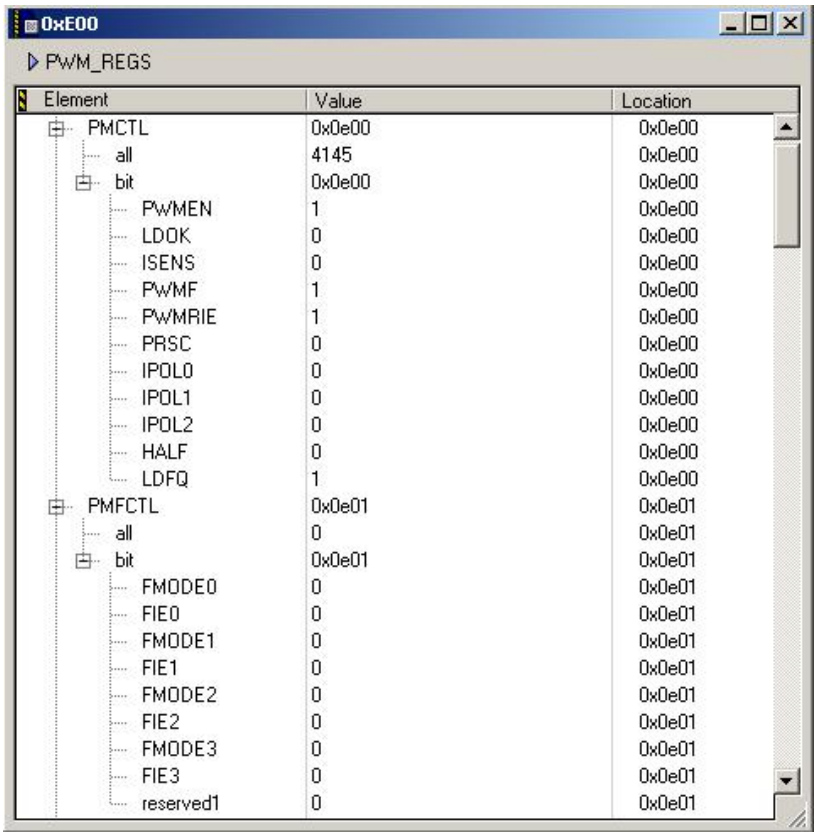
Primer 1. Upotreba auto complete funkcije CodeWarrior razvojnog okruženja.

```
ADCA.ADCR1.bit.
```



Prvo se navodi ime periferije kojoj se pristupa a to je ADCA. Nakon što se posle imena ADCA stavi tačka otvara se prozor koji nudi listu registara A/D konvertora odnosno listu elemenata njegovog registarskog fajla. Nakon što se izabere ADCR1, javlja se mogućnost izbora opcije *bit* ili *all.* Izborom opcijea *bit* otvara se lista prikazana u primeru 1 sa listom imena bitova registra ADCR1. Odabira se polje START. Dakle, upotrebom funkcije *auto complete* pisanje koda se znatno ubrzava i dosta olakšava jer nije potrebno znati tačan naziv svakog registra ili bita. *Auto complete* opcija se uključuje otvaranjem *project settings* prozora pritiskom na tastere alt+F7. U *Target Settings Panels* treba odabrati polje *Build Extras* a zatim, na desnoj strani prozora, za *Generate Browser Data From:* odabrati *Language Parser.* Ovim je *auto complete* funkcija aktivirana.

Još jedna prednost upotrebe *hardvare abstraction layer-a* je mogućnost praćenja stanja periferijskih registara ili njihovih pojedinačnih bitova korišćenjem *Watch* prozora kao i za sve druge promenljive. Dovođenjem kursora na ime promenljive i desnim klikom može se odabrati opcija *View Variable* čime se otvara *watch* prozor u kome se vidi sadržaj odgovarajuće promenljive odnosno u ovom slučaju periferijskog registra ili nekog njegovog bita. Izborom opcije *View as…* i izborom sa liste tipova podataka tipa koji odgovara registarskom fajlu čijem registru se pristupa, ceo registarski fajl se ubacuje u *watch* prozor. Na slici 5.2 je prikazan *watch* prozor koji se dobija izborom tipa *PWM_REGS.*



Slika 5.2.

Kao što se vidi na slici 5.2, moguće je jednostavno nadgledanje svih registara koje pripadaju nekoj periferijskoj jedinici. Moguća je i promena sadržaja ovih registara ili pojedinačnih bitova tokom procesa debagovanja iz *watch* prozora, što dodatno olakšava razvoj softvera.

Na kraju se može zaključiti da je upotreba *hardware abstraction layer-a* daleko praktičnija od tradicionalnog pristupa upotrebom *#define* direktiva ili asemblerskih instrukcija.

6. Demonstracioni modul CSM-56F801

Demonstracioni modul CSM-56F801 je namenjen za testiranje mogućnosti procesora *DSP56F801* kao i za razvoj aplikacija. Na slici 6.1. je prikazan izgled CSM56F801 modula.

Slika 6.1. CSM56F801 modul

Na štampanoj ploči CSM-56F801 modula se pored procesora, nalazi i linearni stabilizator napona koji obezbeđuje 3.3V za napajanje procesora, zatim kolo ICL3232 sa DB-9 konektorom preko koga se ostvaruje serijska veza po RS232 standardu, dva tastera i dve LED diode, JTAG konektor. Osim toga, postoji i reset taster kao i LED dioda koja signalizira postojanje napajanja. Ulazno-izlazni pinovi procesora su izvedeni na 40-pinski konektor J1. Na slici 6.2. je prikazan raspored signala na konektoru J1. Na pinovima 5 i 6 su dostupni signali serijske veze, ili I/O pinovi opšte namene GPIO0 i GPIO1. Signali TD0, TD1 i TD2 pripadaju tajmersko brojačkom modulu D. Ovi pinovi mogu da se koriste i kao I/O pinovi opšte namene PA0, PA1 i PA2. Signali MOSI, MISO, SCLK i SS pripadaju SPI portu. Njihova alternativna funkcija je kao I/O pinovi opšte namene PB4, PB5, PB6 i PB7. Na konektoru nalaze i analogni ulazi A/D konvertora i svih šest PWM izlaza kao i FAULT0 ulaz PWM modula. Poslednja dva pina su vezana na RESET pin i IRQA pin procesora. CSM-56F801 modul se povezuje sa PC računarom preko JTAG adaptera. Šema JTAG adaptera je data u prilogu A. CSM-56F801 modul poseduje 14-pinski JTAG konektor koji se odgovarajućim *flat* kablom povezuje sa JTAG adapterom. JTAG adapter se povezuje na paralelni port PC računara. Detaljno uputstvo za CSM-56F801 modul se nalazi na CD-u, u direktorijumu literatura.

V_X	1	2	IREQA*
GND	3	4	RESET*
TxD0	5	6	
RxD0	7	8	
	9	10	
TD0	11	12	
TD1	13	14	
TD2	15	16	
MOSI	17	18	ANA0
MISO	19	20	ANA1
SCLK	21	22	
SS*	23	24	FAULTA0
	25	26	
	27	28	
ANA2	29	30	PWMA0
ANA3	31	32	PWMA1
ANA4	33	34	PWMA2
ANA5	35	36	PWMA3
ANA6	37	38	PWMA4
ANA7	39	40	PWMA5

Slika 6.2. Konektor J1

Da bi mogli da se isprobaju primeri opisani u narednim poglavljima, potrebno je povezati LED diodu na pin 11 J1 konektora (TD0/PA0) kao i klizač potenciometra na pin 18 (ANA0). Za testiranje primera 4, potrebno je vezati RC filtar na neki od PWM pinova, na primer PWMA5. Na izlazu iz filtra, moguće je osciloskopom posmatrati izgled signala generisanog *Space-Vector* algoritmom. U prilogu B je data šema veza potenciometra, LED diode i RC filtra sa CSM-56F801 modulom.

7. Programiranje procesora DSP56F801

7.1. Primer 1 - Test_LED

U ovom primeru je opisan program koji pali i gasi LED diodu na pinu PA0. Ovaj primer se nalazi u direktorijumu *Test_led,* koji se nalazi u direktorijumu *Examples.* Nakon pokretanja *Code Warrior* razvojnog okruženja opcijama *File→Open...* otvoriti okvir za dijalog i odabrati fajl *Test_led.mcp*. Projekat je otvoren i kod može da se modifikuje, kompajlira i da se putem JTAG interfejsa upiše u *FLASH* memoriju i testira njegovo izvršavanje.

Pin PA0 je dodeljen tajmeru D0 ali se u ovom primeru ne koristi mogućnost da tajmer D0 preko svog *OFLAG* signala upravlja ovim pinom već se pin kontroliše softverski. U ovom primeru se koristi tajmer D0 koji radi u osnovnom *count* modu. Tajmer generiše *Timer Overflow* zahtev za prekid a signal na LED diodi se menja u prekidnoj rutini. Program je razvijen kao *DSP56800x EABI Stationery* projekat, što znači da pored delova koje piše programer postoji i *start-up* kod koji je kreiran od strane razvojnog okruženja kao što je to opisano u poglavlju 4.2. Prva funkcija koja se izvršava je reset rutina *init_M56801_()* koja pripada *start-up* kodu. Ova rutina poziva funkciju *main()* tj. glavni program. Definicija funkcije *main()* se nalazi unutar fajla *main.c.* Glavni program na početku poziva rutinu za inicijalizaciju i nakon toga ulazi u beskonačnu petlju.

main.c :

```
#include "DSP56F80x_device.h"
#include "init.h"

void main(void)
{
        init_reg();         // rutina za inicijalizaciju
        while (1)
                {
                }
}
```

Na početku se nalaze dve #include direktive. Prvom direktivom se omogućava upotreba *hardvare abstraction layer-a* u kodu unutar ovog fajla. Fajl *init.h* sadrži samo prototip funkcije *init_reg()*. Funkcija *init_reg()* inicijalizuje I/O registre tajmera D i ulazno-izlaznog porta GPIOA.

init.c

```
#include "DSP56F80x_device.h"
#include "init.h"

void init_reg()
{
        GPIOA.PER.bit.PER0=0;           // pin 0 porta A se dodeljuju GPIOA modulu
        GPIOA.DDR.bit.DDR0=1;          // pin PA0 je izlazni
        TMRD0.CTRL.bit.PCS=0b1111;// primary clock source = IPbus/128
```

```
        TMRD0.CTRL.bit.CM=0b001;    // Count mode
        TMRD0.SCR.bit.TOFIE=1;      // omogucavanje timer overflow prekida
        ITCN.GRP7.bit.PLR30=1;      // prekid se preusmerava na CH0 IPR registra
        ENABLE_CH0;                 // omogucavanje prekida na kanalu CH0
}
```

Funkcija *init_reg()* koristi *hardware abstraction layer* za pristup I/O registrima. LED dioda je vezana na pin PA0, a pošto ovaj pin može da bude i TD0 linija tajmera D0, potrebno je ovaj pin dodeliti portu opšte namene GPIOA. Upravo to obavlja prva programska linija funkcije *init_reg()*. Naredbom GPIOA.DDR.bit.DDR0=1 se definiše smer I/O pina PA0 kao izlazni. Naredba TMRD0.CTRL.bit.PCS=0b1111 setuje bitove kojima se bira primarni izvor takta brojača tako da to bude preskaler sa faktorm deljenja 1/128. To znači da ako je frekvencija IPbus-a 30MHz, takt brojača iznosi 234.375KHz. Naredna linija sadrži naredbu kojom se bira *count* mod rada brojača. To znači da će brojač brojati do 65535 (0xFFFF) nakon čega se setuje *timer overflow* fleg (TOF) i generše zahtev za prekid a brojač se resetuje na nulu. To znači da brojač generiše periodu od 65536/234375=0.28 sekundi. Frekvencija paljenja i gašenja diode je 2*0.28=0.56 sekundi. Narednom linijom, TMRD0.SCR.bit.TOFIE=1, se omogućavaju *Timer Overflow* prekidi tajmera-brojača D kanal 0. Ovom zahtevu za prekid je potrebno dodeliti prioritet prosleđivanjem na odgovarajući kanal kontrolera prekida, a zatim treba omogućiti prekide na tom kanalu. To obavljaju poslednje dve naredbe funkcije *init_reg().* Pošto je *Timer Overflow* prekid tajmera brojača D kanala 0 ima broj 60 u tabeli vektora prekidnih rutina, to znači da je ovom zahtevu za prekid dodeljeno polje PLR30 registra GRP7. U ovo polje se upisuje broj kanala na koji se prekid prosleđuije uvećan za jedan. Dakle, ako se prekid prosleđuje na kanal CH0 u polje PLR30 registra GRP7 se upisuje 1. Za kanal CH2 bi se upisivao broj 3, itd.

Prekidna rutina je C funkcija čija se definicija nalazi u fajlu *isr.c*. Ime funkcije je *timerd0_ovf* i pošto ne vraća nikakav parametar kao rezultat definisana je kao tip void. U prekidnoj rutini se invertuje stanje na pinu PA0.

isr.c

```
#include "DSP56F80x_device.h"
#include "isr.h"

#pragma interrupt

void timerd0_ovf()
{
        GPIOA.DR.bit.DR0^=1;   // invertovanje stanja na pinu PA0
        TMRD0.SCR.bit.TOF=0;  // bisanje timer overflow flega
}
```

Na početku se nalaze dve #include direktive. Prvom direktivom se omogućava upotreba *hardvare abstraction layer-a* u kodu unutar ovog fajla. Fajl *isr.h* sadrži samo prototip funkcije *timerd0_ovf()*. Pre početka definicije funkcije *timerd0_ovf()* se nalazi direktiva *#pragama interrupt* kojom se prevodiocu signalizira da je naredna funkcija prekidna rutina. Telo funkcije *timerd0_ovf()*

se sastoji od samo dva reda kojima je ilustrovana upotreba *hardvare abstraction layer*-a. Linija GPIOA.DR.bit.DR0^=1 obavlja logičku operaciju ekskluzivno ili (operator ^) između bita PA0 i logičke jedinice čime se stanje bita PA0 invertuje. Linija TMRD0.SCR.bit.TOF=0 briše TOF fleg *(Timer Overflow)* registra SCR. Ova linija je neophodna jer ukoliko TOF fleg nije obrisan procesor će protumačiti da je došlo do novog zahteva za prekid pa će izvršavanje *timerd0_ovf()* rutine nastaviti da se ponavlja.

Da bi se prekidna rutina *timerd0_ovf()* pozivala prilikom timer overflow zahteva za prekid kanala 0 tajmera D, neophodno je u tabelu vektora prekidnih rutina smestiti njenu adresu. Tabela vektora prekidnih rutina se nalazi u fajlu *DSP56F801_vector.asm*. U ovom fajlu se nalazi asemblerski kod koji poziva odgovarajuće funkcije u zavisnosti od vrste zahteva za prekd koji se opslužuje. Da bi se pozvala C funkcija *timerd0_ovf()* potrebno je pronaći poziciju vektora prekida za tajmer D kanal 0 i iza jmp instrukcije upisati *Ftimerd0_ovf*.

Deo tabele vektora prekidnih rutina. Fajl DSP56F801_vector.asm

```
jmp M56801_intRoutine     ; Quad decoder #1 home sw  ($34)
jmp M56801_intRoutine     ; Quad decoder #1 idx pulse($36)
jmp M56801_intRoutine     ; Quad decoder #0 home sw  ($38)
jmp M56801_intRoutine     ; Quad decoder #0 idx pulse($3A)
jmp Ftimerd0_ovf          ; Timer D Channel 0        ($3C)
jmp M56801_intRoutine     ; Timer D Channel 1        ($3E)
jmp M56801_intRoutine     ; Timer D Channel 2        ($40)
jmp M56801_intRoutine     ; Timer D Channel 3        ($42)
jmp M56801_intRoutine     ; Timer C Channel 0        ($44)
jmp M56801_intRoutine     ; Timer C Channel 1        ($46)
jmp M56801_intRoutine     ; Timer C Channel 2        ($48)
jmp M56801_intRoutine     ; Timer C Channel 3        ($4a)
jmp M56801_intRoutine     ; Timer B Channel 0        ($4c)
jmp M56801_intRoutine     ; Timer B Channel 1        ($4e)
jmp M56801_intRoutine     ; Timer B Channel 2        ($50)
jmp M56801_intRoutine     ; Timer B Channel 3        ($52)
jmp M56801_intRoutine     ; Timer A Channel 0        ($54)
jmp M56801_intRoutine     ; Timer A Channel 1        ($56)
jmp M56801_intRoutine     ; Timer A Channel 2        ($58)
jmp M56801_intRoutine     ; Timer A Channel 3        ($5a)
jmp M56801_intRoutine     ; SCI #1 Transmit complete ($5c)
jmp M56801_intRoutine     ; SCI #1 transmitter ready ($5e)
jmp M56801_intRoutine     ; SCI #1 receiver error    ($60)
jmp M56801_intRoutine     ; SCI #1 receiver full     ($62)
```

Ovim je za adresu bezuslovnog skoka određena adresa od koje počinje *timerd0_ovf()* funkcija. Slovo F se dodaje na početku imena funkcije ako je funkcija napisana u jeziku C kao što je to ovde slučaj.

7.2. Primer 2, Test_ADC

Ovaj primer demonstrira upotrebu A/D konvertora kao i tajmera D0 u *Fixed-frequency PWM* modu rada. Kao i u prethodnom primeru, potrebno je vezati LED diodu na PA0 pin. Pored LED diode koristi se i potenciometar sa klizačem vezanim na ANA0 ulaz procesora. A/D konvertor meri signal na pinu ANA0 i na osnovu njega upravlja vremenom uključenosti LED diode na pinu PA0. Kao rezultat, intenzitet svetlosti diode može da se reguliše potenciometrom. Program se nalazi u direktorijumu *Test_ADC.* Kao u prethodnom primeru i ovde je program podeljen na celine. Program poseduje dve prekidne rutine, jednu za tajmer D0 a drugu za prekide koje generiše A/D konvertor po završenoj konverziji. Glavni program je identičan kao i u primeru *Test_LED* dok se razlukuje rutina za inicijalizaciju. *Fixed-frequency PWM* mod je opisan u poglavlju 12.

Intenzitet svetlosti LED diode je direktno srazmeran vremenu uključenosti *Ton*. Vreme uključenosti zavisi od sadržaja CMP1 registra i ono je srazmerno naponu na analognom ulazu ANA0.

Funkcija *init_regs()* obavlja inicijalizaciju I/O registara tajmera D0, A/D konvertora, GPIOA modula kao i kontrolera prekida ITCN.

Glavni program

Kao i u prethodnom primeru, i ovde glavni program nakon poziva rutine za inicijalizaciju *init_regs()* ulazi u beskonačnu petlju. Procesor izlazi iz beskonačne petlje pri opsluživanju zahteva za prekid koje generišu tajmer D0, pri završetku brojačkog ciklusa, i ADCA modul pri završenoj A/D konverziji. Na kraju se ponovo vraća u beskonačnu petlju.

main.c:

```
#include "DSP56F80x_device.h"
#include "init.h"
#include "isr.h"

unsigned int compare_value;

void main(void)
{
        init_regs();  // inicijalizacija I/O registara
        while(1)
        {
        }
}
```

Promenljiva *compare_value* je globalna promenljiva koja sadrži vrednost koju treba upisati u CMP1 registar tajmera D0 pri narednom pozivu *timerd0_ovf()* prekidne rutine.

Inicijalizacija tajmera D0

Da bi brojač radio po prethodno opisanim zahtevima potrebno je upisati odgovarajuće vrednosti u odgovarajuće I/O registre tajmera D0. Ovde će biti opisani samo oni koraci koji su neophodni za ovaj primer.

CTRL registar :

TMR_BASE+$6, E,16, or 1E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read / Write	COUNT MODE			PRIMARY COUNT SOURCE				SECONDARY SOURCE		ONCE	LENGTH	DIR	Co INIT	OUTPUT MODE		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Pošto se u ovom primeru koristi *Fixed-frequency PWM* mod rada *Count Mode* bitovi moraju biti jednaki 0 0 1 (počev od bita 15). Ovaj mod se zadaje linijom :

```
TMRD0.CTRL.bit.CM=1;
```

Dalje je ptrebno odabrati izvor takta za brojač, a to je u ovom primeru preskaler koji deli učestanost IPbus takta od 30MHz sa 4. Izvor takta se definiše bitovima *Primary Count Source.* Prema dokumentaciji procesora *DSP56F80x,* da bi se brojač taktovao internim signalom takta podeljenim sa 4 u *Primary Count Source* polje bitova treba upisati vrednost 1 0 1 0 odnosno decimalni broj 10.

```
TMRD0.CTRL.bit.PCS=10;
```

Takt brojača je 7.5MHz, što znači da trajanje periode PWM signala 65536/7.5 μs a to je približno 8.74ms. Odavde sledi da je učestanost ovako generisanog PWM signala približno 114Hz. Brojač treba da broji do maksimalne vrednosti a to je 65535 (0xFFFF) i nakon toga se brojački registar resetuje i ciklus kreće iz početka. Pošto CMP1 registar može da sadrži vrednost koja je manja od 0xFFFF izjednačavanje stanja brojačkog registra sa sadržajem CMP1 registra ne sme da ima nikakav uticaj na dalje brojanje bit *LENGTH* registra *CTRL* mora biti jednak nuli. Brojač treba da ponavlja cikluse jedan za drugim bez zaustavljanja pa bit *ONCE* registra *CTRL* mora biti jednak nuli.

```
TMRD0.CTRL.bit.LENGTH=0
TMRD0.CTRL.bit.ONCE=0;
```

Početne vrednosti ova dva bita su jednake nuli tako da se ova dva reda mogu i izostaviti. Da bi brojač mogao direktno da upravlja LED diodom na pinu PA0 potrebno je omogućiti hardversku kontrolu ovog pina od strane tajmera D0. Pre svega potrebno je u najniži bit registra PER *(Peripheral Enable Register)* modula GPIOA upisati logičku jedinicu.

```
GPIOA.PER.bit.PER0=1;
```

Dalje je neophodno omogućiti tajmeru D0 hardversku kontrolu ovog pina upisivanjem odgovarajuće vrednosti u *Output Mode* bitove registra *CTRL.* Upisom bitova 1 1 0 tj. decimalno 6 u ovo polje dobija se mod rada u kojem se na pinu PA0 pojavljuje logička jedinica pri dostizanju

vrednosti u CMP1 registru. Pin PA0 se vraća na logičku nulu pri resetu brojača. Ovaj mod rada se zadaje naredbom:

```
TMRD0.CTRL.bit.OM=6;
```

Na kraju je potrebno dati dozvolu da tajmer D0 kontroliše PA0 pin upisom logičke jedinice u bit *OEN* registra *SCR.* Da bi brojač mogao da poziva prekidnu rutinu pri svakom dostizanju svoje maksimalne vrednosti brojanja potrebno je omogućiti *timer overflow* zahtev za prekid setovanjem bita *TOFIE* registra *SCR.*

```
TMRD0.SCR.bit.OEN=1;
TMRD0.SCR.bit.TOFIE=1;
```

SCR registar:

TMR_BASE+$7, F,17, or 1F	15	14	13	12	11	10	9	8	[7:6]	5	4	3	2	1	0
Read	TCF	TCFIE	TOF	TOFIE	IEF	IEFIE	IPS	INPUT	Capture Mode	MSTR	EEOF	VAL	0	OPS	OEN
Write													FORCE		
Reset	0	0	0	0	0	0	0	0	0 0	0	0	0	0	0	0

Bitovi *OEN* i *TOFIE* su jedina dva bita registra *SCR* koja je potrebno setovati tokom inicijalizacije. Zahtev za prekid *timer overflow* se detektuje kad se setuje *TOF* bit. Ovaj bit treba da se resetuje pre izlaska iz prekidne rutine upisom logičke jedinice.

```
TMRD0.SCR.bit.TOF=0;
```

Inicijalizacija A/D konvertora

A/D konvertor treba da uzima odbirke samo sa jednog analognog ulaza, ANA0. Prekidna rutina koja *timerd0_ovf()* startuje novu A/D konverziju pri svakom izvršavanju. Kad se A/D konverzija signala sa ANA0 završi, završava se i *scan* ciklus. Ove zahteve ispunjava *Once sequential* mod rada ADCA modula. Da bi se ciklus zaustavio nakon konverzije signala ANA0 potrebno je onemogućiti konverziju signala sa ulaza ANA1. Ovo se postiže setovanjem *Disable Sample* bita DS1 registra ADSDIS.

ADCR1 registar:

ADC_BASE+$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	STOP	0	SYNC	EOSIE	ZCIE	LLMTIE	HLMTIE	CHNCFG				0	SMODE		
Write			START													
Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1

Da bi ADCA modul radio u *Once sequential* modu rada, bitovi *SMODE* trebaju da budu jednaki nuli. Da bi se omogućio start A/D konverzije bit STOP mora biti resetovan. Bit SYNC određuje da li A/D konverzija započinje pojavom signala na internom *sync* ulazu ADCA modula. Ulaz *sync* je vezan za OFLAG izlaz tajmera C2. Ulazni pin ovog tajmera je izlazni *sync* signal PWMA modula. Zahvaljujući ovim vezama omogućena je jednostavna sinhronizacija A/D konvertora sa PWM modulom. Pošto se u ovom primeru nova konverzij započinje setovanjem START bita, bit SYNC treba resetovati.

ADCA.ADCR1.all=0;

Ovom linijom je obrisan sadržaj ADCR1 registra. Time je zadat *Once sequential* mod rada, a bitovi SYNC i STOP su obrisani. Nakon završetka procesa A/D konverzije ADCA modul generiše *end of scan* zahtev za prekid. Ovaj prekid se omogućava setovanjem EOSIE bita.

ADCA.ADCR1.bit.EOSIE=1;

Dalje je potrebno odabrati učestanost takta za ADCA modul. Takt za ADCA modul se dobija deljenjem učestanosti IPus takta odgovarajućim faktorom u zavisnosti od sadržaja DIV bitova ADCR2 registra.

ADCR2 registar:

ADC_BASE+$1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	0	DIV			
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Učestanost takta ADCA modula se računa po formuli :

$$f_{ADC} = \frac{f_{IPbus}}{2 \cdot N} \quad ; \quad \text{gde je} \quad N = DIV + 1$$

Ako želimo da učestanost takta bude 2.5MHz, tada je u polje DIV potrebno upisati broj 5 (1 0 1).

ADCA.ADCR2.bit.DIV=5;

Da bi se ciklus zaustavio nakon konverzije signala ANA0 potrebno je onemogućiti konverziju signala sa ulaza ANA1. Ovo se postiže setovanjem *Disable Sample* bita DS1 registra ADSDIS.

ADCA.ADSDIS.bit.DS1=1;

ADSDIS registar:

ADC_BASE+$5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	TEST		0	0	0	0	0	0	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ovim registrom se kontroliše *scan* ciklus ADCA modula. Nakon što se setuje START bit, A/D konverzije se izvršavaju sekvencijalno, uzimajući odbirke sa analognih ulaza onim redosledom kako je to definisano stanjem *ADC Channel List* registara ADLST1 i ADLST2. Ciklus se završava dok se ne naiđe na onaj analogni ulaz čije je semplovanje onemogućeno upisom logičke jedinice na odgovarajuću poziciju među DS bitovima ADSDIS registara. Tako na primer, ako je potrebno uzimati odbirke sa prva dva analogna ulaza (ANA0 i ANA1) treba setovati bit DS2.

Polje TEST ima ulogu u tzv. test modu ADCA modula kad se upisom u ovo polje 01 dovodi napon Vref/2 na analogne ulaze ANA0 i ANA4.

Pošto je početno stanje ADLST1 i ADLST2 registara takvo da je redosled odabiranja po kanalima od ANA0 do ANA7 rastućim, u ovom primeru nije bilo potrebe za modifikaciju sadržaja ovih registara.

Inicijalizacija kontrolera prekida ITCN

U ovom primeru se koriste dve prekidne rutine, jedna za tajmer D0 i druga za ADCA modul *(ADC A Conversion complete)*. Zahtev za prekid tajmera D0 se preusmerava na kanal CH0, dok se zahtev za prekid ADCA modula dovodi na kanal CH1 kontrolera prekida ITCN.

```
ITCN.GRP7.bit.PLR30=1;
ITCN.GRP13.bit.PLR55=2;
```

Registar GRP sadrži polje PLR30 kojim se definiše na koji kanal se zahtev za prekid tajmera D0 dovodi. Upisom broja 1 (setovanjem najnižeg bita ovog polja), ovaj zahtev za prekid se dovodi na kanal CH0. Registar GRP13 sadrži polje PLR55 kojim se definiše na koji kanal se dovodi zahtev za prekid *ADC A Conversion complete*. Upisom broja 2 u ovo polje ovaj zahtev za prekid se dovodi na kanal CH1. Na kraju je potrebno omogućiti prekide na ova dva kanala.

```
ENABLE_CH1;
ENABLE_CH0;
```

Init.c:

```
#include "DSP56F80x_device.h"
#include "init.h"

void init_regs()
        {
                ADCA.ADCR1.all=0;               // brisanje ADCR1 registra
                ADCA.ADCR1.bit.EOSIE=1;         // End of scan interrut enable
                ADCA.ADCR2.bit.DIV=5;
                ADCA.ADSDIS.bit.DS1=1;          // Disable SAMPLE1
                GPIOA.PER.bit.PER0=1;           // pin PA0 se dodeljuje tajmeru D0

                TMRD0.CTRL.bit.PCS=10;          // primary clock source = IPbus/128
                TMRD0.CTRL.bit.CM=1;            // Count mode
                TMRD0.CTRL.bit.LENGTH=0
                TMRD0.CTRL.bit.ONCE=0;          // ciklusi brojanja se ponavljaju
                TMRD0.CTRL.bit.OM=6;            // Output mode
                TMRD0.SCR.bit.OEN=1;            //
                TMRD0.SCR.bit.TOFIE=1;          // omogucavanje timer overflow prekida

                ITCN.GRP7.bit.PLR30=1;
                ITCN.GRP13.bit.PLR55=2;
                ENABLE_CH1;                     // dozvola prekida na kanalu CH1 (ADCA)
                ENABLE_CH0;                     // dozvola prekida na kanalu CH0 (TMRD0)
        }
```

Prekidne rutine

Tajmer D0 poziva prekidnu rutinu *timerd0_ovf()*. U ovoj rutini se setuje bit START registra, ADCR1 čime se započinje A/D konverzija, a zatim se briše TOF fleg. Po završetku A/D konverzije ADCA modul setuje EOSI bit statusnog registra ADSTAT što generiše zahtev za

prekid na kanalu CH1 kontrolera prekida ITCN. Poziva se rutina *adca_conversion_complete()* u kojoj se čita rezultat konverzije koji se nalazi u registru ADRSLT0 *(ADC result register 0)*. Rezultat konverzije se dodeljuje promenljivoj *compare_value.* Promenljiva *compare_value* se množi sa dva (pomeranje za jedan bit u desno). Na kraju se resetuje EOSI bit. Za EOSI bit je karaktristično da se briše upisom logičke jedinice. Obe prekidne rutine se nalaze u fajlu *isr.c*.

isr.c:

```
#include "DSP56F80x_device.h"
#include "isr.h"

extern unsigned int compare_value;

// Prekidna rutina tajmera D0
#pragma interrupt
void timerd0_ovf()
        {
                ADCA.ADCR1.bit.START=1;          // start A/D konverzije
                TMRD0.SCR.bit.TOF=0;             // brisanje timer overflow flega
        }


// Prekidna rutina A/D konvertora
#pragma interrupt
void adca_conversion_complete()
        {
                compare_value=ADCA.ADRSLT0;      // citanje rezultata konverzije
                compare_value<<=1;               // mnozenje rezultata sa 2
                TMRD0.CMP1=compare_value;        // rezultat se smesta u CMP1
                ADCA.ADSTAT.bit.EOSI=1;          // brisanje End of Scan interrupt flega
        }
```

Da bi procesor mogao da poziva prekidne rutine, potrebno je njihove adrese smestiti u tabelu vektora prekidnih rutina, koja se nalazi u fajlu *DSP56F801_vector.asm.*

```
jmp M56801_intRoutine     ; Quad decoder #1 idx pulse($36)
jmp M56801_intRoutine     ; Quad decoder #0 home sw  ($38)
jmp M56801_intRoutine     ; Quad decoder #0 idx pulse($3A)
jmp Ftimerd0_ovf          ; Timer D Channel 0        ($3C)
jmp M56801_intRoutine     ; Timer D Channel 1        ($3E)
jmp M56801_intRoutine     ; Timer D Channel 2        ($40)
jmp M56801_intRoutine     ; Timer D Channel 3        ($42)
jmp M56801_intRoutine     ; Timer C Channel 0        ($44)
jmp M56801_intRoutine     ; Timer C Channel 1        ($46)
jmp M56801_intRoutine     ; Timer C Channel 2        ($48)
jmp M56801_intRoutine     ; Timer C Channel 3        ($4a)
jmp M56801_intRoutine     ; Timer B Channel 0        ($4c)
jmp M56801_intRoutine     ; Timer B Channel 1        ($4e)
jmp M56801_intRoutine     ; Timer B Channel 2        ($50)
jmp M56801_intRoutine     ; Timer B Channel 3        ($52)
jmp M56801_intRoutine     ; Timer A Channel 0        ($54)
jmp M56801_intRoutine     ; Timer A Channel 1        ($56)
jmp M56801_intRoutine     ; Timer A Channel 2        ($58)
jmp M56801_intRoutine     ; Timer A Channel 3        ($5a)
jmp M56801_intRoutine     ; SCI #1 Transmit complete ($5c)
jmp M56801_intRoutine     ; SCI #1 transmitter ready ($5e)
jmp M56801_intRoutine     ; SCI #1 receiver error    ($60)
jmp M56801_intRoutine     ; SCI #1 receiver full     ($62)
jmp M56801_intRoutine     ; SCI #0 Transmit complete ($64)
jmp M56801_intRoutine     ; SCI #0 transmitter ready ($66)
jmp M56801_intRoutine     ; SCI #0 receiver error    ($68)
jmp M56801_intRoutine     ; SCI #0 receiver full     ($6a)
jmp M56801_intRoutine     ; ADC B Conversion complete($6c)
jmp Fadca_conversion_complete   ; ADC A Conversion complete($6e)
jmp M56801_intRoutine     ; ADC B zero crossing/error($70)
jmp M56801_intRoutine     ; ADC A zero crossing/error($72)
jmp M56801_intRoutine     ; Reload PWM B             ($74)
jmp M56801_intRoutine     ; Reload PWM A             ($76)
```

Vektor prekida tajmera D0 se nalazi na adresi 3Ch a za ADCA *conversion complete* prekid na adresi 6Eh. Slovo F ispred imena rutine označava da su rutine napisane u jeziku C.

Povezivanjem *CSM-56F801* modula na JTAG adapter i upisivanjem koda u FLASH memoriju procesora komandama *Project→Debug,* ili pritiskom na taster F5 ulazi se u *Debug* mod. Ako se pokrene izvršavanje programa komandama *Project→Run,* okretanjem potenciometra se reguliše intenzitet svetlosti LED diode. Ako se sada izvršavanje programa zaustavi klikom na dugme *Break*, može se primetiti da LED dioda svetli onim intenzitetom kojim je svetlela pre zaustavljanja programa, nezavisno od okretanja potenciometra. To znači da tajmer D0 i dalje radi i generiše PWM signal na pinu PA0, iako je izvršavanje programa zaustavljeno. Dakle, pri debagovanju nekog programa treba voditi računa da iako je program zaustavljen periferijske jedinice i dalje rade, a samim tim mogu da generišu zahteve za prekid. Ako se izvršavanje programa zaustavi unutar neke prekidne rutine, može se desiti da se pri izvršavanju programa korak po korak uvek iznova ulazi u istu prekidnu rutinu.

7.3. Primer 3, RS232_Test

Ovaj primer se bavi realizacijom jednostavne komunikacije između procesora *DSP56F801* i PC računara putem serijske veze. Procesor prima znak serijskom vezom korišćenjem SCI0 modula. Primljeni znak se inkrementira a zatim se šalje nazad. Slanjem bilo kog znaka, kao rezultat se dobija naredni znak iz ASCII tabele. Glavni program, nakon inicijalizacije I/O registara, ulazi u beskonačnu petlju dok sav posao obavljaju prekidne rutine. I u ovom primeru se koristi LED dioda koja je vezana na pin PA0. Pri svakom slanju podatka, stanje na ovoj diodi se invertuje. Program se nalazi u direktorijumu *RS232_Test,* i kao i prethodni primeri podeljen je u vise celina. Program poseduje dve prekidne rutine koje poziva SCI0 modul. Rutina *Receiver_Full()* se poziva po prijemu znaka putem serijske veze, dok se rutina *Transmitter_Ready()* poziva nakon što SCI0 modul prebaci sadržaj SCIDR registra u izlazni šift registar. Program se testira povezivanjem PC računara sa CSM-56F801 modulom serijskim kablom. Na PC računaru treba pokrenuti neki terminalski program koji omogućava konfigurisanje serijske vezer izborom formata i brzine prenosa. Na CD-u se nalazi program *terminal.exe* koji može da posluži za ovu svrhu.

Inicijalizacija

Serijska komunikacija se odvija brzinom od 19200bps, sa 8 bitova podataka jednim stop bitom i bez bita parnosti. Pre nego što se inicijalizuju registri SCI0 modula potrebno je fino podesiti interni relaksacioni oscilator procesora. Naime, procesor *DSP56F801* poseduje interni oscilator čija je učsetanost tokom rada stabilna, ali se ova učestanost razlikuje od procesora do procesora. U ovom primeru je neophodno poznavati tačnu učestanost rada ovog oscilatora, jer se na osnovu učestanosti IPbus takta (koja se dobija iz PLL kola) izračunava vrednost koju je potrebno upisati u SCIBR (SCI Baud Rate) registar. Da bi učestanost takta procesora bila 60MHz, pri čemu se dobija učestanost IPbus takta od 30MHz, potrebno je fino podesiti učestanost relaksacionog oscilatora upisom odgovarajuće vrednosti u polje TRIM registra IOSCTL, koji pripada OCCS modulu *(On-Chip Clock Synthesis).* Za procesor koji se nalazi na CSM-56F801 modulu na kome su testirani ovi primeri, utvrđeno je da je učestanost oscilatora potrebno povećati za 8.74% da bi se dobila učestanost IPbus takta od 30MHz. Zbog toga je potrebno smanjiti inicijalnu vrednost TRIM polja sa 128 na 90 jer svako smanjenje za 1 povećava učestanost oscilatora za 0.23%. Na kraju je potrebno upisati u registar IOSCTL vrednost 90(decimalno) ili 0x5A (heksadecimalno). Nakon što je učestanost takta podešena, pin PA0 se konfiguriše kao izlazni na isti način kao i u prethodnim primerima.

Dalje se vrši inicijalizacija SCI0 modula. Prvo se podešava brzina prenosa podataka upisom odgovarajuće vrednosti u SCIBR registar. Ova vrednost se računa po formuli $SCIBR = \frac{IPbus\_clk}{Baud\_Rate \cdot 16}$. Pošto je učestanost IPbus takta 30MHz, za Baud_Rate od 19200bps se dobija vrednost 97 koju je potrebno upisati u registar SCIBR:

```
SCI0.SCIBR=97;                  // baud rate = 19200bps
```

Da bi SCI0 modul po prijemu novog znaka pozvao prekidnu rutinu, potrebno je setovati bit RFIE registra SCICR. Nakon toga se omogućava rad prijemnika i predajnika SCI0 modula setovanjem bitova RE i TE registra SCICR. Na kraju je potrebno, slično kao u prethodnim primerima, usmeriti zahteve za prekid na odgovarajuće kanale kontrolera prekida ITCN.

```
ITCN.GRP13.bit.PLR53=1;         // SCI0 Receiver Full prekid na kanalu 0
ITCN.GRP12.bit.PLR51=2;         // SCI0 Transmitter Ready prekid na kanalu 1
```

Na kraju je potrebno dozvoliti prekide na kanalima 0 i 1.

```
#include "DSP56F80x_device.h"
#include "init.h"

void init_regs(void)
        {
        OCCS.IOSCTL=90;             // IPbus_clock(new)=IPbus_clock(old)*1.0874

        GPIOA.PER.bit.PER0=0;       // pin 0 porta A se dodeljuje GPIOA modulu
        GPIOA.DDR.bit.DDR0=1;       // pin PA0 je izlazni

        SCI0.SCIBR=97;              // baud rate = 19200bps
        SCI0.SCICR.bit.RFIE=1;      // receiver full interrupt enable
        SCI0.SCICR.bit.RE=1;        // receiver enable
        SCI0.SCICR.bit.TE=1;        // transmitter enable

        ITCN.GRP13.bit.PLR53=1;     // SCI0 Receiver Full prekid na kanalu 0
        ITCN.GRP12.bit.PLR51=2;     // SCI0 Transmitter Ready prekid na kanalu 1

        ENABLE_CH0;                 // Omogucavanje prekida na kanalu 1
        ENABLE_CH1;                 // Omogucavanje prekida na kanalu 1
        }
```

Prekidne rutine

Prekidna rutina *Receiver_Full()* se poziva nakon setovanja RDRF flega registra SCISR. U ovoj rutini se prvo resetuje RDRF fleg čitanjem statusnog registra SCISR i DATA registra SCIDR. Nakon toga se setuje bit TEIE registra SCICR čime se omogućuje *SCI0 Transmitter Ready* zahtev za prekid.

```
void Receiver_Full()
{
        unsigned int status;
        status=SCI0.SCISR.all;  // Citanje statusnog registra
```

```
        temp=SCI0.SCIDR;       // Citanje pristiglog podatka iz data registra
        SCI0.SCICR.bit.TEIE=1;// Dozvola SCI0 Transmitter Empty prekida
}
```

Nakon povratka iz *Receiver_Full()* prekidne rutine, pošto je omogućen *SCI0 Transmitter Ready* zahtev za prekid, poziva se prekidna rutina *Transmitter_Ready().* U ovoj rutini se prvo inkrementira vrednost znaka prethodno pročitanog u prekidnoj rutini *Receiver_Full()*. Nakon toga se briše TDRE fleg registra SCISR čitanjem tog registra i upisom vrednosti za slanje u SCIDR Data registar. Na kraju se onemogućava ponovni poziv ove prekidne rutine resetovanjem TEIE bita SCICR registra.

```
void Transmitter_Ready()
{
        unsigned int status;
        temp++;
        status=SCI0.SCISR.all;          // Citanje statusnog registra
        SCI0.SCIDR=temp;                // Upis podatka za slanje u data registar
        SCI0.SCICR.bit.TEIE=0;          // Onemogucavanje SCI0 Transmitter Empty prekida
        GPIOA.DR.bit.DR0^=1;            // invertovanje stanja na pinu PA0
}
```

Glavni program

Kao što je već opisano, glavni program poziva rutinu za inicijalizaciju nakon čega ulazi u beskonačnu petlju. Iz beskonačne petlje procesor iskače samo pri servisiranju prekidnih rutina.

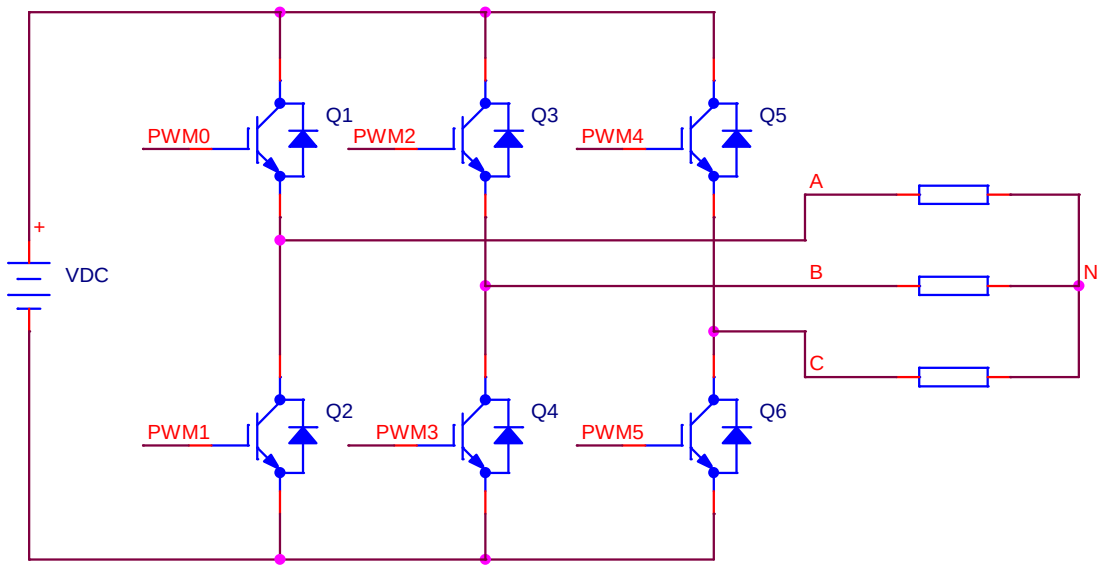
```
void main(void)
{
        init_regs();        // inicijalizacija SCI0 modula i ITCN kontrolera prekida

        while (1)
                {
                }
}
```

Kao i u prethodnim primerima, potrebno je modifikovati tabelu vektora prekidnih rutina tako da se omogući pozivanje prekidnih rutina *Receiver_Full()* i *Transmitter_Ready()* upisom njihovih adresa na odgovarajuće pozicije u tabeli vektora prekidnih rutina *DSP56F801_vector.asm.*

```
jmp M56801_intRoutine     ; SCI #1 Transmit complete ($5c)
jmp M56801_intRoutine     ; SCI #1 transmitter ready ($5e)
jmp M56801_intRoutine     ; SCI #1 receiver error    ($60)
jmp M56801_intRoutine     ; SCI #1 receiver full     ($62)
jmp M56801_intRoutine     ; SCI #0 Transmit complete ($64)
jmp FTransmitter_Ready    ; SCI #0 transmitter ready($66)
jmp M56801_intRoutine     ; SCI #0 receiver error    ($68)
jmp FReceiver_Full        ; SCI #0 receiver full     ($6a)
jmp M56801_intRoutine     ; ADC B Conversion complete($6c)
jmp M56801_intRoutine     ; ADC A Conversion complete($6e)
jmp M56801_intRoutine     ; ADC B zero crossing/error($70)
jmp M56801_intRoutine     ; ADC A zero crossing/error($72)
```

7.4. Primer 4, Space-Vector PWM

U ovom primeru je prikazan jedan način implementacije *Space-Vector PWM* algoritma. *Space-Vector PWM* (u daljem tekstu SVPWM) je jedna od najčešće korišćenih tehnika impulsno-širinske modulacije (PWM) u upravljanju invertorima koji napajaju elektromotorne pogone naizmenične struje. Kod ovakvih pogona se zahteva da invertor može da generiše napon promenljive amplitude i učestanosti. SVPWM tehnika omogućava maksimalno iskorišćenje napona jednosmernog međukola, kojim se napaja invertor, kao i minimizaciju prekidačkih gubitaka u invertoru.



Slika 7.1. Trofazni invertor

Na slici 7.1 je prikazan trofazni tranzistorski invertor, na koji je priključeno simetrično opterećenje. Sa Vdc je označen izvor jednosmernog napajanja. Opterećenje je vezano u zvezdu a neutralna tačka (zvezdište) je označena slovom N. Kao što se vidi sa slike, zvezdište je izolovano, što je najčešći slučaj pri napajanju asinhronog motora. Tranzistori Q1 do Q6 rade u prekidačkom režimu rada. Pošto tranzistor može da bude ili uključen ili isključen, a budući da u svakoj grani samo jedan tranzistor može da bude uključen, za predstavljanje stanja invertora mogu se koristiti vektori od tri bita koji reprezentuju prekidačko stanje grupe tranzistora Q1, Q3 i Q5. Ukupan broj vektora je osam. Stanja tranzistora Q2, Q4 i Q6 su invertovana u odnosu na stanja tranzistora Q1, Q3 i Q5. Analizom rada invertora mogu se jednostavno izračunati fazni

naponi V_{AN}, V_{BN} i V_{CN} pri svim mogućim prekidačkim stanjima invertora. Ovi naponi su dati u tabeli 7.1.

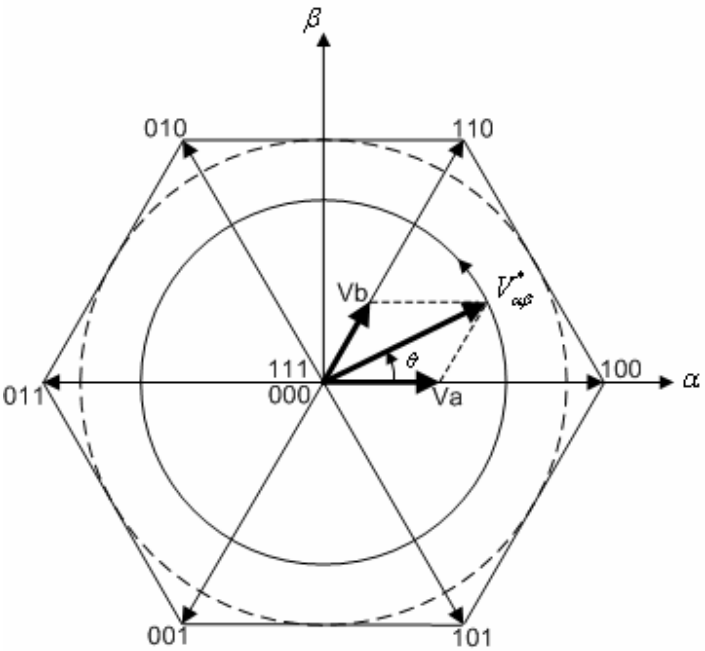
	Q1	Q3	Q5	V_{AN}	V_{BN}	V_{CN}	$V_{\alpha n}$	$V_{\beta n}$
V0	0	0	0	0	0	0	0	0
V1	1	0	0	$\frac{2}{3}V_{DC}$	$-\frac{1}{3}V_{DC}$	$-\frac{1}{3}V_{DC}$	$\frac{2}{3}V_{DC}$	0
V2	1	1	0	$\frac{1}{3}V_{DC}$	$\frac{1}{3}V_{DC}$	$-\frac{2}{3}V_{DC}$	$\frac{1}{3}V_{DC}$	$\frac{\sqrt{3}}{3}V_{DC}$
V3	0	1	0	$-\frac{1}{3}V_{DC}$	$\frac{2}{3}V_{DC}$	$-\frac{1}{3}V_{DC}$	$-\frac{1}{3}V_{DC}$	$\frac{\sqrt{3}}{3}V_{DC}$
V4	0	1	1	$-\frac{2}{3}V_{DC}$	$\frac{1}{3}V_{DC}$	$\frac{1}{3}V_{DC}$	$-\frac{2}{3}V_{DC}$	0
V5	0	0	1	$-\frac{1}{3}V_{DC}$	$-\frac{1}{3}V_{DC}$	$\frac{2}{3}V_{DC}$	$-\frac{1}{3}V_{DC}$	$-\frac{\sqrt{3}}{3}V_{DC}$
V6	1	0	1	$\frac{1}{3}V_{DC}$	$-\frac{2}{3}V_{DC}$	$\frac{1}{3}V_{DC}$	$\frac{1}{3}V_{DC}$	$-\frac{\sqrt{3}}{3}V_{DC}$
V7	1	1	1	0	0	0	0	0

Tabela 7.1.

Vektori 000 i 111 se zovu nulti vektori. Ovi vektori ne proizvode naponsku razliku na krajevima invertora. Ostalih šest vektora se nazivaju aktivni vektori. Da bi invertor generisao trofazni sistem napona na svojim krajevima potrebno je upravljati tranzistorima primenom aktivnih i nultih vektora odgovarajućim redosledom i vremenima trajanja svakog vektora. Trofazni sistem napona, fazno pomerenih za $\frac{2\cdot\pi}{3}$, može se predstaviti tzv. prostornim vektorom *(Space-Vector)* koji rotira u kompleksnoj ravni. Poznavanjem ugaonog položaja i amplitude prostornog vektora mogu se odrediti trenutne vrednosti sve tri fazne veličine koje su predstavljene prostornim vektorom. Budući da prostorni vektor poseduje samo dve komponente, realnu i imaginarnu, potrebno je sve tri fazne veličine svesti na dve komponente. To se postiže Klarkovom transformacijom (7.1).

$$\begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix} = \frac{2}{3} \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} V_{AN} \\ V_{BN} \\ V_{CN} \end{bmatrix} \quad (7.1)$$

Klarkovom transformacijom je trofazni A,B,C sistem sveden na dvofazni α,β sistem. Ako se sada primeni klarkova transformacija na napone V_{AN}, V_{BN} i V_{CN} u tabeli, dobijaju se poslednje dve kolone sa naponima V_α, V_β u α,β sistemu. Koeficijent 2/3 je uveden da bi maksimalna amplitude napona u α,β sistemu bila jednaka maksimalnoj amplitudi faznog napona u ABC sistemu. Ako se uzme da je osa α realna, a osa β imaginarna osa kompleksne ravni tada se vektori mogu predstaviti u kompleksnoj ravni kao na slici 7.2.



Slika 7.2. *Space-Vector* dijagram

Aktivni vektori su iste amplitude i međusobno su ugaono pomereni za po 60°. Vektori 100 i 011 se nalaze na realnoj osi pošto im je imaginarna komponenta jednaka nuli kao što se vidi iz tabele. Nulti vektori 111 i 000 se nalaze u koordinatnom početku. Sa $V^*_{\alpha\beta}$ je označen prostorni vektor trofaznog sistema napona koji se generiše na krajevima invertora. Od amplitude i ugaone brzine ovog vektora zavisi amplituda i učestanost izlaznog napona invertora. U posmatranom trenutku ugaoni položaj vektora $V^*_{\alpha\beta}$ je označen sa θ a projekcije ovog vektora na ose vektora 100 i 110 sa V_a i V_b. Vektor $V^*_{\alpha\beta}$ se, u posmatranom trenutku, realizuje kao zbir vektora V_a i

V_b. Budući da ugao između aktivnih vektora 100 i 110 iznosi $\frac{\pi}{3}$, mogu se izračunati projekcije V_a i V_b.

$$V_{\alpha\beta}^{*} \cdot \sin\left(\frac{\pi}{3} - \theta\right) = V_a \cdot \sin\left(\frac{\pi}{3}\right) \tag{7.2}$$

$$V_{\alpha\beta}^{*} \cdot \sin(\theta) = V_b \cdot \sin\left(\frac{\pi}{3}\right) \tag{7.3}$$

Jednačina 8.2 se izvodi projekcijom vektora $V_{\alpha\beta}^{*}$ i V_a na osu koja je normalna na osu vektora 110, dok se jednačina 8.3. izvodi projekcijom vektora $V_{\alpha\beta}^{*}$ i V_b na osu β. Iz jednačina (7.2) i (7.3) sledi:

$$V_a = \frac{2}{\sqrt{3}} \cdot V_{\alpha\beta}^{*} \cdot \sin\left(\frac{\pi}{3} - \theta\right) \tag{7.4}$$

$$V_b = \frac{2}{\sqrt{3}} \cdot V_{\alpha\beta}^{*} \cdot \sin(\theta) \tag{7.5}$$

Vektor V_a se realizuje aktiviranjem vektora V1 u trajanju $\frac{t_a}{T_{PWM}}$ a vektor V_b aktiviranjem vektora V2 u trajanju $\frac{t_b}{T_{PWM}}$. Sa T_{PWM} je označena perioda PWM učestanosti. Ostatak vremena $t_0 = T_{PWM} - t_a - t_b$ je aktivan neki od nultih vektora V0 ili V7.

$$\frac{t_a}{T_{PWM}} = \frac{V_a}{\frac{2}{3} \cdot V_{DC}} \tag{7.6}$$

$$\frac{t_b}{T_{PWM}} = \frac{V_b}{\frac{2}{3} \cdot V_{DC}} \tag{7.7}$$

Iz jednačina (8.6) i (8.7) sledi:

$$t_a = T_{PWM} \cdot \sqrt{3} \cdot \frac{V_{\alpha\beta}^{*}}{V_{DC}} \cdot \sin\left(\frac{\pi}{3} - \theta\right) = T_{PWM} \cdot m \cdot \sin\left(\frac{\pi}{3} - \theta\right) \tag{7.9}$$

$$t_b = T_{PWM} \cdot \sqrt{3} \cdot \frac{V_{\alpha\beta}^{*}}{V_{DC}} \cdot \sin(\theta) = T_{PWM} \cdot m \cdot \sin(\theta) \tag{7.10}$$

$$t_0 = T_{PWM} - t_a - t_b \tag{7.11}$$

Sa m je označen indeks modulacije, odnosno odnos maksimalne amplitude generisanog faznog napona na krajevima invertora sa naponom jednosmernog napajanja V_{DC}. Maksimalna amplituda faznog napona na krajevima invertora je:

$$\left|V_{\alpha\beta}\right| = \frac{\sqrt{3}}{2} \cdot \frac{2}{3} \cdot V_{DC} = \frac{\sqrt{3}}{3} \cdot V_{DC} = \frac{V_{DC}}{\sqrt{3}} \tag{7.12}$$

Iz jednačine (7.11) sledi da je maksimalna vrednost indeksa modulacije:

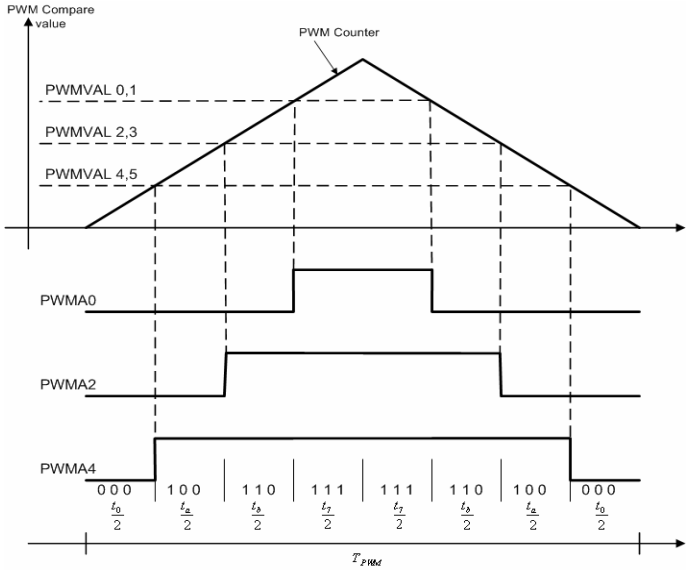
$$m^{\max} = \sqrt{3} \cdot \frac{\left|V_{\alpha\beta}\right|}{V_{DC}} = \sqrt{3} \cdot \frac{V_{DC}}{\sqrt{3}} \cdot \frac{1}{V_{DC}} = 1 \tag{7.13}$$

Kada prostorni vektor pređe u sektor II, tada se za njegovu realizaciju koriste vektori V2 (110) i V3 (010). Vremena t_a i t_b se izračunavaju po formulama (7.9) i (7.10) bez obzira na sektor u kome se vektor $V_{\alpha\beta}^*$, pri čemu je važno napomenuti da se ugao θ meri od početka u svakom sektoru.

Uzimajući u obzir prethodno izvođenje, može se napisati:

$$V_{\alpha\beta}^* = \frac{t_a \cdot V_1 + t_b \cdot V_2 + t_0 \cdot V_0}{T_{PWM}} \tag{7.14}$$

Za generisanje vektora $V_{\alpha\beta}^*$ nije od značaja redosled kojim će se tokom jednog PWM intervala uključivati aktivni vektori kao i njihova pozicija unutar intervala (tačan trenutak uključenja), već je samo bitno da njihovo trajanje bude t_a i t_b. Takođe, nije bitno ni koji se nulti vektor koristi kao i njegova tačna pozicija unutar T_{PWM} intervala. Zahvaljujući toj činjenici, postoji više SVPWM tehnika. Ovde je opisana najčešće korišćena tehnika simetrične PWM sa minimalnim brojem komutacija po periodi. Simetrična PWM se može jednostavno realizovati PWM modulom procesora *DSP56F80x*. Na slici 7.3 je prikazan izgled tri simetrična PWM signala namenjena za upravljanje trofaznim invertorom.



Slika 7.3. Simetrična PWM

Da bi se ovi signali mogli generisati potrebno je znati vremena uključenosti pojedinih tranzistora t_{Q1}, t_{Q3} i t_{Q5} tako da bude realizovan SVPWM algoritam. Do ovih vremena se može doći na osnovu vremena t_a, t_b i t_0 primenom inverzne Klarkove transformacije. Ovaj postupak je numerički zahtevan pa su zbog toga razvijene razne tehnike realizacije SVPWM. Jedna od tehnika je SVPWM bazirana na nosiocu *(Carrier-based SVPWM)* koja je i realizovana u ovom primeru.

Na slici 7.3 je prikazan izgled PWM signala za slučaj kada se prostorni vektor nalazi u sektoru 1. Koriste se vektori V1 u trajanju t_a, V2 u trajanju t_b i simetrična upotreba nultih vektora V0 i V7 u ukupnom trajanju t_0.

Q1	Q3	Q5	time
0	0	0	$\frac{t_0}{2}$
1	0	0	t_a
1	1	0	t_b
1	1	1	$\frac{t_0}{2}$

Na osnovu ove tabele moguće je izračunati vremena vođenja pojedinačnih tranzistora :

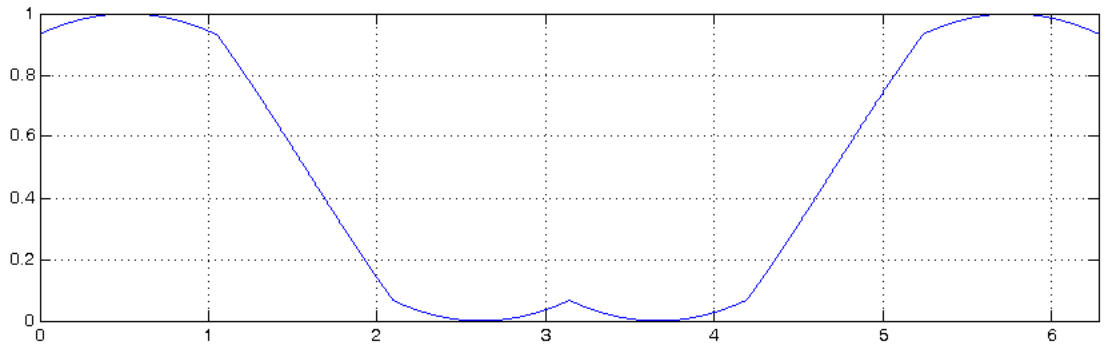
$$t_{Q1} = T_{PWM} - \frac{t_0}{2} \;;\; t_{Q3} = t_b + \frac{t_0}{2} \;;\; t_{Q5} = \frac{t_0}{2}$$

Na sličan način se dolazi do vremena vođenja pojedinačnih tranzistora za svaki sektor. U tabeli 7.2 su date vrednosti vremena vođenja tranzistora za svih šest sektora, odnosno punu rotaciju prostornog vektora.

Sektor	I	II	III	IV	V	VI
t_{Q1}	$T_{PWM}-\frac{t_0}{2}$	$t_a+\frac{t_0}{2}$	$\frac{t_0}{2}$	$\frac{t_0}{2}$	$t_b+\frac{t_0}{2}$	$T_{PWM}-\frac{t_0}{2}$
t_{Q3}	$t_b+\frac{t_0}{2}$	$T_{PWM}-\frac{t_0}{2}$	$T_{PWM}-\frac{t_0}{2}$	$t_a+\frac{t_0}{2}$	$\frac{t_0}{2}$	$\frac{t_0}{2}$
t_{Q5}	$\frac{t_0}{2}$	$\frac{t_0}{2}$	$t_b+\frac{t_0}{2}$	$T_{PWM}-\frac{t_0}{2}$	$T_{PWM}-\frac{t_0}{2}$	$t_a+\frac{t_0}{2}$

Tabela 7.2.

Na osnovu formula (7.9), (7.10) i (7.11) dobijaju se izrazi za vremena vođenja tranzistora u zavisnosti od ugaonog položaja prostornog vektora. Pošto je srednja vrednost faznog napona tokom perioda T_{PWM} linearno srazmerna vremenu vođenja tranzistora te faze, može se smatrati da su izrazi za vremena vođenja tranzistora identični izrazima za fazni napon. Grafički prikaz zavisnosti vremena t_{Q1} (napon faze A) od ugla prostornog vektora $(\omega \cdot t = 2 \cdot \pi \cdot f \cdot t)$ je dat na slici 7.4. Uzima se da je indeks modulacije m jednak jedan.

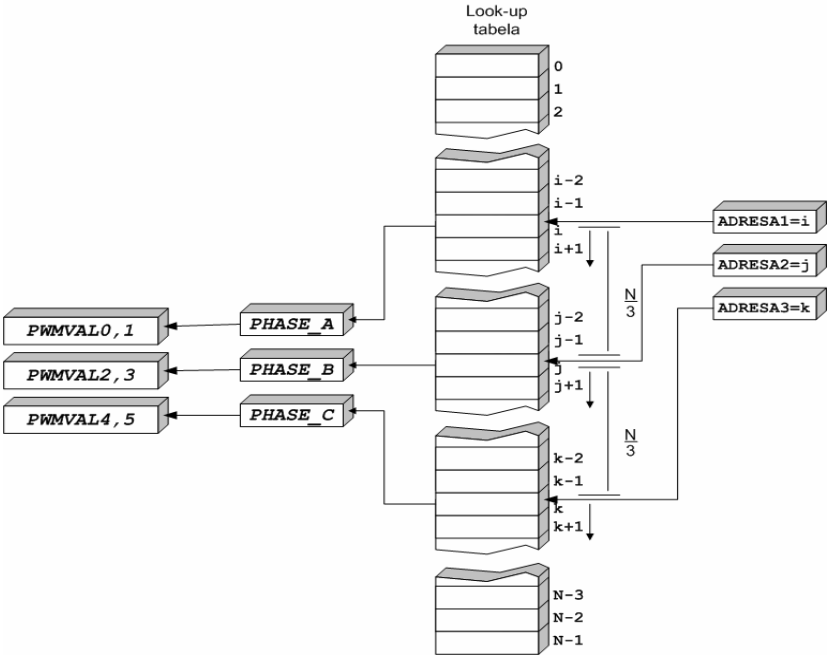


Slika 7.4. Zavisnost vremena uključenosti tranzistora Q1

Izgled faznog napona, generisanog po *Space-Vector* algoritmu, je prikazan na slici 7.4. Na krajevima invertora se javljaju tri ovakva napona fazno pomerena za 120°.

Kod *Carrier-based PWM* modulacije se referentni modulacioni signal poredi sa trougaonim modulišućim signalom čime se generišu logički signali odgovarajućeg trajanja za uključivanje tranzistora. Kao modulacioni signal može da se primeni sinusni signal, za slučaj da se želi sinusna modulacija, ili signal sa slike 7.4. čime se dobija *Space-Vector* modulacija. Signal sa slike 7.4. se može sačuvati tabelarno u *look-up* tabeli, iz koje se u diskretnim vremenskim intervalima mogu uzimati vrednosti u zavisnosti od trenutne vrednosti ugla prostornog vektora $\theta=\omega \cdot t$, gde je ω ugaona učestanost generisanog naizmeničnog napona. *Look-up* tabela se može formirati na osnovu formula iz tabele 7.2. Modulišući signal, odnosno vrednosti koje se upisuju u PWMVAL registre PWM modula, se za sve tri faze dobija iz jedne tabele. Ovo se izvodi

uzimanjem tri elementa sa pozicija međusobno pomerenih za 1/3 od ukupog broja elemenata u tabeli. Ovo je ilustrovano na slici 7.5.



Slika 7.5. *Look-up* tabela

Kao što je ilustrovano na slici, potrebna su tri pokazivača odnosno adrese. Ovim pokazivačima se adresiraju tri različite memorijske lokacije *Look-up* tabele. Međusobno rastojanje između ovih elemenata za tabelu od N elemenata iznosi N/3. Nakon što se podaci pročitaju iz tabele potrebno je uvećati pokazivače tako da se u sledećem ciklusu dobijaju nove vrednosti iz tabele. U praktičnoj realizaciji, dovoljan je jedan pokazivač koji ukazuje na novu vrednost za jednu fazu dok se vrednosti za druge dve faze dobijaju uzimanjem elemenata sa pozicija koje su udaljene za N/3 i 2N/3. Ovde je potrebno napomenuti da nakon elementa N-1 dolazi element 0, dakle ide se u krug. Iznos inkrementa pokazivača u svakom ciklusu zavisi od željene učestanosti generisanog napona. U zavisnosti kakav je signal sačuvan u *Look-up* tabeli, takav signal se dobija kao fazni napon invertora. Ako se želi *Space-Vector* PWM, u tabelu se smešta signal sa slike 7.4. a ako se želi sinusna modulacija, u tabelu će biti smešten sinusoidalni signal.

Za formiranje *Look-up* tabele SVPWM signala za primer opisan u daljem tekstu, iskorišćena je formula 8.15.

$$t_{Q1} = \frac{T_{PWM}}{2} + \frac{2}{\sqrt{3}} \cdot \left(t_{SQ1} - \frac{\max(t_{SQ1}, t_{SQ3}, t_{SQ5}) + \min(t_{SQ1}, t_{SQ3}, t_{SQ5}}{2} \right) \qquad (7.15)$$

Gde je $t_{SQ1} = \frac{T_{PWM}}{2} \cdot \cos(\alpha); t_{SQ3} = \frac{T_{PWM}}{2} \cdot \cos\left(\alpha + \frac{2 \cdot \pi}{3}\right)$ i $t_{SQ5} = \frac{T_{PWM}}{2} \cdot \cos\left(\alpha + \frac{4 \cdot \pi}{3}\right)$

Implementacija SVPWM algoritma na DSP56F801 procesoru

U direktorijumu SVPWM se nalazi program koji generiše Space-Vector PWM signale na pinovima PWM0 do PWM5. Kao i u primeru 7.2, i ovde se koristi potenciometar vezan na ANA0 ulaz ADCA modula. Na pin PA0 je vezana LED dioda koja se pali i gasi sa učestanošću od 1Hz čime se signalizira ispravan rad procesora. Potenciometrom se zadaje učestanost izlaznog napona u opsegu od 5Hz do 100Hz. Kao i u prethodim primerima, i ovde se program može podeliti na tri celine. To su glavni program, inicijalizacija i prekidne rutine. Program poseduje dve prekidne rutine. Rutinu *PWM_Reload()* poziva PWMA modul dok se rutina *adca_conversion_complete()* poziva nakon završetka A/D konverzije. Za generisajne SVPWM signala koristi se *look-up* tabela sa 1024 elementa koji su izračunati po formuli 7.15.

Glavni program

Kao i u prethodnim primerima, i ovde glavni program nakon inicijalizacije periferijskih jedinica ulazi u beskonačnu petlju iz koje se iskače samo pri obradi zahteva za prekid.

```
#include "DSP56F80x_device.h"
#include "init.h"
#include "isr.h"
#include "svtab.h"

unsigned int omega;
unsigned int angle,time,alfa;
int m;

void main(void)
{
        init_pwm();             // inicijalizacija PWM modula
        init_adc();             // inicijalizacija A/D konvertora
        init_interrupt();       // inicijalizacija kontrolera prekida

        GPIOA.PER.bit.PER0=0;       // pin PA0 se dodeljuje GPIOA modulu
        GPIOA.DDR.bit.DDR0=1;       // pin PA0 je izlazni

        PWMA.PMCTL.bit.LDOK=1;      // Load OK
        PWMA.PMCTL.bit.PWMEN=1;     // PWM enable

        alfa=45;                    // pocetna ucestanost 5Hz
        time=0;

        while(1)
        {
                if(time>=3665)              // da li je proslo 0.5 s?
                {
                        time=0;
                        GPIOA.DR.bit.DR0^=1; // LED na TD0 pinu
                }
        }
}
```

U ovom primeru se koriste tri odvojene funkcije koje inicijalizuju PWM modul ADCA modul kao i kontroler prekida. Nakon inicijalizacije ovih periferijskih jedinica, definiše se PA0 pin kao izlazni, na koji se može vezati LED dioda. Dalje se omogućava rad PWMA modula setovanjem PWMEN bita PMCTL registra kao i bita LDOK. Nakon toga program ulazi u beskonačnu petlju u kojoj se nadgleda promenljiva *time.* Ova promenljiva se inkrementira pri svakom pozivu *PWM_Reload()* prekidne rutine. Pošto se *PWM_Reload()* rutina poziva svakih 136.4μs, to znači da nakon pola sekunde promenljiva *time* dostigne vrednost 3665. Tada se ova promenljiva vraća na nulu, a stanje na LED diodi se invertuje. Promenljiva *angle* je 16-bitna promenljiva koja sadrži podatak o trenutnom ugaonom položaju prostornog vektora. Promenljiva *alfa* sadrži vrednost za koju se pri svakom pozivu prekidne rutine *PWM_Reload()* inkrementira promenljiva *angle.* Promenljiva *omega* sadrži podatak o zahtevanoj izlaznoj učestanosti napona. Za vrednost *omega*=32767 dobija se izlazna učestanost od 100Hz. Vrednost promenljive *alfa* se izračunava iz promenljive *omega* u prekidnoj rutini *adca_conversion_complete().*

Inicijalizacija PWMA modula

PWMA modul je inicijalizovan tako da radi u centralno-simetričnom modu. Brojač se taktuje *IPbus* taktom od 30MHz. Moduo brojanja je 1024, što znači da je rezolucija desetobitna. PWMA modul kontroliše PWMA0-5 pinove koji rade kao komplementarni parovi. Između paljenja jednog i gašenja drugog tranzistora jedne grane invertora (komplementarnog para) se ubacuje mrtvo vreme od 633ns.

PMCTL registar:

Base + $0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read / Write	LDFQ				HALF	IPOL2	IPOL1	IPOL0	PRSC		PWMRIE	PWMF	ISENS		LDOK	PWMEN
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Grupom bitova LDFQ se određuje koliko često će se pozivati *PWM reload* prekidna rutina. Brojač PWM modula broji u oba smera, od 0 do 1023 pa ponovo do 0, generišući na taj nacin trougaoni signal nosioc. Zahtev za prekid se može generisati nakon završetka svakog od ovih brojačkih ciklusa ili nakon nekoliko brojačkih ciklusa (perioda PWM nosioca) u zavisnosti od stanja LDFQ bitova. Izrazom:

```
PWMA.PMCTL.bit.LDFQ=1;
```

je izabrano da se *PWM reload* prekidna rutina poziva nakon svakog drugog ciklusa.

Grupom bitova PRSC se kontroliše učestanost takta PWMA modula. Ovi bitovi kontrolišu preskaler koji deli učestanost *IPbus* takta. U ovom primeru se koristi faktor deljenja 1/1 tj. direktno *IPbus* takt. Na CSM-56F801 modulu procesor radi na 60MHz, pa je *IPbus* učestanost 30MHz.

```
PWMA.PMCTL.bit.PRSC=0;
```

Ovim izrazom je setovan preskaler na faktor deljenja 1/1 sto znači da brojač PWMA modula radi na 30MHz.

Da bi *PWM reload* prekidna rutina mogla da se pozove potrebno je omoguciti ovu vrstu prekida setovanjem PWMRIE bita:

```
PWMA.PMCTL.bit.PWMRIE=1;
```

Setovanjem bita PWMEN brojač počinje sa radom. Ovaj bit se setuje u glavnom programu nakon što se obave sve inicijalizacije.

Nakon svake promene sadržaja PWMVALx ili PWMCM registra, potrebno je setovati LDOK bit da bi ove promene bile prihvaćene. Ovo se obavlja u *PWM reload* prekidnoj rutini.

PMOUT registar:

Base + $3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	PAD_EN	0	OUT CTL5	OUT CTL4	OUT CTL3	OUT CTL2	OUT CTL1	OUT CTL0	0	0	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Setovanjem bita PAD_EN se omogućava kontrola PWMA0-5 pinova od strane PWM modula.

```
PWMA.PMOUT.bit.PAD_EN=1;
```

Ostali bitovi ovog registra služe za softverski kontrolu pinova PWMA0-5. Ovi bitovi treba da budu resetovani.

PMCFG registar:

Base + $F	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	EDG	0	TOPNEG 45	TOPNEG 23	TOPNEG 01	0	BOTNEG 45	BOTNEG 23	BOTNEG 01	INDEP 45	INDEP 23	INDEP 01	WP
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bitom EDG se bira da li će PWM modul raditi u simetričnom ili asimetričnom režimu. Da bi se odabrao simetrični režim rada ovaj bit treba da bude resetovan. Bitovi TOPNEG45/23/01 kao i BOTNEG45/23/01 definišu polaritet signala za paljenje gornje i donje grupe tranzistora u invertoru.

Bitovima INDEP45/23/01 se bira da li PWM pinovi rade nezavisno ili kao komplementarni parovi. Rad u režimu komlementarnih parova se dobija resetovanjem ovih bitova.

Dakle, u ovom primeru, svi bitovi registra PMCFG treba da budu resetovani.

```
PWMA.PMCFG.all=0x0000;
```

Da bi se zadao moduo brojanja od 1024, potrebno je upisati vrednost 1023 (0x03FF) u registar PWMCM.

```
PWMA.PWMCM=0x03FF;
```

Mrtvo vreme *(dead time)* između paljenja jednog i gašenja drugog tranzistora u grani invertorskog mosta se uvodi upisom odgovarajuće vrednosti u PMDEADTM registar. Trajanje mrtvog vremena je jednako trajanju takta PWM modula pomnoženom sa 2*PMDEADTM-1. Ako

se želi mrtvo vreme od 633ns, pri taktu PWM modula od 30MHz, potrebno je upisati vrednost 10 u PMDEADTM registar.

```
PWMA.PMDEADTM=10;
```

Na kraju je potrebno resetovati sve bitove PMDISMAP1 i PMDISMAP2 registara.

```
PWMA.PMDISMAP1=0x0000;
PWMA.PMDISMAP2=0x0000;
```

Listing funkcije *init_pwm():*

```
/*********************************************************
* init_pwm() - inicijalizacija PWMA modula           *
*********************************************************/
void init_pwm(void)
{
        OCCS.IOSCTL=90;              // IPbus_clock(new)=IPbus_clock(old)*1.0874
        PWMA.PMCTL.bit.LDFQ=1;       // load frequency=every second PWM opportunities
        PWMA.PMCTL.bit.PWMRIE=1;  // PWM reload interrupt enable
        PWMA.PMCTL.bit.PRSC=0;       // PWM clock frequency = Fipbus/1
        PWMA.PMOUT.bit.PAD_EN=1; // PWM output pad enable
        PWMA.PWMCM=0x03FF;           // PWM period=1023*2*PWM clock frequency
        PWMA.PMCFG.all=0x0000;       // clear PWM Configure register
        PWMA.PMDEADTM=10;            // dead time=(2*10-1)/Fipbus=633.33ns
        PWMA.PMDISMAP1=0x0000;    // clear disable map registers
        PWMA.PMDISMAP2=0x0000;
}
```

Inicijalizacija ADCA modula

Rutina za inicijalizaciju ADCA modula je identicna delu funkcije koji inicijalizuje ADCA modul u primeru Test_ADC:

```
/*********************************************************
* init_adc() - inicijalizacija ADCA modula             *
*********************************************************/
void init_adc(void)
{
        ADCA.ADCR1.all=0;                    // clear ADCR1 register
        ADCA.ADCR1.bit.EOSIE=1;              // End of scan interrut enable
        ADCA.ADCR2.bit.DIV=5;                // ADC clock = 30MHz/(2*(5+1))=2MHz
        ADCA.ADSDIS.bit.DS1=1;               // Disable SAMPLE1.
}
```

Inicijalizacija kontrolera prekida ITCN

Program poseduje dve prekidne rutine. Rutinu *PWM_Reload()* poziva PWMA modul dok se rutina *adca_conversion_complete()* poziva nakon završetka A/D konverzije, kao i u Test_ADC primeru. Zahtev za prekid *PWM reload* se prosleđuje na kanal CH1 dok se zahtev za prekid ADCA modula prosleđuje na CH0 kanal.

```
ITCN.GRP13.bit.PLR55=1;
ITCN.GRP14.bit.PLR59=2;
```

Registar GRP13 sadrži grupu bitova PLR55 kojom se definiše na koji se kanal prosleđuje *ADCA Conversion complete* zahtev za prekid. Polje PLR59 registra GRP14 određuje kanal na koji se preusmerava *PWM Reload* zahtev za prekid.

Na kraju je potrebno omogućiti prekide na CH1 iCH0 kanalima.

```
/*************************************************************
* init_interrupt() - inicijalizacija ITCN kontrolera      *
*************************************************************/
void init_interrupt(void)
{
        ITCN.GRP13.bit.PLR55=1;
        ITCN.GRP14.bit.PLR59=2;
        ENABLE_CH0;
        ENABLE_CH1;
}
```

Prekidna rutina *PWM_reload()*

Prekidna rutina *PWM_Reload()* predstavlja najznačajniji deo programa SVPWM. Ova prekidna rutina se poziva na svakih 136.4μs, i njen zadatak je da obezbedi nove vrednosti za PWMVAL0-5 registre za narednu PWM periodu. Prekidna rutina koristi globalne promenljive *angle* i *alfa*, na osnovu kojih izračunava pozicije u *look-up* tabeli sa kojih se uzimaju vrednosti za PWMVALx registre. Pošto tabela poseduje 1024 elementa, a promenljiva ugao je 16-bitna, potrebno je najpre promenljivu ugao podeliti sa 64 što se postiže logičkim pomeranjem u levo za šest mesta. Dalje se za svaki par PWMVAL registara izračunava nova vrednost na osnovu pročitanog podatka iz tabele i indeksa modulacije m. Množenje indeksom modulacije m obavlja funkcija __mult(a,b), gde su a i b celi brojevi koji se pri množenju tretiraju kao brojevi sa fiksnim zarezom. Ovo je malo detaljnije opisano u delu teksta koji se bavi *adca_conversion_complete()* prekidnom rutinom. Nakon što su obezbeđene nove vrednosti PWMVAL registara, setuje se LDOK bit da bi se one i prihvatile od strane PWMA modula. Nakon toga se startuje A/D konverzija, uveća se promenljiva *angle* za iznos koji zavisi od zadate izlazne učestanosti I inkrementira se promenljiva *time.* Na kraju se briše PWMF fleg i izlazi iz prekidne rutine.

```
#pragma interrupt
//*************************************************************************//
//                                                                         //
// Prekidna rutina  Reload PWM A                    ($76)                  //
// Poziva se svakih 136us                                                  //
//                                                                         //
// Globalne promenljive : angle, alfa                                      //
// Lokalne promenljive  : addr1, addr2, addr3, t                           //
//                                                                         //
//*************************************************************************//
void PWM_Reload()
{
        unsigned int addr1,addr2,addr3,result;
        signed int temp;

        addr1=(angle)>>6;                   // look up adresa za fazu A
        addr2=(angle+21845)>>6;             // look up adresa za fazu B
        addr3=(angle+43691u)>>6;            // look up adresa za fazu C

        temp=svtab[addr1];
        temp=__mult(temp,m);
        result=(unsigned int)(32767+temp);
        result>>=6;

        PWMA.PWMVAL0=result;
        PWMA.PWMVAL1=result;

        temp=svtab[addr2];
        temp=__mult(temp,m);
        result=(unsigned int)(32767+temp);
        result>>=6;

        PWMA.PWMVAL2=result;
        PWMA.PWMVAL3=result;

        temp=svtab[addr3];
        temp=__mult(temp,m);
        result=(unsigned int)(32767+temp);
        result>>=6;

        PWMA.PWMVAL4=result;
        PWMA.PWMVAL5=result;

        PWMA.PMCTL.bit.LDOK=1;
        ADCA.ADCR1.bit.START=1;             // start ADC

        angle+=alfa;
        time++;

        PWMA.PMCTL.bit.PWMF=0;              // Brisanje PWMF flega
}
```

Prekidna rutina *adca_conversion_complete()*

Ova prekidna rutina se poziva od strane ADCA modula po završetku A/D konverzije. U prekidnoj rutini se očitava rezultat A/D konverzije napona sa potenciometra, kojim se zadaje učestanost generisanog trofaznog naizmeničnog napona, i izračunava se indeks modulacije m. Pre izlaska iz prekidne rutine se briše EOSI fleg upisom logičke jedinice na njegovo mesto. Na osnovu izmerenog napona izračunava se ugaona učestanost *omega* na osnovu koje se dobija priraštaj *alfa* promenljive *angle,* u prekidnoj rutini *PWM_Reload()*. Ako je zahtevana učestanost generisanog napona manja od 50Hz, odnosno ako je omega manje od 2048, indeks modulacije m je manji od 1. Za učestanosti veće ili jednake od 50Hz indeks modulacije je jednak jedinici. Ovde je upotrebljena frakcionalna aritmetika. Naime, indeks modulacije m je definisan kao označeni ceo broj (16 bita). Bit najveće težine je bit znaka, pa se za predstavljanje vrednosti koristi ostalih 15 bitova. Ovakav ceo broj se može posmatrati kao broj sa fiksnim zarezom, gde je celobrojni deo jednak nuli a decimalni deo predstavljen sa 15 bita. To znači da je broj 32768 isto što i –1.0. Broj 32767 predstavlja broj 0.99996948. Dakle svaki broj manji od 1 se može predstaviti kao broj sa fiksnim zarezom množenjem tog broja sa 32768. Tako na primer broj 0.25 se predstavlja kao $0.25 \cdot 32768 = 8192$, broj 0.89 se predstavlja kao $0.89 \cdot 32768 = 29164$. Jasno je da se brojevima sa fiksnim zarezom može predstaviti samo ograničen broj realnih brojeva.

Osnovna prednost primene brojeva sa fiksnim zarezom je u veoma brzom množenju ovakvih brojeva upotrebom celobrojnog množača aritmetičko logičke jedinice. *Code Warrior* razvojno okruženje koristi C kompajler koji poseduje ugrađene funkcije za rad sa brojevima u fiksnom zarezu. Jedna od tih funkcija je __mult(a,b) gde su a i b, kao i rezultat ove funkcije celi brojevi kojima su predstavljene odgovarajuće vrednosti u fiksnom zarezu.

Maksimalna moguća učestanost generisanog napona je ograničena na 100Hz. Minimalna učestanost je ograničena na 5Hz.

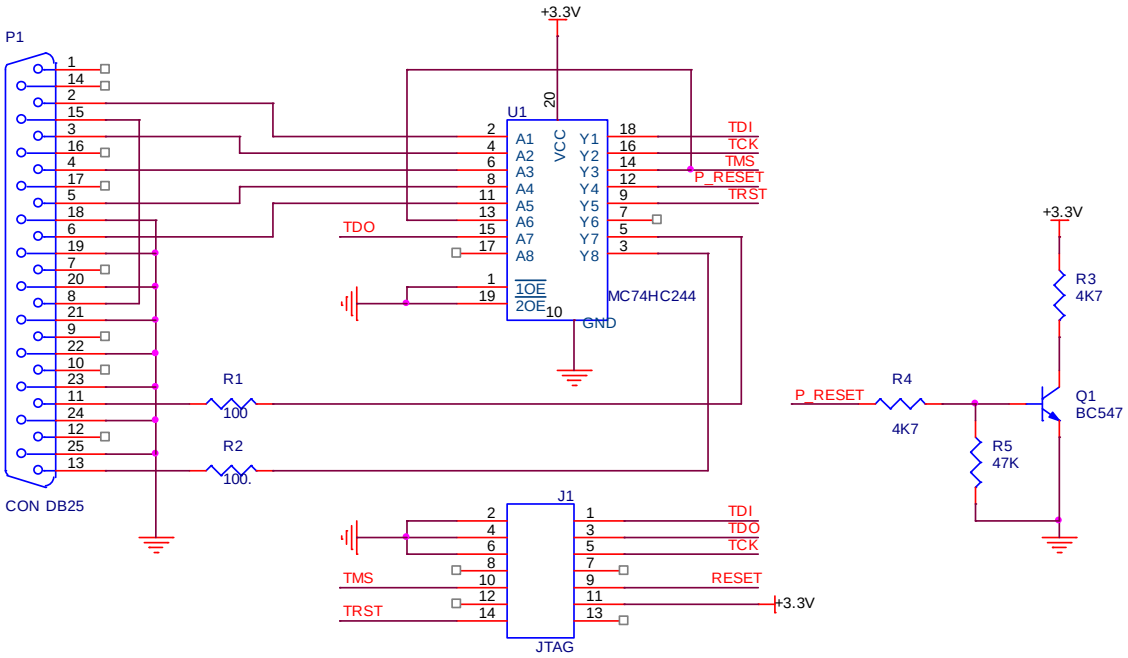
```
#pragma interrupt
//*****************************************************************************//
//                                                                             //
// Prekidna rutina       ADC A Conversion complete    ($6e)                    //
//                                                                             //
// Poziva se posle svakog izvrsavanja Reload PWM A prekidne rutine             //
// Globalne promenljive : omega, alfa, m                                       //
// Lokalne promenljive  : adc_result                                           //
//                                                                             //
//*****************************************************************************//
void adca_conversion_complete()
{
        int adc_result;
        adc_result=ADCA.ADRSLT0;                    // ocitavanje rezultata A/D konverzije
        omega=adc_result>>=3;
        if(omega<OMEGA_5Hz) omega=OMEGA_5Hz;// donje ogrranicenje ucestanosti na 5Hz
```

```
        if(omega<OMEGA_50Hz)
        {
                m=(signed int)omega<<4;          // m ; indeks modulacije
        }
        else m=32767;                            // m=1 za frekvenciju vecu do 50Hz
        alfa=__mult(omega,K_OMEGA);              // ugaoni pomeraj se dobija množenjem
                                                 // omega sa koeficijentom K_OMEGA

        ADCA.ADSTAT.bit.EOSI=1;                  // brisanje End of Scan interrupt flega
}
```

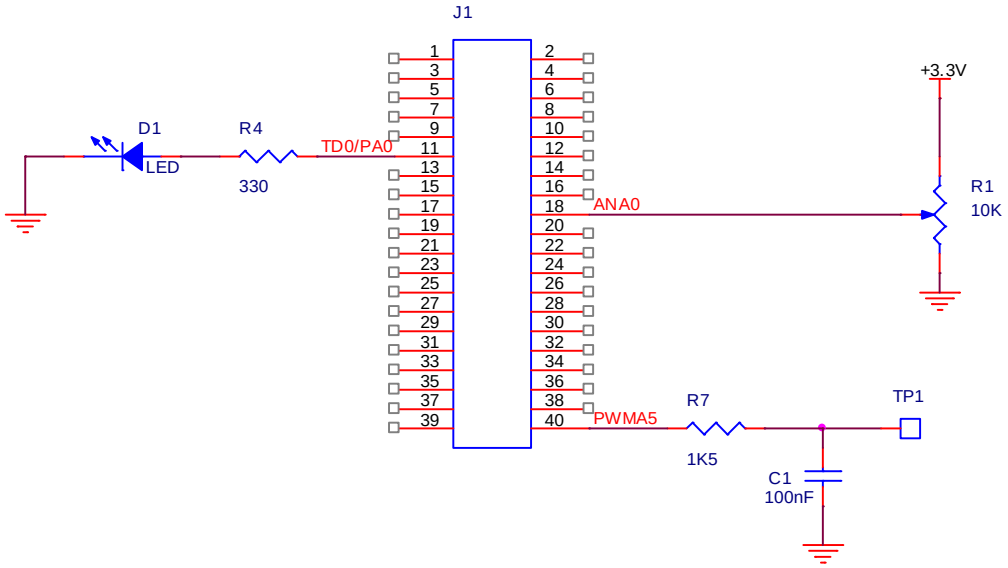
Kao i u prethodnim primerima, u tabeli vektora prekidnih rutina su na odgovarajuce poyicije upisane adrese *adca_conversion_complete()* i *PWM_Reload()* prekidnih rutina.

```
    jmp M56801_intRoutine     ; SCI #0 transmitter ready ($66)
    jmp M56801_intRoutine     ; SCI #0 receiver error    ($68)
    jmp M56801_intRoutine     ; SCI #0 receiver full     ($6a)
    jmp M56801_intRoutine     ; ADC B Conversion complete($6c)
    jmp Fadca_conversion_complete     ; ADC A Conversion complete($6e)
    jmp M56801_intRoutine     ; ADC B zero crossing/error($70)
    jmp M56801_intRoutine     ; ADC A zero crossing/error($72)
    jmp M56801_intRoutine     ; Reload PWM B             ($74)
    jmp FPWM_Reload     ; Reload PWM A             ($76)
    jmp M56801_intRoutine     ; PWM B Fault              ($78)
    jmp M56801_intRoutine     ; PWM A Fault              ($7a)
    jmp M56801_intRoutine     ; PLL loss of lock         ($7c)
    jmp M56801_intRoutine     ; low voltage detector     ($7e)
```

Prilog A. Šema JTAG adaptera



Prilog B. Povezivanje CSM56F801 modula sa potenciometrom, LED diodom i RC filtrom



Literatura

[1] “DSP56F800 User Manual”, DSP56F801-7UM Rev. 6, 07/2004, Freescale Semiconductor

[2] “DSP56F800 16-Bit Digital Signal Processor Family Manual”, DSP56F800FM/D Rev. 3.0 12/2003, Freescale Semiconductor

[3] “Code Warrior Development Studio for Freescale 56800/E Hybrid Controllers: DSP56F80x/DSP56F82x Family Targeting Manual”, Metrowerks 2005

[4] “Programming TMS320x280x and 281x Peripherals in C/C++” Application Note - spraa85, Texas Instruments, 2005

[5] Laslo Kraus, “Programski jezik C sa rešenim zadacima” Akademska misao, Beograd 2004