Vežba 10 : Stablo uređaja (*device tree*) i drajveri za platformske uređaje (*platform device drivers*)

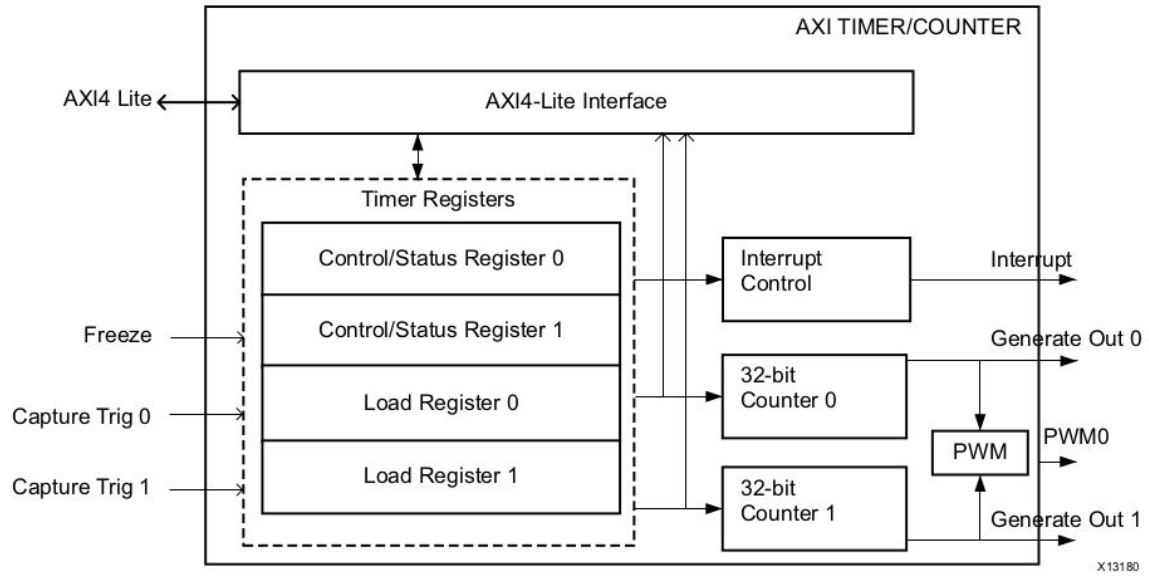
1. Uvod

Na ovim vežbama će se nastaviti rad sa drajverima za platformske uređaje. Budući da se *gpio* uređajem koji je kontrolisao LED može upravljati pomoću samo jednog registra, on nije najbolji primer tipičnih IP jezgara koji se sreću u savremenom sistemu na čipu (*SoC*). Za bolju demonstraciju drajvera za platformske uređaje, na ovim vežbama će se uzeti **Xilinx AXI Timer/Counter IP**. Posebna pažnja će biti posvećena rukovanju hardverskih prekida (*interrupt*) koje ovaj brojač poseduje, dok će većina ostalih drajverskih konstrukata biti ista kao kod LED uređaja koji je obrađen na prošlim vežbama.

Kako bi inženjer napisao drajver za uređaj, on mora biti u potpunosti upoznat sa načinom rukovanja tim uređajem kao i svim funkcionalnostima koje on poseduje. Stoga, pre nego što krenemo sa pisanjem drajvera, preporučeno je pročitati dokumentaciju (*data sheet*) koju obezbeđuje dizajner IP jezgra. Dokumentacija za AXI Timer/Counter se može preuzeti sa Xilinx sajta, posetom sledećeg linka: https://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v2_0/pg079-axi-timer.pdf Za nas najbitniji delovi ove dokumentacije su dati u sledećem poglavlju.

2. AXI Timer/Counter IP

AXI Timer/Counter je jezgro koje predstavlja dva 32-bitna ili jedan 64-bitni brojački modul koji komunicira sa ostatkom sistema putem AXI4-Lite interfejsa. AXI Timer je uređen kao dva identična brojačka modula kao što je prikazano na blok šemi (slika 1). Svi registri su 32-bitni, te svaki brojač ima sopstveni Load, Control/Status i Counter registar. **Control/Status** se koristi za podešavanja brojača, kao i očitavanje statusnih signala (flags). Svaki bit unutar ovog registra ima posebnu funkcionalnost. **Counter** registar čuva trenutnu vrednost brojača, koji može biti konfigurisan da broji na više ili na niže. Brojač ima više režima rada pri čemu se uloga **Load** registra menja u zavisnosti od odabranog režima. Interrupt signali brojača su dovedeni na ILI (OR) logičko kolo tako da iz IP jezgra izlazi samo jedana, univerzalna prekidna linija. PWM blok pravi impulsno širinsku modulaciju na izlaznom portu PWM0, gde se frekvencija i faktor ispune mogu proizvoljno podesiti. PWM mod koristi brojač 0 (Counter 0) za generisanje periode, a brojač 1 za generisanje impulsa.



Slika 1: Blok dijagram brojačkog jezgra

2.1. Režimi rada

Oba brojača podržavaju četiri različita režima rada:

2.1.1. Generate mod

U *Generate* modu, vrednost *Load* registra je inicijalna vrednost brojača koja se kopira u Counter registar. Brojač kreće da broji od te vrednosti na više ili na niže u zavisnosti od UDT bita u Control/Status registru. Kada brojač dosegne maksimalnu ili minimalnu vrednost (*overflow, underflow*), izlazni signal *Generate out* se postavlja na visoku vrednost za dužinu jednog takta, te se prekidni (*interrupt*) signal postavi na

visoku vrednost (*TINT* bit u *Control/Status* registru). Prekidni signal je potrebno ručno vratiti na nulu upisom jedinice u *TINT* bit. Ukoliko je *AutoReload/Hold* bit u *Control/Status* registru postavljen na 1, vrednost iz *Load* registra se ponovno prenosi u *Counter* registar, te brojač kreće da broji iznova. Ovaj mod se može koristiti ukoliko je potrebno generisati periodične prekide ili eksterne signale.

2.1.2. Capture mod

U Capture modu, brojač broji u punom opsegu 2^{32}, te u trenutku kada se postavi ulazni signal Capture, vrednost brojača *Counter* se sačuva u *Load* registar. Kada se detektuje ulazni signal, generiše se prekidni signal (*TINT* u *Control/Status* registru se postavlja na 1). *AutoReload/Hold* bit u *Control/Status* registru kontroliše da li je moguće prebrisati vrednost Load registra pre nego što se resetuje TINT signal. Ovaj mod se može koristiti ukoliko je potrebno zapamtiti vreme kada se desio neki eksterni dogadjaj.

2.1.3. PWM mod

U PWM modu, oba brojača se koriste kako bi se generisala impulsno širinska modulacija na izlaznom portu PWM0. Moguće je postaviti željenu frekvenciju i faktor ispune. Brojač 0 definiše dužinu periode, dok brojač 1 definiše dužinu impulsa.

2.1.4. Cascade mod

U kaskadnom modu, dva 32-bitna brojača se spajaju u jedan 64-bitni. Kaskadni brojač može raditi u *Generate* ili *Capture* modu. Control/Status registar brojača 0 se koristi za kontrolu kaskadnog brojača. Ovaj mod se može koristiti kada je potreban neki od prva dva moda ali nad dužim vremenskim intervalom.

2.2. Opis registara

Adresni opseg registara ovog IP jezgra je dat u tabeli na slici 2.

Address Offset	Register Name	Description
0h	TCSR0	Timer 0 Control and Status Register
04h	TLR0	Timer 0 Load Register
08h	TCR0	Timer 0 Counter Register
0Ch-0Fh	RSVD	Reserved
10h	TCSR1	Timer 1 Control and Status Register
14h	TLR1	Timer 1 Load Register
18h	TCR1	Timer 1 Counter Register
1Ch-1Fh	RSVD	Reserved

Slika 2. Registri brojačkog IP jezgra

2.2.1. *Control/Status* registri 0 (TCSR0 i TSCR1)

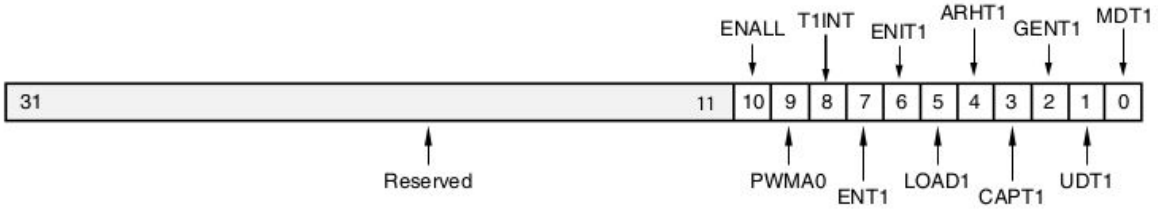
Sledeća slika prikazuje raspored i imena bitova kod *Control/Status* registra za brojač 0 (*TCSR0*).

Slika 3. Raspored bita TCSR0 registra

U sledećoj tabeli su naznačene funkcionalnosti svakog bita:

BIT	IME	Funkcija
31:12	Rezervisani	Ne koriste se
11	CASC	1 = Aktivira kaskadni mod dva brojača
10	ENALL	1 = Signal dozvole za oba brojača, postavlja ENT bite
9	PWMA0	1 = Aktivira PWM mod brojača 0
8	T0INT	Čitanje : 1 = Prekid se desio 0 = Prekid se nije desio Upis: 1 = Postavi na nula (izbriši prekid) 0 = Ništa se ne desi
7	ENT0	1 = Signal dozvole (*enable*) za brojač 0
6	ENIT0	1 = Signal dozvole (*enable*) za prekid brojača 0
5	LOAD0	1 = Prebaci vrednost iz *Load* registra u *Counter* registar
4	ARHT0	1 = Automatsko upisivanje vrednosti Load registra u Counter registar kada se desi *overflow*/*underflow*
3	CAPT0	1 = Signal dozvole (*enable*) za eksterni *capture* signal
2	GENT0	1 = Signal dozvole (*enable*) za eksterni *generate* signal
1	UDT0	1 = Brojanje na više; 0 = Brojanje na niže
0	MDT0	1 = *Generate* mod 0 = *Capture* mod

TCSR 1 ima iste funkcionalnosti bita kao i TCSR0 registar. Jedina razlika je što se u kaskadnom modu ovaj registar koristi samo za upisivanje u TLR1 registar. Raspored bita se može videti na sledećoj slici:



2.2.2. *Load* registri (*TLR0* i *TLR1*)

Load registri sadrže 32-bitnu vrednost koja se menja u zavisnosti od režima rada. U generate modu je to inicijalna vrednost od koje brojač kreće da broji. U capture modu ona čuva vrednost koja je bila u *Counter* registru kada se desio eksterni signal *capture*. U kaskadnom modu, TLR0 sadrži niža 32 bita dok TLR1 sadrži viša 32 bita.

2.2.3. *Timer/Counter* registri (*TCR0* i *TCR1*)

Timer/Counter su 32-bitni registri koji čuvaju trenutnu vrednost brojača. U kaskadnom modu TCR0 sadrži niža 32 bita dok TCR1 sadrži viša 32 bita.

3. Primer drajvera za brojač

Drajver koji će se razmatrati u ovom poglavlju je demonstrativni primer, te ne odražava glavna pravila pisanja drajvera za IP jezgra. Naime, ukoliko bi hteli da napravimo korektan drajver za prethodno opisani brojač, morali bismo da implementiramo sve funkcionalnosti koje brojač podržava. U tom slučaju bi korisnik slao komande za konfigurisanje uređaja, a drajver bi translirao te komande na manipulisanje registrima.

Budući da je jedini cilj ovih vežbi da se demonstrira opsluživanje prekida, u ovom drajveru će se uređaj ograničiti na samo jedan režim rada - ***generate***. Korisnik će moći da zada broj prekida i vremenski interval između njih u milisekundama. Na primer: **echo "3,1000" > /dev/timer** će pokrenuti brojač 0 u *generate* modu na takav način da se svake sekunde desi prekid, te da se brojač 0 zaustavi nakon generisana 3 prekidna signala.

Kako bi kod drajvera bio pregledniji, pomoću makroa su definisane maske za manipulisanje bitima TCSR0 registra:

```
#define XIL_AXI_TIMER_CSR_CASC_MASK 0x00000800
#define XIL_AXI_TIMER_CSR_ENABLE_ALL_MASK 0x00000400
#define XIL_AXI_TIMER_CSR_ENABLE_PWM_MASK 0x00000200
#define XIL_AXI_TIMER_CSR_INT_OCCURED_MASK 0x00000100
#define XIL_AXI_TIMER_CSR_ENABLE_TMR_MASK 0x00000080
#define XIL_AXI_TIMER_CSR_ENABLE_INT_MASK 0x00000040
#define XIL_AXI_TIMER_CSR_LOAD_MASK 0x00000020
#define XIL_AXI_TIMER_CSR_AUTO_RELOAD_MASK 0x00000010
#define XIL_AXI_TIMER_CSR_EXT_CAPTURE_MASK 0x00000008
#define XIL_AXI_TIMER_CSR_EXT_GENERATE_MASK 0x00000004
#define XIL_AXI_TIMER_CSR_DOWN_COUNT_MASK 0x00000002
#define XIL_AXI_TIMER_CSR_CAPTURE_MODE_MASK 0x00000001
```

Takođe su definisani ofseti za pristup registrima:

```
#define XIL_AXI_TIMER_TCSR_OFFSET 0x0
#define XIL_AXI_TIMER_TLR_OFFSET    0x4
#define XIL_AXI_TIMER_TCR_OFFSET    0x8
```

Kao i u primeru drajvera za LED, definišemo globalnu strukturu *timer_info* u kojoj čuvamo početnu i krajnju fizičku i početnu virtuelnu adresu uređaja. Za razliku od strukture sa prethodnih vežbi, ova ima dodatno polje *irq_num* koje će čuvati broj prekidne linije koja je dodeljena uređaju.

```
struct timer_info {
	unsigned long mem_start;
	unsigned long mem_end;
	void __iomem *base_addr;
	int irq_num;
};
```

Sem ove globalne promenljive, definisane su još dve: *i_num* i *i_cnt*. Promenljiva *i_num* će čuvati broj prekida koji se moraju izvršiti, dok će promenljiva *i_cnt* indicirati koliko se prekida desilo do sada.

3.1. *Probe* funkcija

Prvi deo probe funkcije se ni po čemu ne razlikuje od primera drajvera za LED. Prvo se iz strukture *platform_device* izdvajaju fizičke adrese brojačkog modula, te se one smeštaju u pomoćnu strukturu *timer_info*. Pre nego što se adrese smeste u polja *start* i *end,* pomoću funkcije *kmalloc* se zauzima memorija za pomoćnu strukturu. Zatim se dati memorijski opseg rezerviše pozivom funkcije ***request_mem_region****.* Za kraj se taj opseg fizičkih adresa virtualizuje pozivom funkcije **ioremap** i smešta u polje *base* strukture *timer_info.*

```
static int timer_probe(struct platform_device *pdev)
{
	struct resource *r_mem;
	int rc = 0;
	// Get physical register adress space from device tree
	r_mem = platform_get_resource(pdev, IORESOURCE_MEM, 0);
	if (!r_mem) {
		printk(KERN_ALERT "xilaxitimer_probe: Failed to get reg
resource\n");
		return -ENODEV;
	}
	// Get memory for structure timer_info
	tp = (struct timer_info *) kmalloc(sizeof(struct timer_info),
GFP_KERNEL);
	if (!tp) {
		printk(KERN_ALERT "xilaxitimer_probe: Could not allocate
timer device\n");
		return -ENOMEM;
	}
	// Put physical adresses in timer_info structure
	tp->mem_start = r_mem->start;
	tp->mem_end = r_mem->end;
```

```
      // Reserve that memory space for this driver
      if (!request_mem_region(tp->mem_start,tp->mem_end - tp->mem_start
+ 1,  DEVICE_NAME))
      {
            printk(KERN_ALERT "xilaxitimer_probe: Could not lock memory
region at %p\n",(void *)tp->mem_start);
            rc = -EBUSY;
            goto error1;
      }
      // Remap phisical to virtual adresses
      tp->base_addr = ioremap(tp->mem_start, tp->mem_end - tp->mem_start
+ 1);
      if (!tp->base_addr) {
            printk(KERN_ALERT "xilaxitimer_probe: Could not allocate
memory\n");
            rc = -EIO;
            goto error2;
      }
...
```

Za razliku od LED, brojački modul koristi prekide, te je potrebno imati broj prekidne linije preko koje šalje signal procesoru. Broj prekidne linije se može izdvojiti iz strukture *platform_device* na sličan način na koji su preuzete adrese registara, s tim što se sada poziva funkcija ***platform_get_irq***.

```
...
// Get interrupt number from device tree
      tp->irq_num = platform_get_irq(pdev, 0);
      if (!tp->irq_num) {
            printk(KERN_ALERT "xilaxitimer_probe: Failed to get irq
resource\n");
            rc = -ENODEV;
            goto error2;
      }
...
```

Poslednji korak u probe funkciji je da se rezerviše ova prekidna linija i da se ona poveže sa funkcijom koja će opsluživati prekid (*interrupt handler, interrupt service routine*). U tu svrhu postoji funkcija ***request_irq*** koja je zajedno sa pratećim tipovima definsana u <linux/interrupt.h>. Parametar *irq* je broj prekidne linije koja se registruje. Parametar *irq_handler_t* je pokazivač na funkciju koja će se pozvati svaki put kada se desi prekid. Parametar *irqflags* definiše dodatne flegove koji opisuju prekid, *devname* je ime uređaja i dev_id je dodatna informacija koja se prosleđuje

prekidnoj rutini kao preko parametra. Ova informacija se često postavlja na baznu adresu uređaja kako bi u prekidnoj rutini bilo moguće komunicirati sa registrima uređaja. U našem slučaju je bazna adresa uređaja globalna informacija (*timer_info->base*), te će svakako biti vidljiva u prekidnoj rutini.

```
int request_irq (unsigned int  irq,
                 irq_handler_t  handler,
                 unsigned long  irqflags,
                 const char *  devname,
                 void *  dev_id);
```

Komplementarna funkcija je **free_irq** koja se poziva u remove funkciji.

```
const void * free_irq(unsigned int irq, void * dev_id)
```

Ostatak probe funkcije je povezivanje prekida sa prekidnom rutinom *xilaxitimer_isr:*

```
...
// Reserve interrupt number for this driver
      if (request_irq(tp->irq_num, xilaxitimer_isr, 0, DEVICE_NAME,
NULL)) {
            printk(KERN_ERR "xilaxitimer_probe:
                        Cannot register IRQ %d\n", tp->irq_num);
            rc = -EIO;
            goto error3;
      }
      else
      {
            printk(KERN_INFO "xilaxitimer_probe:
                           Registered IRQ %d\n", tp->irq_num);
      }
      printk(KERN_NOTICE "xilaxitimer_probe: Timer platform driver
registered\n");
      return 0;//ALL OK
```

3.2. Prekidna rutina *(ISR, interrupt service routine)*

U probe funkciji je prekid povezan sa prekidnom rutinom pod imenom ***xilaxitimer_isr***. Definicija ove funkcije se može videti u kodu u nastavku. U prethodnom poglavlju je napomenuto da kada se desi prekid, potrebno je ručno resetovati bit *T0INT* u registru *TCSR0* kako bi se prekid mogao desiti ponovo. Ovo je prva stvar što se uradi u prekidnoj rutini. Čita se registar *TCSR0*, i smešta se njegova vrednost u pomoćnu promenljivu *data*. Resetovanje bita se vrši upisom jedinice na to mesto u registru. Poziva se *iowrite32* funkcija gde se nova vrednost dobija logičkom operacijom "ILI" pročitane vrednosti i maske koju smo definisali u makrou, kako bi vrednosti ostalih bita u registru ostale iste.

Neposredno nakon toga se uveća vrednost *i_cnt* koja predstavlja broj prekida koji su se desili do sada. Ukoliko je vrednost *i_cnt* jednaka *i_num*, to znači da se desio poslednji prekid, te da je potrebno zaustaviti brojač. Ovo se radi u "if" naredbi spuštanjem bita dozvole *ENT0* u kontrolnom registru *TCSR0*.

```
static irqreturn_t xilaxitimer_isr(int irq,void*dev_id)
{
	unsigned int data = 0;
	printk(KERN_INFO "xilaxitimer_isr: Interrupt %d occurred
!\n",i_cnt);
	// Clear Interrupt
	data = ioread32(tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
	iowrite32(data | XIL_AXI_TIMER_CSR_INT_OCCURED_MASK,
			tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
	// Increment number of interrupts that have occured
	i_cnt++;
	// Disable Timer after i_num interrupts
	if (i_cnt>=i_num)
	{
		printk(KERN_NOTICE "xilaxitimer_isr: All of the interrupts
have occurred. Disabling timer\n");
		data = ioread32(tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
		iowrite32(data & ~(XIL_AXI_TIMER_CSR_ENABLE_TMR_MASK),
tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
		i_cnt = 0;
	}
	return IRQ_HANDLED;
}
```

3.3. Funkija za inicijalizaciju brojača

Kako bi brojač započeo brojanje u *generate* modu sa željenom periodom, potrebno je pravilno inicijalizovati uređaj. Dizajner IP jezgra je u dokumentaciji naznačio korake koji se moraju odraditi kako bi brojač radio na korektan način. Mi ćemo ispoštovati korake u pomoćnoj funkciji **setup_and_start_timer** koja će kao jedini parametar primati dužinu periode između prekida. Na početku funkcije se proračunava koja se vrednost treba upisati u *TCLR0* (*load*) registar. Ako se podsetimo generate režima rada, brojač kreće da broji od vrednosti load registra, te generiše prekid kada se desi prekoračenje (*overflow*). Kako naš sistem radi na 100MHz, to znači da se vrednost brojača inkrementira svakih 10 ns. Kako bi prošla jedna milisekunda potrebno je da prođe 100000 perioda taktnog signala. Dakle naš brojač treba da odbroji `milliseconds*100000` puta. Početna vrednost load registra se zatim dobija oduzimanjem ovog izraza od nule (kada se dešava prekoračenje).

```
static void setup_and_start_timer(unsigned int milliseconds)
{
	// Disable Timer Counter
	unsigned int timer_load;
	unsigned int zero = 0;
	unsigned int data = 0;
	timer_load = zero - milliseconds*100000;
	...
```

Koraci za konfigurisanje, opisani u uputstvu za AXI TImer/Counter IP:

1. Bit dozvole *ENT0* postaviti na nula dok se god brojač konfiguriše

```
...
	data = ioread32(tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
	iowrite32(data & ~(XIL_AXI_TIMER_CSR_ENABLE_TMR_MASK),
			tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
...
```

2. Postaviti vrednost *TCLR0* (load) registra

```
...
iowrite32(timer_load, tp->base_addr + XIL_AXI_TIMER_TLR_OFFSET);
...
```

3. Postaviti bit *LOAD0* na jedinicu kako bi se vrednost iz load registra *TLR0* upisala u brojački registar *TCR0*. Neposredno nakon toga vratiti bit *LOAD0* na nulu kako bi brojač mogao da radi.

```
...
data = ioread32(tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
      iowrite32(data | XIL_AXI_TIMER_CSR_LOAD_MASK,
                  tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);

      data = ioread32(tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
      iowrite32(data & ~(XIL_AXI_TIMER_CSR_LOAD_MASK),
                  tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
...
```

4. Konfigurisati funkcionalnosti brojača u *TCSR0* registru. U našem slučaju treba omogućiti prekid na brojaču 0 postavljanjem bita *ENIT0* na jedinicu, i uključiti *autoreload* funkciju postavljanjem bita *ARHT0* na jedinicu. Ostatak bita treba da je na nuli.

```
...
      iowrite32(XIL_AXI_TIMER_CSR_ENABLE_INT_MASK |
                XIL_AXI_TIMER_CSR_AUTO_RELOAD_MASK,
                tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
...
```

5. Pokrenuti brojač postavljanjem bita *ENT0* na jedinicu

```
...
      data = ioread32(tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
      iowrite32(data | XIL_AXI_TIMER_CSR_ENABLE_TMR_MASK,
                  tp->base_addr + XIL_AXI_TIMER_TCSR_OFFSET);
```

3.4. Funkija za upis (*timer_write*)

Kao što je malopre napomenuto, očekujemo od korisnika komandu koja sadrži dva broja odvojena zarezom. Prvi je broj prekida koji se treba izvršiti, a drugi perioda u milisekundama. Već smo u par navrata videli kako se iz stringa mogu ove vrednosti efikasno izdvojiti pomoću *sscanf* funkcije. Vrednosti se smeštaju u promenljive *number* i *millis*. Kako bismo podesili broj prekida dovoljno je da novu vrednost *number* dodelimo globalnoj promenljivoj *i_num.* Za podešavanje periode možemo pozvati prethodno opisanu pomoćnu funkciju *setup_and_start_timer* kojoj ćemo poslati vrednost *millis* putem argumenta. Sa ovim na umu izgled timer_write funkcije je sledeći:

```
ssize_t timer_write(struct file *pfile, const char __user *buffer,
size_t length, loff_t *offset)
{
	char buff[BUFF_SIZE];
	int millis = 0;
	int number = 0;
	int ret = 0;
	ret = copy_from_user(buff, buffer, length);
	if(ret)
		return -EFAULT;
	buff[length] = '\0';

	ret = sscanf(buff,"%d,%d",&number,&millis);
	if(ret == 2)//two parameters parsed in sscanf
	{

		if (millis > 40000)
		{
			printk(KERN_WARNING "xilaxitimer_write: Maximum period
exceeded, enter something less than 40000 \n");
		}
		else
		{
			printk(KERN_INFO "xilaxitimer_write: Starting timer
for %d interrupts every %d miliseconds \n",number,millis);
			i_num = number;
			setup_and_start_timer(millis);
		}

	}
	else
	{
		printk(KERN_WARNING "xilaxitimer_write: Wrong format,
expected n,t \n\t n-number of interrupts\n\t t-time in ms between
interrupts\n");
	}
	return length;
}
```

3.5. Testiranje drajvera

Ukoliko kompajliramo modul, te pozovemo **insmod** komandu, u terminalu dobijemo sledeći ispis:

```
[  +4.585877] xilaxitimer_init: Char device region allocated
[  +0.004301] xilaxitimer_init: Class created
[  +0.003919] xilaxitimer_init: Device created
[  +0.002893] xilaxitimer_init: Cdev added
[  +0.004901] xilaxitimer_init: Hello world
[  +0.006568] xilaxitimer_probe: Registered IRQ 165
[  +0.009749] xilaxitimer_probe: Timer platform driver registered
```

Može se primetiti da se uspešno završila ***timer_init*** funkcija i neposredno nakon nje i ***timer_probe*** funkcija, što nam govori da je drajver uparen sa uređajem.

Ukoliko pošaljemo string “3,1000” pomoću *echo* komande, dobijamo sledeći ispis u terminalu:

```
# echo 3,1000 > /dev/timer
[Dec15 01:30] xilaxitimer_write: Starting timer for 3 interrupts.
                                 One every 1000 miliseconds
[  +1.006546] xilaxitimer_isr: Interrupt 0 occurred !
[  +0.999998] xilaxitimer_isr: Interrupt 1 occurred !
[  +1.000001] xilaxitimer_isr: Interrupt 2 occurred !
[  +0.003490] xilaxitimer_isr: All of the interrupts have occurred.
                               Disabling timer
```

Iz ispisa koje je dala prekidna rutina i pomoćnih relativnih vremena datih u uglastim zagradama se vidi da su se desila tri prekida, i to jedan svake sekunde, što potvrđuje pravilan rad ovog drajvera.