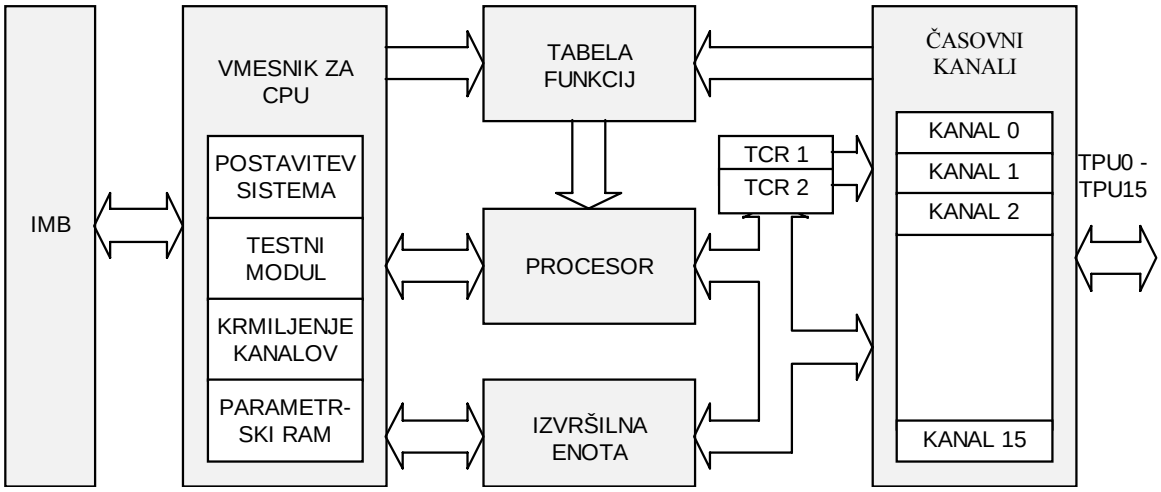
7. ČASOVNO-PROCESNA ENOTA - TPU

Mikrokrmilnik MC68332 vsebuje poleg CPU-ja še dodatni procesor, t.i. *časovno-procesno enoto* (angl. Time Processing Unit - TPU). Zato govorimo o multiprocesorski konfiguraciji. TPU je zadolžena za zajemanje in izdajanje binarnih vrednosti (1 in 0) v dosvisnosti od časa. Slika 7. 1 kaže blokovno shemo TPU-ja (glej tudi pogl. 3).



Slika 7. 1: Blokovna shema časovno-procesne enote

TPU vsebuje nekatere vnaprej programirane časovne funkcije, ki jih je za njihovo uporabo treba ustrezno parametrirati[1]. Večino teh funkcij lahko uporabimo na kateremkoli izmed šestnajstih kanalov. Funkcije razdelimo na dve skupini:

- **vhodne časovne funkcije** so predvidene za zajemanje in ovrednotenje binarnih časovnih signalov,
- za izdajanje vnaprej definirane sekvence pulzov uporabljamo **izhodne časovne funkcije.**

Z ozirom na že programirane TPU funkcije imamo na voljo dve enačici mikrokrmilnika MC68332: A in G. Pri prvi vsebuje TPU splošnejše funkcije, druga pa je prilagojena krmiljenju in regulaciji elektromotorjev. Nekatere izmed časovnih funkcij so:

- **Zajemanje vhodnih signalov** oz. **štetje prehodov na vhodu** (angl. Input Capture/Input Transition Counter - ITC) - (inačica A). S funkcijo ITC merimo čas med izbranimi prehodi vhodnega signala.

[1] Časovne funkcije lahko prilagodimo našim posebnim zahtevam, za kar pa je potrebno dodatno poznavanje najnižjega nivoja programiranja: *mikroprogramiranja*. Vsak ukaz (npr. CPU ukaz) je sestavljen iz sekvence še enostavnejših ukazov. Delo z mikroprogramom je izredno zahtevno in uporabniku načeloma onemogočeno, zato ga ne bomo posebej obravnavali.

- **Diskretni vhodi/izhodi** (angl. Discrete Input/Output) - (A). Če izberemo vhodno funkcijo, se v posebnem registru shranjuje stanje prehodov v zadnjih šestnajstih urinih ciklih TPU. Pri izhodni funkciji pa se na izhod pošlje vnaprej definirana sekvenca stanj.
- **Primerjanje vhodov** (angl. Output Compare) - (A). Generiranje stanja 1, 0 ali sprememba prejšnjega stanja (angl. toggle) z določeno zakasnitvijo.
- **Pulzno-širinska modulacija** (angl. Pulse-Width Modulation) - (A) je zelo pogosto uporabljana funkcija, o kateri bomo več povedali v nadaljevanju (pogl. 10.3.5.7). Z njo lahko določimo vlak pulzov, ki jim lahko spreminjamo periodo in trajanje stanja 1.
- **Koračni motor** (angl. Stepper Motor) - (A). Vrtenje koračnih motorjev dosežemo z generiranjem dveh vlakov pulzov, ki sta premaknjena za 90°, na statorskih navitjih. Smer vrtenja je odvisna od zaporedja faz (pogl. 12.6). Funkcija omogoča programirano pospeševanje, zaviranje in obratovalno stanje, kjer je trenutna hitrost motorja proporcionalna frekvenci vhodnih pulzov.
- **Akumulacija trajanja periode** oz. **širine pulzov** (angl. Period/Pulse-Width Accumulator - PPWA) - (A). Funkcija omogoča merjenje skupnega trajanja izbranega števila period ali skupno trajanje pulzov v določenem številu period.
- **Zajemanje dveh časovno premaknjenih pulzov** (angl. Quadrature Decode - QDEC) - (A) uporabimo pri ugotavljanju kota zasuka rotorja z inkrementalnim dajalnikom (pogl. 12.5.2).
- **Merjenje frekvence** (angl. Frequency Measurement - FQM) - (G). Znotraj vnaprej določenega časovnega intervala štejemo periode.
- **Asinhronski oddajnik/sprejemnik** (angl. Universal Asynchronous Receiver/Transmitter - UART) - (G) smo že spoznali v poglavju 6.3 o SCI modulu. Možna je uporaba osmih UART-ov.
- **Večfazna komutacija** (angl. Multiphase Motor Commutation - COMM) - (G) omogoča generiranje sekvenc za krmiljenje enosmernih elektronsko komutiranih motorjev (angl. brushless motors). Funkcijo lahko uporabimo tudi na trifaznih motorjih.
- Pogoj za pravilno delovanje elektronsko komutiranih motorjev (glej prejšnjo funkcijo) je preklapljanje faz sinhrono s kotom rotorja oz. s položajem trajnega magneta, za kar so zadolženi Hallovi senzorji. V ta namen uporabljamo **funkcijo dekodiranja signala iz Hallovih senzorjev** (angl. Hall Effect Decode - HALLD) - (G), ki deluje skupaj s funkcijo COMM. Možno je dekodiranje signalov iz dveh ali treh senzorjev.

O eni izmed funkcij (PWM) in načinu parametriranja TPU-ja bomo več govorili v nadaljevanju. Na tem mestu omenimo osnovno prednost posebnega časovnega procesorja:

- Funkcije TPU bistveno zmanjšajo potrebo po aparaturni opremi za zajemanje, obdelavo in generiranje časovnih pulzov (npr. posebni števci, flip-flopi itd.).
- TPU deluje povsem neodvisno od CPU, s katero komunicira le preko prekinitev. Na ta način se občutno zmanjša potreba po posegih CPU pri obdelavi časovnih funkcij.

TPU vsebuje 16 kanalov (angl. channels) oz. priključnih sponk, ki jih označujemo z oznakami TPU0 - TPU15 (glej sliki 7. 1 in 3. 3). Na vsakem od teh kanalov lahko uporabimo poljubno časovno funkcijo. Kanali lahko obratujejo samostojno ali v povezavi z drugimi kanali (npr. pri generiranju krmilnih pulzov za koračni motor sodelujeta dva kanala, ki sta konfigurirana kot izhoda).

7.1 Registri in RAM pomnilnik TPU modula

V prejšnjih poglavjih smo povedali, da so nekatera področja v pomnilniški mapi MC68332 rezervirana za registre in pomnilnike podmodulov. Za TPU je rezervirano področje $Yffe00_{HEX}$ - $Yfffeff_{HEX}$[2]. V njem se nahajata:

- **registrska mapa** in
- **parametrski RAM**.

Na omenjenih naslovih se mora nahajati RAM pomnilnik ker moramo imeti možnost sprotnega spreminjanja vsebin obeh področij.

7.1.1 Registrska mapa TPU-ja

Razlikujemo tri vrste TPU registrov:

- **Registri za sistemsko konfiguracijo,**
- **statusni registri in registri za krmiljenje kanalov** ter
- **registri za podporo razvoja in preizkušanje**.

Osnovne funkcije registrov so:

- določanje osnovne konfiguracije celotnega TPU-ja (osnovni takt delovanja, prekinitvena prioriteta in ustrezni vektorji...) ter
- konfiguriranje posameznih kanalov (izbira časovne funkcije in njenih parametrov, dodelitev in spremljanje izvajanja prekinitev...).

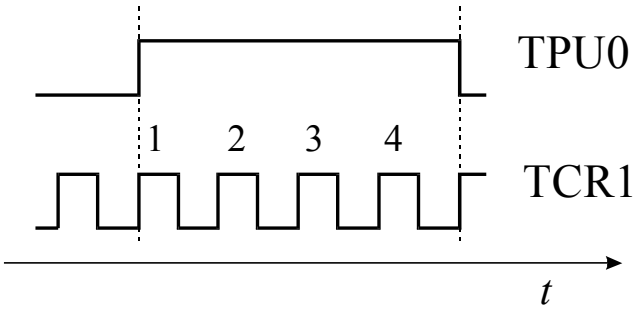
Registrsko mapo TPU-ja kaže slika 7. 2.

Vsi uporabljeni kanali oz. njihove funkcije so sinhronizirani na enega izmed dveh 16-bitnih prostotekočih časovnikov TCR1 in TCR2 (angl. Timer Count Registers - registri za štetje časa). Frekvenci obeh moramo nastaviti pri inicializaciji TPU. Dolžina periode TCR1 je mnogokratnik dolžine periode osnovnega dajalnika takta MC68332 (v našem primeru obravnavamo le inačico s frekvenco 16,77 MHz, to je s periodo 59,6 ns), za določanje frekvence TCR2 pa lahko uporabimo še nek dodatni zunanji dajalnik takta (glej poglavje 7.1.1.1). Na sliki 7. 3 je prikazan primer generiranja pulza (signal 1) na kanalu TPU0 v trajanju štirih period TCR1 (funkcija PWM).

[2] Y je f_{HEX} ali 7_{HEX}, odvisno od izbire bita MM v SIM registru MCR (naslov $Yffa00_{HEX}$). Običajno velja Y = f_{HEX}.

	BESEDA				
Naslov:	15 **BYTE *n*** 8	7 **BYTE *n*+1** 0			
$Yffe00_{HEX}$	Module Configuration Register (TPUMCR)				
$Yffe02_{HEX}$	Test Configuration Register				
$Yffe04_{HEX}$	Development Support Control Register				
$Yffe06_{HEX}$	Development Support Status Register				
$Yffe08_{HEX}$	Interrupt Configuration Register (TICR)				
$Yffe0a_{HEX}$	Channel Interrupt Enable Register (CIER)				
$Yffe0c_{HEX}$	Channel Function Select Register 0 (CFSR 0)				
$Yffe0e_{HEX}$	Channel Function Select Register 1 (CFSR 1)				
$Yffe10_{HEX}$	Channel Function Select Register 2 (CFSR 2)				
$Yffe12_{HEX}$	Channel Function Select Register 3 (CFSR 3)				
$Yffe14_{HEX}$	Host Sequence Register 0 (HSQR 0)				
$Yffe16_{HEX}$	Host Sequence Register 0 (HSQR 1)				
$Yffe18_{HEX}$	Host Service Request Register 0 (HSRR 0)				
$Yffe1a_{HEX}$	Host Service Request Register 1 (HSRR 1)				
$Yffe1c_{HEX}$	Channel Priority Register 0 (CPR 0)				
$Yffe1e_{HEX}$	Channel Priority Register 1 (CPR 1)				
$Yffe20_{HEX}$	Channel Interrupt Status Register (CISR)				
$Yffe22_{HEX}$	Link Register				
$Yffe24_{HEX}$	Service Grant Latch Register				
$Yffe26_{HEX}$	Decoded Channel Number Register				
$Yffe28_{HEX}$ ..... $Yffeff_{HEX}$	**ZASEDENO**				

Slika 7. 2: Registrska mapa TPU-ja



Slika 7. 3: Primer sinhronizacije časovnega signala s TCR1 ali TCR2

Delo s TPU-jem poteka v treh korakih:

1. Inicializacija: nastavitev osnovne konfiguracije TPU, definiranje uporabljenih kanalov in njihovih funkcij, zagon TPU.
2. Izvajanje TPU funkcij neodvisno od CPU-ja: TCR1 ali TCR2 obratujeta kot prostotekoča dajalnika takta.
3. Prekinitev delovanja TPU - možna le ob resetiranju celotnega MC68332 oz. ob postavitvi bita STOP v registru TPUMCR. Drugače izvajanja funkcij TPU ne moremo prekiniti, celo z ABORT prekinitvijo ne.

Med obratovanjem TPU-ja neodvisno od izvajanja CPU instrukcij (točka 2) lahko obe procesorski enoti komunicirata (CPU je nadrejena). Pri vseh funkcijah je možno generiranje zahteve po prekinitvi delovanja CPU-ja s strani TPU-ja. Npr. takšno zahtevo je možno generirati na koncu vsake periode TPU funkcije PWM. Ali se bo CPU na zahtevo odzval, je odvisno od nastavitev v fazi inicializacije (glej TPU registra TICR in CIER v nadaljevanju).

V naslednjih podpoglavjih si bomo ogledali registre TPU-ja ter nekatere njihove funkcije.

7.1.1.1 Register za konfiguracijo TPU modula (TPUMCR)

TPUMCR Naslov: $Yffe00_{HEX}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP	TCR1 PRESC.		TCR2 PRESC.		EMU	T2CG	STF	SUPV	PSCK	0	0	ARBITRAŽA PREKINITEV IARB			
RES:															
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

STOP - Stop bit
Setiranje tega bita pomeni ustavitev delovanja TPU-ja, kar signalizira bit STF.

TCR1 presc. (TCR1 Prescaler Control)
PSCK - Prescaler Clock

Ti trije biti določajo frekvenco inkrementiranja časovnika TCR1 po naslednji tabeli (slika 7. 4):

TCR1 presc.	PSCK = 0		PSCK = 1	
	Št. period	Dolž. periode (pri 16 MHz)	Št. period	Dolž. periode (pri 16 MHz)
00	32	≈ 2 μs	4	≈ 250 ns
01	64	≈ 4 μs	8	≈ 500 ns
10	128	≈ 8 μs	16	≈ 1 μs
11	256	≈ 16 μs	32	≈ 2 μs

Slika 7. 4: Nastavljanje frekvence TCR1 v odvisnosti od bitov TCR1 presc. in PSCK

Podoben princip velja tudi za polja **TCR2 presc**.

IARB biti - Interrupt Arbitration
Vsak modul v MC68332, ki je povezan na intermodularni bus (IMB), lahko generira prekinitev delovanja CPU-ja. IARB polje določa prioriteto modula (tukaj TPU-ja), ki je potrebna za arbitražo, ko pride do hkratne zahteve po prekinitvi s strani več modulov.

7.1.1.2 Register za konfiguracijo prekinitev TPU-ja (TICR)

TICR (TPU Interrupt Configuration Register) vsebuje le dve polji: nivo zahteve po prekinitvi kanalov (angl. Channel Interrupt Request Level) in bazni prekinitveni vektor kanalov (angl. Channel Interrupt Base Vector).

TICR Naslov: $YFFE08_{HEX}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**CHANNEL INTERR. REQUEST LEVEL**			**CHANNEL INTERRUPT BASE VECTOR**				**0**	**0**	**0**	**0**
RES:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

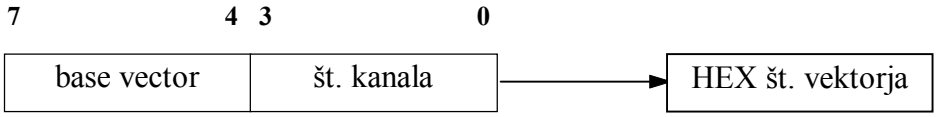
V poglavju o prekinitvah smo povedali, da obstaja veliko razlogov za generiranje prekinitev (npr. softverska: deljenje z ničlo; hardverska: zunanji signal). Zaradi izvajanja ustreznega podprograma je treba ugotoviti kaj je vzrok za prekinitev. Prekinitveni vektorji so shranjeni v posebnem področju RAM-a. Vektorji vsebujejo začetne naslove prekinitvenih podprogramov in morajo biti definirani pred zagonom glavnega programa (med inicializacijo). Večina vektorjev ustreza točno določenemu tipu prekinitve, nekatere vektorje pa lahko dodelimo poljubnim prekinitvam.

Vsak vektor vsebuje svojo zaporedno številko; njegov dejanski naslov izračunamo z izrazom:

$$\text{zaporedna številka vektorja} \cdot 4 + (\text{VBR}).$$

Spomnimo se: CPU register VBR (Vector Base Register) vsebuje lokacijo ničtega vektorja v RAM pomnilniku. Zaporedno številko vektorja množimo s štiri, saj je vsak vektor sestavljen iz štirih bytov ali ene dolge besede (le na ta način lahko zapišemo 24-bitni naslov).

Prekinitve, ki jih generirajo TPU funkcije na posameznih kanalih, sodijo med tiste, katerih vektorji nimajo vnaprej določenih pozicij v vektorski mapi, zato jim moramo pred zagonom podati ustrezne naslove. Zaporedna številka vektorja je določena z vsebino polja Channel Interrupt Base Vector v TICR registru (zgornji nibble številke vektorja) in številko TPU kanala (spodnji nibble - od 0_{HEX} do f_{HEX}):



Postavljanje prekinitvenega vektorja je opisano v poglavju 7.2.

Prioriteto TPU prekinitev določimo v polju Channel Interrupt Request Level, kjer ima prekinitev z najvišjo prioriteto zaporedno številko sedem (vsi biti setirani).

Do sedaj opisana registra vsebujeta splošne parametre TPU. V naslednji skupini registrov pa so splošni parametri za posamezne kanale.

7.1.1.3 Register za omogočanje prekinitev kanalov (CIER)

Časovne funkcije na posameznih kanalih lahko generirajo prekinitve le, če je v CIER registru (angl. Channel Interrupt Enable Register) setiran pripadajoči biti.

CIER — Naslov: $Yffe0a_{HEX}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
kanal 15	kanal 14	kanal 13	kanal 12	kanal 11	kanal 10	kanal 9	kanal 8	kanal 7	kanal 6	kanal 5	kanal 4	kanal 3	kanal 2	kanal 1	kanal 0
RES:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

7.1.1.4 Statusni register prekinitev kanalov (CISR)

Če na določenem TPU kanalu pride do prekinitve, se to označi s stanjem 1 v pripadajočem bitu registra CISR (angl. Channel Interrupt Status Register).

CISR — Naslov: $Yffe20_{HEX}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
kanal 15	kanal 14	kanal 13	kanal 12	kanal 11	kanal 10	kanal 9	kanal 8	kanal 7	kanal 6	kanal 5	kanal 4	kanal 3	kanal 2	kanal 1	kanal 0
RES:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

7.1.1.5 Registri za izbiro časovnih funkcij kanalov (CFSR0, CFSR1, CFSR2, CFSR3)

Z naslednjimi štirimi CFSR registri (angl. Channel Function Select Registers) izberemo časovno funkcijo za posamezne kanale.

CFSR0 — Naslov: $Yffe0c_{HEX}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KANAL 15				KANAL 14				KANAL 13				KANAL 12			
RES:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CFSR1 — Naslov: $Yffe0e_{HEX}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KANAL 11				KANAL 10				KANAL 9				KANAL 8			
RES:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CFSR2 — Naslov: $Yffe10_{HEX}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KANAL 7				KANAL 6				KANAL 5				KANAL 4			
RES:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CFSR3 Naslov: $Yffe12_{HEX}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KANAL 3				KANAL 2				KANAL 1				KANAL 0			
RES:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabela na sliki 7. 5 kaže bitne vzorce oz. kode nekaterih funkcij. V tabeli so zbrane tudi opcije posameznih funkcij, ki jih definirajo registri HSQR in HSRR (opisani v nadaljevanju). Vrednost 0_{HEX} v polju "koda funkcije" pomeni, da kanalu ni dodeljena nobena funkcija (kanal je neaktiven).

Naziv funkcije (kratica)	**Hex koda funkcije** ***pogl. 7.1.1.5***	**Koda HSRR** ***pogl. 7.1.1.7***	**Koda HSQR** ***pogl. 7.1.1.6***
PPWA	f	0 = nič 1 = neuporabljeno 2 = inicializacija 3 = neuporabljeno	0 = 24-bitna perioda 1 = 16-bitna perioda+povezava 2 = 24-bitna širina pulza 3 = 16-bitna širina pulza+povezava
DIO	8	0 = nič 1 = postavi izhod v stanje 1 2 = postavi izhod v stanje 0 3 = inicializacija, vhod določen 3 = inicializacija, periodični vhod 3 = ažuriraj parameter statusa nožice	0 0 = Zajemanje sprememb: shrani stanje nožice ob spremembi 0 = Zajemanje sprememb: shrani stanje nožice ob spremembi 0 = Zajemanje sprememb: shrani stanje nožice ob spremembi 1 = Ujemanje: shrani stanje nožice pri MATCH_RATE 2 = Shrani stanje nožice v HSR11
ITC	a	0 = nič 1 = inicializacija 2,3 = neuporabljeno	0 = brez povezav, eno zajemanje 1 = brez povezav, stalno zajemanje 2 = povezava z drugimi kanali, eno zajemanje 3 = povezava z drugimi kanali, stalno zajemanje
PWM	9	0 = nič 1 = signalizacija zahteve po ažuriranju 2 = inicializacija 3 = neuporabljeno	neuporabljeno
SM	d	0 = nič 1 = nič 2 = inicializacija 3 = zahtev po koraku	neuporabljeno

Slika 7. 5: Izbiranje nekaterih funkcij in njihovih opcij

7.1.1.6 Registra za sekvence CPU-ja (HSQR0 in HSQR1)

HSQR0 Naslov: $Yffe14_{HEX}$

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Kanal 15	Kanal 14	Kanal 13	Kanal 12	Kanal 11	Kanal 10	Kanal 9	Kanal 8
RES:							
0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

HSQR1 Naslov: $Yffe16_{HEX}$

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Kanal 7	Kanal 6	Kanal 5	Kanal 4	Kanal 3	Kanal 2	Kanal 1	Kanal 0
RES:							
0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

Registra sta namenjena izbiri načina obratovanja izbrane časovne funkcije. Dostop do HSQR (angl. Host Sequence Register) ima vodilni modul na IMB (angl. master ali host), običajno pa je to kar CPU. Pomen bitov je odvisen od izbrane časovne funkcije, kot je to prikazano na sliki 7. 5.

7.1.1.7 Registra za zahtevo po servisiranju CPU-ja (HSRR0 in HSRR1)

HSRR0 Naslov: $Yffe18_{HEX}$

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Kanal 15	Kanal 14	Kanal 13	Kanal 12	Kanal 11	Kanal 10	Kanal 9	Kanal 8
RES:							
0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

HSRR1 Naslov: $Yffe1a_{HEX}$

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Kanal 7	Kanal 6	Kanal 5	Kanal 4	Kanal 3	Kanal 2	Kanal 1	Kanal 0
RES:							
0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

HSRR registra (angl. Host Service Request Registers) določata eno izmed treh možnih interakcij med določenim TPU kanalom in CPU-jem oz. nekim drugim nadrejenim modulom IMB vodila (slika 7. 5). Četrta možna kombinacija bitov (00_{BIN}) je začetno stanje, ki se ponovno vzpostavi po servisiranju kanala, zato tudi označuje konec izvajanja postopka, ki je bil izbran z eno izmed ostalih treh kombinacij.

7.1.1.8 Registra za določanje prioritete kanalov (CPR0 in CPR1)

CPR0 Naslov: $Yffe1c_{HEX}$

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Kanal 15	Kanal 14	Kanal 13	Kanal 12	Kanal 11	Kanal 10	Kanal 9	Kanal 8
RES: 0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

CPR1 Naslov: $Yffe1e_{HEX}$

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Kanal 7	Kanal 6	Kanal 5	Kanal 4	Kanal 3	Kanal 2	Kanal 1	Kanal 0
RES: 0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

Za določanje prioritete kanala oz. verjetnosti servisiranja sta zadolžena CPR registra (angl. Channel Priority Registers):

Bit 1	**Bit 2**	**Prioriteta servisiranja**
0	0	kanal zapahnjen (angl. disabled) - zač. stanje
0	1	nizka
1	0	srednja
1	1	visoka

7.1.2 Parametrski RAM TPU-ja

Po splošnem definiranju načina delovanja TPU-ja in izbiri časovnih funkcij na posameznih kanalih sledi določanje parametrov, ki jih zahtevajo funkcije na izbranih kanalih. V ta namen je v TPU RAM-u rezervirano 256 bytov, ki so razdeljeni na šestnajst skupin (za TPU0 - TPU15) po osem besed (za osem različnih parametrov - slika 7. 6). Tako bo npr. prvi parameter funkcije, ki smo jo dodelili TPU8, na lokaciji $Yfff80_{HEX}$, drugi na $Yfff82_{HEX}$ itd. Besedi 6 in 7 se le poredkoma uporabljata in sta na voljo samo pri kanalih TPU14 in TPU15.

TPU kanal:	Registri parametrskega RAM-a							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	$Yfff00_{HEX}$	$Yfff02_{HEX}$	$Yfff04_{HEX}$	$Yfff06_{HEX}$	$Yfff08_{HEX}$	$Yfff0a_{HEX}$	-	-
1	$Yfff10_{HEX}$	$Yfff12_{HEX}$	$Yfff14_{HEX}$	$Yfff16_{HEX}$	$Yfff18_{HEX}$	$Yfff1a_{HEX}$	-	-
2	$Yfff20_{HEX}$	$Yfff22_{HEX}$	$Yfff24_{HEX}$	$Yfff26_{HEX}$	$Yfff28_{HEX}$	$Yfff2a_{HEX}$	-	-
3	$Yfff30_{HEX}$	$Yfff32_{HEX}$	$Yfff34_{HEX}$	$Yfff36_{HEX}$	$Yfff38_{HEX}$	$Yfff3a_{HEX}$	-	-
4	$Yfff40_{HEX}$	$Yfff42_{HEX}$	$Yfff44_{HEX}$	$Yfff46_{HEX}$	$Yfff48_{HEX}$	$Yfff4a_{HEX}$	-	-
5	$Yfff50_{HEX}$	$Yfff52_{HEX}$	$Yfff54_{HEX}$	$Yfff56_{HEX}$	$Yfff58_{HEX}$	$Yfff5a_{HEX}$	-	-
6	$Yfff60_{HEX}$	$Yfff62_{HEX}$	$Yfff64_{HEX}$	$Yfff66_{HEX}$	$Yfff68_{HEX}$	$Yfff6a_{HEX}$	-	-
7	$Yfff70_{HEX}$	$Yfff72_{HEX}$	$Yfff74_{HEX}$	$Yfff76_{HEX}$	$Yfff78_{HEX}$	$Yfff7a_{HEX}$	-	-
8	$Yfff80_{HEX}$	$Yfff82_{HEX}$	$Yfff84_{HEX}$	$Yfff86_{HEX}$	$Yfff88_{HEX}$	$Yfff8a_{HEX}$	-	-
9	$Yfff90_{HEX}$	$Yfff92_{HEX}$	$Yfff94_{HEX}$	$Yfff96_{HEX}$	$Yfff98_{HEX}$	$Yfff9a_{HEX}$	-	-
10	$Yfffa0_{HEX}$	$Yfffa2_{HEX}$	$Yfffa4_{HEX}$	$Yfffa6_{HEX}$	$Yfffa8_{HEX}$	$Yfffaa_{HEX}$	-	-
11	$Yfffb0_{HEX}$	$Yfffb2_{HEX}$	$Yfffb4_{HEX}$	$Yfffb6_{HEX}$	$Yfffb8_{HEX}$	$Yfffba_{HEX}$	-	-
12	$Yfffc0_{HEX}$	$Yfffc2_{HEX}$	$Yfffc4_{HEX}$	$Yfffc6_{HEX}$	$Yfffc8_{HEX}$	$Yfffca_{HEX}$	-	-
13	$Yfffd0_{HEX}$	$Yfffd2_{HEX}$	$Yfffd4_{HEX}$	$Yfffd6_{HEX}$	$Yfffd8_{HEX}$	$Yfffda_{HEX}$	-	-
14	$Yfffe0_{HEX}$	$Yfffe2_{HEX}$	$Yfffe4_{HEX}$	$Yfffe6_{HEX}$	$Yfffe8_{HEX}$	$Yfffea_{HEX}$	$Yfffec_{HEX}$	$Yfffee_{HEX}$
15	$Yffff0_{HEX}$	$Yffff2_{HEX}$	$Yffff4_{HEX}$	$Yffff6_{HEX}$	$Yffff8_{HEX}$	$Yffffa_{HEX}$	$Yffffc_{HEX}$	$Yfffe_{HEX}$

Slika 7. 6: Parametrski RAM TPU-ja

7.2 Primer uporabe PWM funkcije

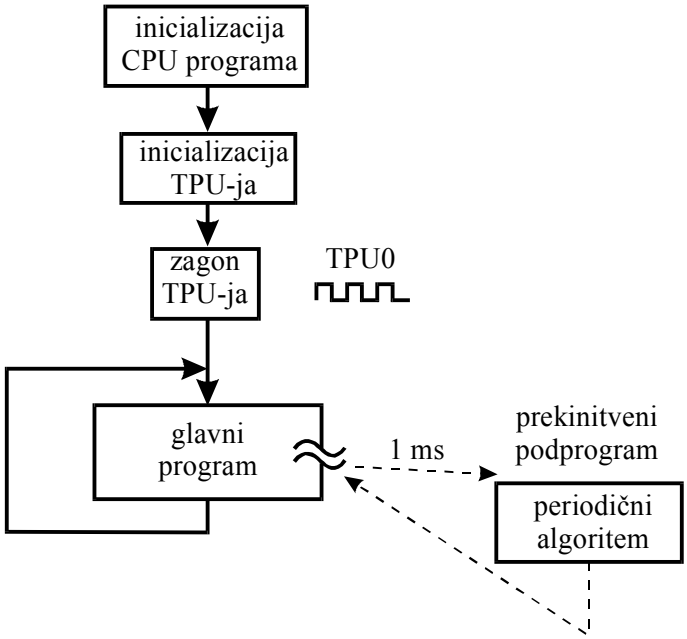
7.2.1 Princip delovanja programa

Opis konfiguriranja TPU-ja, izbire vseh funkcij in definiranja potrebnih parametrov zahteva preveč prostora, zato bomo tukaj opisali karakteristični zgled uporabe funkcije PWM (za ostale funkcije glej [26]).

Naloga, ki jo želimo realizirati, je splošno konfiguriranje TPU-ja ter definiranje in zagon funkcije PWM na kanalu TPU0, ki bo generirala neskončni vlak pulzov s periodo 1 ms. Po zagonu delujeta CPU in TPU neodvisno. Po preteku vsake periode naj TPU prekine delovanje CPU, ki bo takrat izvedel prekinitveni podprogram (njegovo trajanje je krajše od 1 ms!). Po vrnitvi iz podprograma naj CPU nadaljuje s svojim dotedanjim delom do naslednje prekinitve.

Očitno bomo funkcijo PWM v tem kontekstu uporabili za generiranje periodične prekinitve v trajanju 1 ms. Na ta način lahko zagotovimo periodičnost izvajanja prekinitvenega podprograma s časom vzorčenja T_v = 1 ms (glej tudi pogl. 10). Slika 7. 7 kaže posamezne korake, ki jih je treba izvršiti v CPU programu:

1. Inicializacija splošnih spremenljivk, ki jih uporabljamo v programu
2. Inicializacija TPU registrov, definiranje PWM funkcije na kanalu TPU0 ter njena parametrizacija; omogočanje TPU prekinitve.
3. Zagon TCR1 in PWM funkcije s periodo 1 ms.
4. Izvajanje glavnega CPU programa (manj zahtevne funkcije, npr. komunikacija z uporabnikom). Čakanje na prekinitev. Čas izvajanja podprograma mora biti krajši od 1 ms.
5. Prekinitveni podprogram (v bistvu “glavni” program), ki se izvaja periodično s T_V = 1 ms.



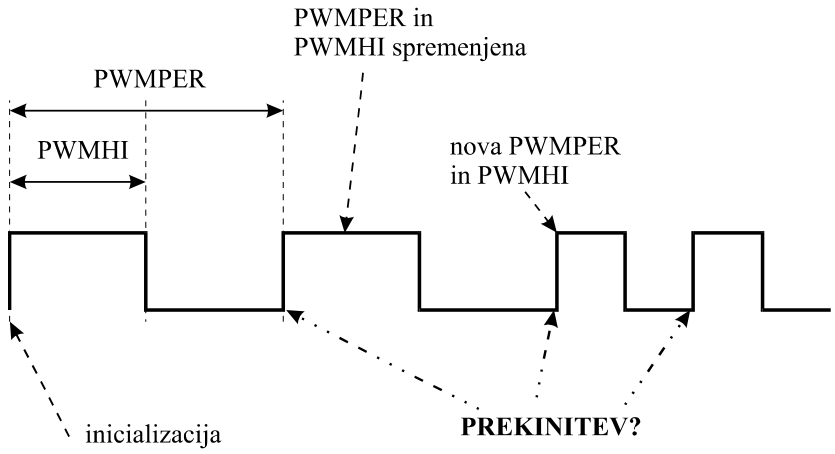
Slika 7. 7: Diagram poteka generiranja periodičnega vzorčenja s pomočjo PWM

7.2.2 Funkcija PWM s periodo 1 ms

PWM funkcija generira vlak pulzov (0 V, 5 V) na izhodu TPU kanala, (načeloma) spremenljive periode. Osnovna parametra funkcije sta na sliki 7. 8 označena s simboloma PWMPER in PWMHI in določata trajanje periode ter trajanje stanja 1 (5 V). Oba parametra podajamo v obliki mnogokratnikov periode osnovnega takta (TCR1 ali TCR2), med obratovanjem pa ju lahko spreminjamo[3]. Pri vsakem prehodu 0 $\rightarrow$ 1 lahko kanal prekine izvajanje programa v CPU.

V našem primeru bosta PWMPER in PWMHI konstantna, ker pa bo namen te funkcije le generiranje periodične prekinitve, vrednost PWMHI nima nobenega pomena (vsekakor pa mora veljati PWMPER > PWMHI).

[3] Od tod tudi naziv PWM - pulzno-širinska modulacija. Možna aplikacija je generiranje spreminjajoče se napetosti ob konstantnem PWMPER in spreminjajočem se PWMHI. Na binarnem izhodu bomo po filtriranju z nizkopasovnim filtrom dobili "zvezno" napetost.



Slika 7. 8: Definiranje dolžine pulzov v funkciji PWM

7.2.2.1 Konfiguriranje in parametriranje funkcije PWM

Po izbiri časovne funkcije v registru CFSR (za PWM je to koda 9_{HEX}, glej pogl. 7.1.1.5) moramo izbrati še nekatere druge, za to funkcijo značilne konfiguracijske nastavitve (v HSQR in HSRR registrih, pogl. 7.1.1.6 in 7.1.1.7) in parametre (v parametrskem RAM-u za posamezne kanale).

V primeru PWM register HSQR nima nobenega pomena, bita v HSRR0 ali HSRR1 za posamezne kanale pa ponujata naslednje opcije (glej tudi sliko 7. 5):

00 brez servisiranja ali servisiranje končano (npr. inicializacija)
01 takojšnje ažuriranje PWM
10 inicializacija.

Parametrski RAM posameznega kanala za PWM funkcijo kaže slika 7. 9.

naslovi (HEX)	**mnemonik**	15 9	8 0
YfffW0[4]	TPUW_0		CHANNEL_CONTROL
YfffW2	TPUW_2	OLDRIS	
YfffW4	TPUW_4	PWMHI	
YfffW6	TPUW_6	PWMPER	
YfffW8	TPUW_8	PWMRIS	

Slika 7. 9: Lokacije v parametrskem RAM posameznega kanala za funkcijo PWM

[4] W je oznaka kanala, na katerega se nanaša funkcija, npr. prvi parameter kanala 5 se nahaja na TPU RAM lokaciji $Yfff50_{HEX}$, drugi na $Yfff52_{HEX}$ itd.

Parametra PWMHI in PWMPER že poznamo. Oba sta 16-bitna (največja vrednost 65535) mnogokratnika TCR1 ali TCR2. OLDRIS je čas, pri katerem je prišlo do zadnjega prehoda 0 → 1, PWMRIS pa trenutek nastopa naslednjega prehoda (OLDRIS + PWMPER), ki se izračuna na začetku vsakega pulza.

Parameter CHANNEL_CONTROL je 9-bitno polje (slika 7. 10), v katerem določimo, katero izmed dveh ur (TCR1 ali TCR2) bomo upoštevali pri podajanju PWMPER in PWMHI, ter začetno stanje izhoda po inicializaciji (polje PSC).

TBS				**PAC**			**PSC**		**Dejavnost**	
8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Vhod**	**Izhod**
							0	0	-	Postavi nožico kot določa PAC
							0	1	-	Postavi nožico na ″1″
							1	0	-	Postavi nožico na ″0″
							1	1	-	Ne spreminjaj stanja
				1	x	x			Ne spreminjaj PAC	Ne spreminjaj PAC
0	**1**	x	x						-	**Izhodni kanal**
0	1	0	0						-	Zajemanje TCR1, primerjava TCR1
0	1	1	1						-	Zajemanje TCR2, primerjava TCR2
1	1	x	x						Ne spreminjaj TBS	Ne spreminjaj TBS

x je poljubna vrednost

Slika 7. 10: Pomen bitov v parametru CHANNEL_CONTROL funkcije PWM

7.2.2.2 Inicializacija TPU-ja

O programu za inicializacijo TPU parametrov, napisanem v C jeziku, bo več govora v nadaljevanju (glej tudi sliko 7. 11). Na tem mestu si oglejmo nastavitve posameznih registrov. Pri delu bomo uporabljali njihove standardne simbole oz. kratice, ki jim moramo na začetku programa dodeliti ustrezne pomnilniške lokacije (glej tudi sliki 7. 2 in 7. 6).

TPUMCR

TPUMCR smo postavili v stanje 6041_{HEX} = 0110 0000 0100 0001_{BIN}. Na ta način smo definirali periodo ure TCR1, ki bo približno enaka 2 μs (pogl. 7.1.1.1)[5].

CFSR3

Vsebina registra CFSR3 je 9_{HEX} = 0000 0000 0000 1001_{BIN}, s tem smo pa definirali funkcijo PWM na kanalu TPU0 (vsi ostali kanali so neaktivni; glej tudi sliko 7. 5).

CPR1

[5] V praksi se ta podatek pogostokrat nekoliko razlikuje od deklariranega, zato je treba v primerih ko želimo zelo natančno določiti periodo, opraviti ustrezne meritve in korigiranje nastavitve.

(CPR1) = 3_{HEX} = 0000 0000 0000 0011_{BIN}. Kanal TPU0 ima najvišjo prioriteto.

CIER

(CIER) = 1_{HEX} = 0000 0000 0000 0001_{BIN}. S tem smo kanalu TPU0 omogočili generiranje prekinitve (za PWM funkcijo je to ob vsakokratnem prehodu z 1 na 0).

TICR

(TICR) = 0640_{HEX} = 0000 0110 0100 0000_{BIN}. V podpoglavju 7.1.1.2 o registru za konfiguracijo prekinitev smo omenili njegovo dvojno vlogo: določanje nivoja zahteve po prekinitvi in baznega prekinitvenega vektorja. Navedena vsebina registra določa drugo po vrsti prekinitveno prioriteto (biti 8,9,10 = 110_{BIN} = 6; najvišja je 7). Bazni prekinitveni vektor je v tem primeru 4, kar pomeni, da je zaporedna številka prekinitveni vektor za TPU0 40_{HEX}, za TPU1 41_{HEX}, za TPU15 pa $4f_{HEX}$. Konkretne lokacije vektorjev za posamezne kanale v RAM pomnilniku izračunamo z znano formulo (glej tudi proceduro `set_handler` na sliki 7. 11 ter poglavje o TICR registru):

(VBR) + 4 · 40_{HEX} = (VBR) + 4 · 64_{DEC} za TPU0,
(VBR) + 4 · 41_{HEX} = (VBR) + 4 · 65_{DEC} za TPU1,
(VBR) + 4 · $4f_{HEX}$ = (VBR) + 4 · 79_{DEC} za TPU15 itd.

Tako definirani vektorji TPU-ja zasedejo naslednje lokacije v vektorski mapi:

Št. vektorja	**Premik (offset)**		**Opis**
	dec	hex	
0	0	000	Reset: SP
1	4	004	Reset: PC
.....	.....	.....	.....
64	256	100	TPU kanal 0
65	260	104	TPU kanal 1
66	264	108	TPU kanal 2
67	268	10C	TPU kanal 3
68	272	110	TPU kanal 4
69	276	114	TPU kanal 5
70	280	118	TPU kanal 6
71	284	11C	TPU kanal 7
72	288	120	TPU kanal 8
73	292	124	TPU kanal 9
74	296	128	TPU kanal 10
75	300	12C	TPU kanal 11
76	304	130	TPU kanal 12
77	308	134	TPU kanal 13
78	312	138	TPU kanal 14
79	316	13C	TPU kanal 15

TPU0_0

Preden končamo z inicializacijo, moramo določiti še parametre kanala (tukaj TPU0) za izbrano funkcijo (glej sliki 7. 9 in 7. 10). V konkretnem primeru je TPU0_0 = 0091_{HEX} = 0000 0000 1001 0001_{BIN}, torej smo izbrali TCR1 ter začetno stanje kanala TPU0 1.

TPU0_4 in TPU_6

TPU0_4 in TPU_6, torej PWMHI in PWMPER postavimo v stanje 524_{DEC} in 250_{DEC}. To sta mnogokratnika periode izbranega TCR1 (≈2 μs; glej register TPUMCR). Trajanje PWM periode bo potemtakem 1 ms, signala 1 pa približno 0,5 ms (vrednost 524 in ne 500 je izbrana po natančni meritvi trajanja periode TCR1).

HSRR1

TPU kanal 0 je konfiguriran in parametriran. Sedaj preostaja le inicializacija kanala, ki jo sproži CPU s postavitvijo HSRR1 v stanje 2 = 0000 0000 0000 0010_{BIN} (glej sliko 7. 5). Postopek zahteva določeno število urinih ciklov, edino zagotovilo o končani proceduri pa je samodejno postavljanje teh bitov v stanje 00_{BIN}. Zaradi tega je v program vstavljena pogojna zanka, ki onemogoča nadaljevanje programa preden je pogoj (HSRR1) = 0 izpolnjen.

7.2.2.3 Glavni in prekinitveni program

Po končani inicializacijski proceduri oz. zagonu TPU-ja, nadaljuje CPU z obdelovanjem "glavnega programa" do nastopa časovne prekinitve. V našem zgledu je glavni program le neskončna zanka (v proceduri `main`). Pri prekinitvi pride do skoka v prekinitveni podprogram (`handler`). Načeloma je v njem dejanski regulacijski program, v tem primeru pa le resetiramo prekinitveni statusni bit kanala TPU0 in s tem omogočimo ponovno prekinitev. Ponovimo še enkrat, da mora celotno trajanje tega podprograma biti krajše od časa med zaporednima prekinitvama, drugače lahko pride do nenadzorovanega dogajanja (glej tudi zgled v pogl. 10).

7.2.3 C program za generiranje periodične prekinitve

Delovanje programa smo v grobem že opisali. Čeprav so funkcije relativno transparentne, omenimo tukaj le nekatere konvencije C jezika, ki jih v programu uporabljamo:

- Prefiks `0x` določa šestnajstiško notacijo števila.
- Komentar se nahaja znotraj oznak `/*` in `*/` .
- *Kazalci* (angl. pointers; simbol *) so zelo eleganten način naslavljanja spominskih lokacij. Velja: če je `TPUMCR` naslov neke lokacije (tukaj `0xfffffe00`) je `*TPUMCR` vsebina te lokacije. Večina deklaracij `#define` v programu dodeljuje simbole naslovom spominskih lokacij (npr. `TPUMCR`). Kazalci na te naslove (`*TPUMCR`) imajo dostop le do formata, ki je vsebovan v deklaraciji (za TPUMCR je to t.i. `unsigned short int` oz. 16-bitni celoštevilski format brez predznaka).

- V C-ju pogostokrat uporabljana skrajšana oblika ukaza, npr. `*CISR &= 0xfffe` (velja tudi za ostale podobne logične in aritmetične operacije), je enaka bolj znani sintaksi `*CISR = *CISR & 0xfffe.`
- Prvi del programa je le definiranje simbolov, najprej simboličnih lokacij, potem pa konstant.
- Temu sledijo deklaracije podprogramov: `init` (inicializacija TPU-ja), `set_handler` (definiranje prekinitvenega vektorja za TPU0), `handler` (prekinitveni program; angl. "handler" - upravljač - je običajen naziv za prekinitvene podprograme) in `main` (glavni program). V tem delu so eventualni vhodni parametri (npr. `set_handler(handler,vecno)`) podani le simbolično. Šele pri klicu posamezne rutine jim dodelimo dejanske vrednosti.
- Če smo v C-ju definirali proceduro npr. `handler()`, je spremenljivka `handler` (brez oklepajev!) naslov, na katerem se ta procedura nahaja.
- `set_handler` je procedura za postavljanje naslova prekinitvenega programa (`handler`) na prekinitveni vektor z zaporedno številko $40_{HEX} = 64_{DEC}$. Oba podatka sta vhodna parametra pri klicu te procedure: `set_handler(handler,64).`
- Maskiranje: pogostokrat želimo določene bite v registru ali spominski lokaciji postaviti v stanje 1 ali 0, ne glede na stanje ostalih bitov (npr. glej CISR register v programu). To dosežemo s t.i. *"maskiranjem"*: če gre za postavitev bitov v stanje 0, naredimo to z operacijo "logični IN" (angl. AND, simbol `&`), v stanje 1 pa z operacijo "logični ALI" (angl. OR, simbol `|`). V primeru iz programa (`*CISR &= 0xfffe` oz. zaradi boljšega pregleda: 1111 1111 1111 1110_{BIN}) se v stanje 0 postavi le bit 0, vsem ostalim pa se vsebina ne spremeni. Če bi hoteli npr. le bit 3 gotovo postaviti v stanje 1, bi to lahko storili z ukazom `*CISR |= 0x0008` (0000 0000 0000 1000_{BIN}). **PAZI:** selektivnega postavljanja bitov z običajnim dodeljevanjem (simbol =) ne moremo doseči, ker s tem nujno postavimo vse bite v določeno stanje!
- Ukaz ali ukazi (v olepajih {}) v zanki `while(x)` se izvajajo toliko časa, dokler je izpolnjen pogoj `x`. Ukaz za zanko ne zaključujemo s podpičjem "`;`" kot navadne ukaze. C pozna tudi "prazne" ukaze, ki se ne izvršujejo, kot vsak pravi ukaz pa se morajo končati s podpičjem. Npr. `;;;` so trije takšni ukazi. V našem primeru imamo opravka z zanko "`while(1) ;`", torej v neskončni zanki ne izvaja nič, razen ponovnega preverjanja izpolnjenosti pogoja.

```
/* vljucevanje C knjiznic */

#include <stdio.h>
#include <stdlib.h>

#define TPUMCR (unsigned short int*)  0xffffe00  /* module config. reg.
*/
#define TICR (unsigned short int*) 0xffffe08  /* interrupt conf. reg.  */
#define CIER (unsigned short int*)  0xffffe0a  /* interrupt enable reg.
*/
#define CISR (unsigned short int*)  0xffffe20  /* interrupt status re.
*/
#define CFSR3(unsigned short int*)  0xffffe12 /* chann. fn. sel. reg.  */
#define HSRR1(volatile unsigned short int*)0xffffe1a      /* host service
                                                          req.reg. */
#define CPR1   (unsigned short int*) 0xffffe1e  /* chann. prior. reg. */
#define TPU0_0 (unsigned short int*) 0xffffff00  /* channel control  */
#define TPU0_4 (unsigned short int*) 0xffffff04  /* naslov PWMHI */
#define TPU0_6 (unsigned short int*) 0xffffff06  /* naslov PWMPER */

#define PWMPER  524             /* 524 x 1,9 us = 1 ms */
```

```
#define PWMHI   250


/*****************************************************************/
/*          INICIALIZACIJA TPU-ja                                */
/*****************************************************************/

void init()
{
      *TPUMCR=0x6041;
      *CFSR3=0x0009;    /* izbrana funkcija */
      *CPR1=0x0003;     /* nastavitev prioritet */
      *CIER=0x0001;     /* omogoca prekinitev na TP0 */
      *TICR=0x640;      /*nastavitev nonmaskable prekintive, lokacija
                          prekinitvenega vektorja*/
      *TPU0_0=0x0091;   /* nastavitev channel control */
      *TPU0_6=PWMPER;   /* trajanje periode PWMPER */
      *TPU0_4=PWMHI;    /* trajanje PWMHI */

      *HSRR1=0x0002;    /* inicializacija TPU; izvaja PWM */

        while (*HSRR1 != 0x0000);   /* cakamo do konca inicializacije
                                       takrat je *HSRR1=0x0000 */
      *CISR &= 0xfffe;

}/* konec init */


/*****************************************************************/
/*    P0DPR0GRAM ZA ZAPIS ZA□ETNEGA NASL0VA PREKINITVENEGA       */
/*    P0DPR0GRAMA (SPREM handler JE P0INTER NA P0DPR0GRAM        */
/*    handler()) NA L0KACIJ0 Z NASL0V0M (VBR)+4*vecno            */
/*****************************************************************/

void   set_handler(handler,vecno)        /*handler   je   definirani   prek.
podprogram,
                                    vecno=64*/
      void(*handler);
      int vecno;
{

             *((void(**) () ) (4*vecno))=handler;

}/* konec set_handler */


/*****************************************************************/
/*          PREKINITVENI P0DPR0GRAM                              */
/*****************************************************************/

_SWI void handler()
{

/* namesto tega komentarja pride "glavni" regulacijski program */


  *CISR &= 0xfffe;              /* omogocanje ponovne prekinitve */

}/* konec handler */


/*****************************************************************/
/*          GLAVNI PR0GRAM                                       */
/*****************************************************************/

main()
```

```
{
      set_handler(handler,64); /* definicija vektorja za int. handler */
      init();                  /* klic funkcije za inicializacijo TPU-ja */
      while(1);                /* neskon□na zanka; pogoj vedno izpolnjen
*/
}/* konec main*/
```

Slika 7. 11: Program v C-ju za generiranje periodične prekinitve s pomočjo TPU-ja