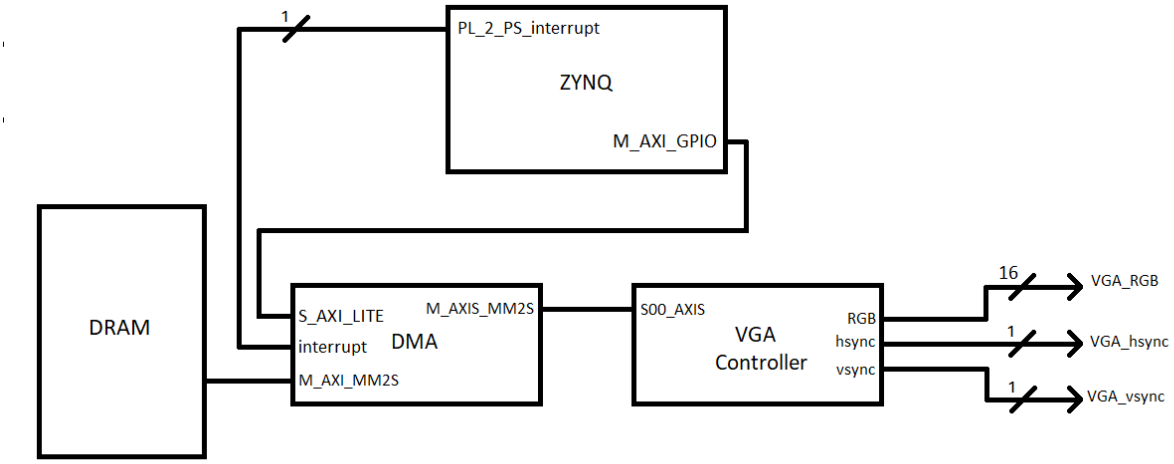
Implementacija VGA kontrolera na Zybo razvojnoj ploči

Opis sistema

Za razliku od prethodnog sistema kod koga je slika koju je trebalo prikazati na monitoru bila smeštana u BRAM memoriju, u ovom sistemu će se slika nalaziti u RAM memoriji Zybo razvojne ploče, a razlog za to je taj da slika rezolucije 640x480 ne može da stane u BRAM memoriju. Način kojim se omogućava direktan pristup RAM memoriji i pojedinačnim pikselima slike koji su tu smešteni je pomoću Xilinx DMA (Direct Memory Access) kontrolera. Na sledećoj slici je prikazan jedan takav sistem:

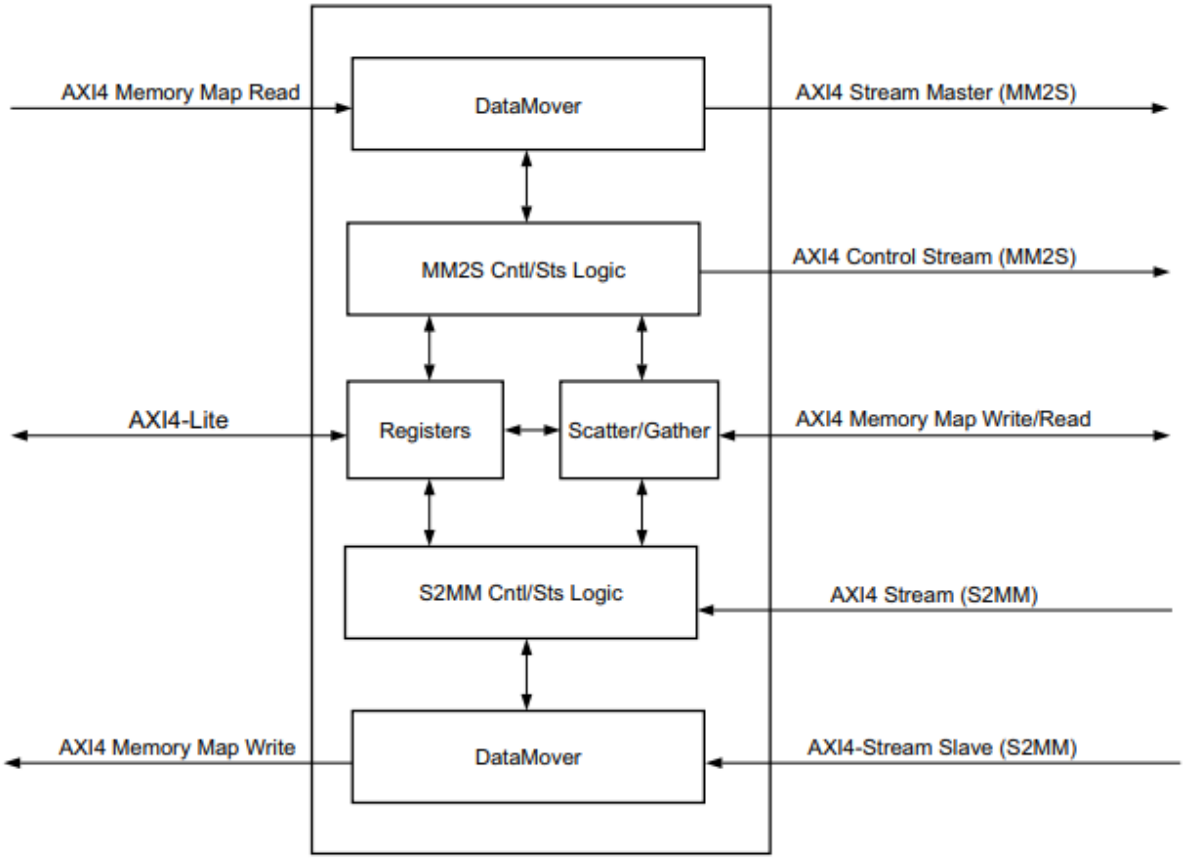


Slika 1.

Sa slike se može videti da se sistem sastoji iz sledećih komponenti:

- DMA (Direct Memory Access)
- Zynq procesor
- VGA Controller

AXI DMA (Direct Memory Access) kontroler omogućava ulazno/izlaznom uređaju (VGA kontroler u ovom slučaju) da šalje ili prima podatke iz RAM memorije bez direktnog posredstva procesora. Uloga procesora je da preko AXI Lite interfejsa konfiguriše AXI DMA upisom u njegove registre.



Slika 2.

Nakon što je procesor konfigurisao DMA kontroler, kontroler uzima podatke iz memorije bez daljeg posredstva procesora, čime se procesor rasterećuje, i može da obavlja druge funkcije. Informacija o tome da li je DMA kontroler završio sa prenosom paketa se može dobiti u vidu prekida koji DMA kontroler šalje procesoru, ili metodom prozivke gde procesor čitanjem određenog statusnog registra proverava da li je DMA kontroler završio sa prenosom. Konfigurisanje registara AXI DMA kontrolera zavisi od toga u kom režimu rada se on nalazi, i ti režimi mogu biti:

- ***Direct Register Mode*** (režim direktnog adresiranja) se koristi kada su podaci kojima se pristupa kontinualni u memoriji, odnosno svi podaci se nalaze jedan za drugim, i maksimalna veličina paketa iznosi 2^23 bajtova.
- ***Scatter-Gather mode*** se koristi kada su podaci kojima se pristupa razbacani u memoriji, odnosno nalaze se u blokovima koji nisu kontinualni
- ***Micro mode*** (mikro režim) se koristi kada je potrebno iz memorije prebacivati male pakete podataka. Broj podataka koji se šalje ne sme da bude veći od *data_width*burst_length/8* odnosno ako je širina podataka 32 bita, i ako je *burst_length*=256, onda je maksimalna veličina paketa 1024 bajta. *Data width i burst length* su parametri koji se mogu podešavati prilikom instanciranja DMA kontrolera u Vivado alatu.

Za potrebe ovog sistema DMA kontroler je potrebno konfigurisati da radi u direktnom režimu, i on će biti detaljno opisan, dok se za ostale režime upućuje na sledeći pdf:

https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf

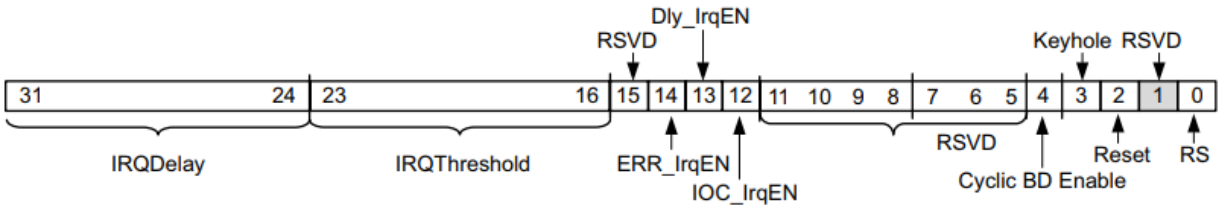
Direct Register Mode (režim direktnog adresiranja)

U ovom režimu DMA može da šalje podatke iz memorije prema nekom IP jezgru (to je slučaj sa VGA kontrolerom u ovom sistemu) ili da podatke koje šalje IP jezgro smešta u memoriju.

U slučaju prenosa podataka iz memorije prema IP jezgru, neophodno je ispoštovati sledeće korake:

1) Izvršiti reset DMA komponente upisivanjem logičke jedinice na Reset bit unutar memorijski mapiranog MM2S_DMACR (slika 3) registra koji se nalazi na adresi 0x0, odnosno potrebno je na lokaciju 0x0 upisati vrednost 0x00000004.
2) Pokrenuti MM2S (*memory to stream*) kanal upisivanjem logičke jedinice na run/stop bit unutar memorijski mapiranog MM2S_DMACR registra koji se nalazi na adresi 0. Odnosno potrebno je upisati na adresu 0x0 vrednost 0x00000001.
3) Nakon toga sledi opciono omogućavanje prekida tako što se na bite IOC_IrqEn i Err_IrqEn unutar MM2S_DMACR (slika 3) memorijski mapiranog registra upiše logička jedinica. Odnosno na adresu 0x0 neophodno je upisati vrednost 0x00005000.
4) Nakon toga potrebno je upisati validnu početnu adresu u RAM memoriji od koje DMA komponenta treba da krene sa prenosom podataka ka IP jezgru. Ta informacija se upisuje u MM2S_SA registar, koji je memorijski mapiran na adresu 0x18.
5) Poslednje što je potrebno uraditi jeste da se u registar MM2S_LENGTH upiše broj bajtova koji je potrebno preneti iz RAM memorije ka ulazno/izlaznom uređaju. Taj registar je memorijski mapiran na adresi 0x28. **Ovaj registar se mora podesiti poslednji**.

Na sledećoj slici je prikazan MM2S_DMACR registar, on je veličine 32 bita, i sa slike se vidi zašto je u njega potrebno upisati na primer vrednost 0x00000004 kako bi se sistem resetovao:



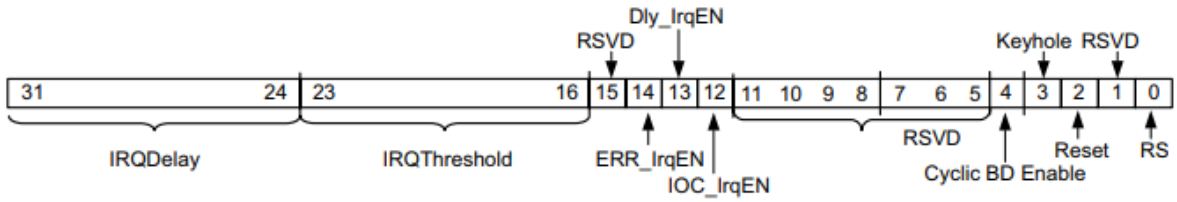
Slika 3.

Detalji o tome šta se konfiguriše određenim bitima u ovom registru mogu da se nađu u pdf-u na prethodnom linku.

Kada DMA kontroler prenosi podatke od ulazno izlaznog uređaja ka memoriji, procesor mora da konfiguriše kontroler prateći sledeće korake:

1) Izvršiti reset DMA komponente upisivanje logičke jedinice na Reset bit unutar memorijski mapiranog S2MM_DMACR (slika 4) registra koji se nalazi na adresi 0x0, odnosno potrebno je na lokaciju 0x0 upisati vrednost 0x00000004.
2) Pokrenuti S2MM (*stream to memory*) kanal upisivanjem logičke jedinice na run/stop bit unutar memorijski mapiranog S2MM_DMACR registra koji se nalazi na adresi 0. Odnosno potrebno je upisati na adresu 0x0 vrednost 0x00000001.
3) Nakon toga sledi opciono omogućavanje prekida tako što se na bite IOC_IrqEn i Err_IrqEn unutar S2MM_DMACR (slika 4) memorijski mapiranog registra upiše logička jedinica. Odnosno na adresu 0x0 neophodno je upisati vrednost 0x00005000.
4) Nakon toga potrebno je upisati validnu početnu adresu u RAM memoriji od koje DMA komponenta treba da upisuje podatke koje ulazno/izlazni uređaj šalje. Ta informacija se upisuje u S2MM_DA registar, koji je memorijski mapiran na adresu 0x18.
5) Poslednje što je potrebno uraditi jeste da se u registar S2MM_LENGTH upiše broj bajtova koje ulazno/izlazni uređaj upisuje u RAM memoriju preko DMA kontrolera. Taj registar je memorijski mapiran na adresi 0x28. **Ovaj registar se mora podesiti poslednji**.

Izgled S2MM_DMACR registra je prikazan na sledećoj slici:



Slika 4.

Napomena:

Prethodno pomenute adrese kojima se pristupa registrima DMA kontrolera su samo ofseti na baznu adresu koju generiše Vivado alat prilikom instanciranja komponente. Na primer da bi se upisala neka vrednost u S2MM_LENGTH registar, neophodno je upisati na adresu bazna_adresa+0x28(bazna_adresa+40, decimalnim zapisom). Ti ofseti su definisani u datasheet-u AXI DMA kontrolera, i taj datasheet se može skinuti sa prethodnog linka.

VGA kontroler je zadužen za generisanje hsync i vsync signala koji se prosleđuju na hsync i vsync portove VGA konektora. Ti signali su sinhronizacioni signali koji su neophodni kako bi monitor znao na koju poziciju na ekranu da iscrta piksel koji mu se trenutno šalje. RGB izlaz je povezan sa VGA_RGB izlazom VGA konektora, i u njemu je sadržana boja onog piksela koji

treba da se pojavi na određenoj lokaciji na ekranu. VGA kontroler prima vrednosti piksela od strane DMA kontrolera preko AXI *stream* interfejsa, i te vrednosti u pravovremenim trenucima prosleđuje na RGB izlaz (u skladu sa vsync i hsync signalima). Kako bi monitor uvek znao šta da iscrtava, neophodno je obezbediti da pikseli slike ne prestanu da stižu na RGB izlaz,odnosno DMA kontroler nakon završenog slanja slike kreće sa ponovnim slanjem iste. Slično je radio i VGA kontroler u prethodnoj skripti, stim što je tamo on stalno čitao vrednosti piksela slike iz BRAM memorije, dok ovde DMA kontroler mora stalno da šalje istu sliku VGA kontroleru kako bi se ona prikazala na ekranu. Da bi se neka druga slika prikazala na ekranu neophodno je piksele te slike upisati na iste memorijske lokacije u RAM memoriji.

Opis Drajvera

Sam drajver je jako sličan LED drajveru o kome je pričano na prethodnim vežbama, sa par dodatnih stvari koje će ovde biti opisane.

Init funkcija:

Za razliku od init funkcije drajvera za rad sa LED, u init funkciji drajvera za ovaj sistem neophodno je još uraditi sledeće:

```
                                                .....

tx_vir_buffer = dma_alloc_coherent(NULL, MAX_PKT_LEN, &tx_phy_buffer, GFP_DMA | GFP_KERNEL);
if(!tx_vir_buffer)
{
   printk(KERN_ALERT "Could not allocate dma_alloc_coherent for img");
   goto fail_3;
}
else
   for (i = 0; i < MAX_PKT_LEN/4;i++)
      tx_vir_buffer[i] = 0x00000000;

                                                ......
```

Slika 5.

Kao što je prethodno pomenuto AXI DMA kontroler je potrebno konfigurisati u režimu direktnog adresiranja (*eng. Direct register mode*), odakle sledi da podaci moraju biti sekvencijalni u memoriji. Da bi se to postiglo koristi se sledeća funkcija:

*void *dma_alloc_coherent(struct device *dev, size_t size, dma_addr_t *dma_handle, int flag);*

Parametri ove funkcije su:

- ***struct device *dev*** – Uređaj za koji se izdvaja koherentna memorija (u ovom slučaju se tu prosleđuje vrednost NULL).

- ***size_t size*** - veličina memorije koju je potrebno alocirati.
- ***dma_addr_t *dma_handle*** – adresa koja se prosleđuje DMA kontroleru, i od te adrese on kreće da šalje podatke.
- ***int flag*** – Uvek se specificira jedan od dva flega: GFP_ATOMIC ili GFP_KERNEL. GFP_ATOMIC znači da proces koji poziva ovu funkciju ne sme da se blokira pri alokaciji. Iz tog razloga može da se desi da ova funkcija ne dobije memorijski prostor, jer ga trenutno nema na raspolaganju, i vratiće se poruka o neuspelom alociranju. Kada se *flag* postavi na GFP_KERNEL omogućava se blokiranje pri alokaciji, odnosno ukoliko funkcija ne dobije memorijski prostor, ona će se blokirati, i čekaće dok se taj prostor ne oslobodi. Pored ova dva, mogu se postaviti dodatni flegovi, kao što je GFP_DMA koji zahteva memorijski opseg unutar prvih 16MB fizičke memorije.
- ***povratna vrednost*** – je kernel virtuelna adresa na početak alocirane memorije. Preko ove adrese drajver može da menja sadržaj alociranog memorijskog opsega, u ovom slučaju je to upis nove slike koju DMA kontroler prosleđuje monitoru.

Razlog zašto memorija nije koherentna je zato što Linux operativni sistem radi sa virtuelnim adresama, odnosno memorijski prostori u koje su smešteni programi nisu direktno mapirani na adrese fizičke memorije koje koristi hardver. Stoga, da nije pozvana funkcija *dma_alloc_coherent,* već da je samo napravljen običan niz u koji se smeštaju vrednosti piksela slike i da se DMA kontroleru prosledio pokazivač na početak tog niza, kontroler ne bi radio kako treba. Razlog je to što on skuplja vrednosti jednu za drugom iz fizičke memorije (samo radi ofset u odnosu na prosleđenu adresu), a linux operativni sistem je prilikom izdvajanja memorije za taj niz koristio virtuelne adrese i podaci koje je on smestio u fizičku memoriju se ne nalaze jedan iza drugog (nisu koherentni).

Exit funkcija:

U exit funkciji je neophodno osloboditi zauzetu memoriju pomoću *dma_alloc_coherent* i to na sledeći način:

void dma_free_coherent(structdevice dev, size_t size ,void* vaddr, dma_addr_t dma_handle);*

Parametri funkcije su:

- ***struct device *dev*** – uređaj za koji se izdvaja koherentna memorija (u ovom slučaju se tu prosleđuje vrednost NULL).
- ***size_t size*** - veličina memorije koju je potrebno osloboditi.
- ***void* vaddr*** – pokazivač na kernel virtuelnu adresu. To je povratna vrednost *dma_alloc_coherent* funkcije.
- ***dma_addr_t dma_handle*** – pokazivač na fizičku adresu koja se prosleđuje DMA komponenti.
- ***povratna vrednost*** – nema povratne vrednosti.

Probe funkcija:

Unutar probe funkcije neophodno je:

- Zahtevati broj prekida, na osnovu koga se može registrovati prekidni signal koji šalje DMA kontroler kada završi sa prenosom jednog paketa
- Inicijalizovati DMA kontroler.

```
                                                .....

vp->irq_num = platform_get_irq(pdev, 0);
printk("irq number is: %d\n", vp->irq_num);

if (request_irq(vp->irq_num, dma_isr, 0, DEVICE_NAME, NULL))
{
   printk(KERN_ERR "vga_dmai_init: Cannot register IRQ %d\n", vp->irq_num);
   return -EIO;
}
else
{
   printk(KERN_INFO "vga_dma_init: Registered IRQ %d\n", vp->irq_num);
}
  /* INICIJALIZACIJA DMA kontrolera*/
 dma_init(vp->base_addr);
 dma_simple_write(tx_phy_buffer, MAX_PKT_LEN, vp->base_addr);

                                                .....
```

Slika 6.

Da bi se dobio broj prekida, potrebno je koristiti funkciju *platform_get_irq (pdev, 0)*, čija je povratna vrednost broj prekida. Kada se dobije broj prekida, pozivanjem funckije *request_irq* se taj broj povezuje sa prekidnom rutinom (funkcijom koja se poziva svaki put kada se desi prekid). Ime te prekidne rutine je u ovom slučaju *dma_isr* i ona je opisana kasnije.

DMA kontroler se inicijalizuje korišćenjem *dma_init i dma_simple_write* funkcija. One su napisane na sledeći način:

```
int dma_init(void __iomem *base_address)
{
  u32 reset = 0x00000004;
  u32 IOC_IRQ_EN;
  u32 ERR_IRQ_EN;
  u32 MM2S_DMACR_reg;
  u32 en_interrupt;

  IOC_IRQ_EN = 1 << 12; //  IOC_IrqEn bit u MM2S_DMACR registru
  ERR_IRQ_EN = 1 << 14; // Err_IrqEn bit u MM2S_DMACR registru

  iowrite32(reset, base_address); // upisivanje u MM2S_DMACR registar. postavljanje reset bita na logičku jedinicu (3. bit)

  MM2S_DMACR_reg = ioread32(base_address); // čitanje iz MM2S_DMACR registra DMA kontrolera
  en_interrupt = MM2S_DMACR_reg | IOC_IRQ_EN | ERR_IRQ_EN;// postavljanje na logičku jedinicu 13. i 15. bita u MM2S_DMACR

  iowrite32(en_interrupt, base_address); // upisivanje u MM2S_DMACR registar
  return 0;
}
```

Slika 7.

Funkcija je realizovana tako da se ispoštuju drugi, treći i četvrti korak koji su navedeni kada je objašnjen način konfigurisanja DMA kontrolera u režimu direktnog adresiranja. Odnosno sistem je resetovan, omogućeni su prekidi upisivanjem prave vrednost u MM2S_DMACR registar i u registar MM2S_DMACR registar je upisana početna adresa od koje DMA kontroler treba da započne sa slanjem podataka.

Parametri *dma_init* funkcije su:

- *void __iomem *base_address* - pokazivač na adresu preko koje se pristupa memorijski mapiranim registrima unutar DMA kontrolera preko AXI_LITE interfejsa.

Napomena:

Obratiti pažnju da je nakon reseta, pre upisivanja nove vrednosti u registar MM2S_DMACR, taj registar prvo pročitan korišćenjem ioread32 (base_address) funkcije, i nakon toga je ta vrednost promenjena u upisana ponovo u MM2S_DMACR registar. To je urađeno kako se slučajno ne bi poništile moguće prethodne konfiguracije.

Dma_simple_write funkcija je realizovana tako da se ispoštuju prvi i peti korak konfigurisanja DMA kontrolera u režimu direktnog adresiranja:

```
u32 dma_simple_write(dma_addr_t TxBufferPtr, u32 max_pkt_len, void __iomem *base_address)
{
   u32 MM2S_DMACR_reg;

   MM2S_DMACR_reg = ioread32(base_address); // čitanje iz MM2S_DMACR registra

   iowrite32(0x1 |  MM2S_DMACR_reg, base_address); // setovanje RS bita u MM2S_DMACR registru
čime startujemo DMA.

   iowrite32((u32)TxBufferPtr, base_address + 24); // Upisivanje u MM2S_SA registar vrednost odakle
DMA kontroler da krene.

   iowrite32(max_pkt_len, base_address + 40); // upisivanje u MM2S_LENGTH registar veličinu paketa
koji DMA kontroler treba da pošalje. U ovom slučaju ta veličina je (640*480*4).
  return 0;
}
```

Slika 8.

dma_simple_write funkciju je neophodno pozvati svaki put kada DMA kontroler treba da pošalje paket (u ovom slučaju sliku), i ona je pozvana u *probe* funkciji kako bi se inicijalno aktivirao DMA kontroler. Svaki sledeći put se ona poziva iz prekidne rutine, koja je objašnjena u nastavku.

Parametri *dma_simple_write* funkcije su:

- *dma_addr_t TxBufferPtr* – pokazivač na adresu koju je generisala *dma_alloc_coherent f*unkcija (pogledati sliku 5, *tx_phy_buffer* pokazivač). To je fizička adresa koja se prosleđuje DMA kontroleru kako bi on znao odakle da započne sa slanjem podataka iz memorije.
- *u32 max_pkt_len* – veličina paketa koji DMA kontroler treba da pošalje iz memorije.
- *void __iomem *base_address* - pokazivač na adresu preko koje procesor pristupa memorijski mapiranim registrima unutar DMA kontrolera preko AXI_LITE interfejsa.

Prekidna rutina:

Iz razloga što je potrebno da vrednosti piksela konstantno dolaze na VGA interfejs, nakon završetka prvog slanja slike korišćenjem dma_simple_write funkcije, prekidna rutina će inicirati sledeće slanje i ciklus se nastavlja. Prekidna rutina je realizovana na sledeći način*:*

```
static irqreturn_t dma_isr(int irq,void*dev_id)
{
   u32 IrqStatus;

   IrqStatus = ioread32(vp->base_addr + 4);//čitanje irq_status bita iz MM2S_DMASR registra
   iowrite32(IrqStatus | 0x00007000, vp->base_addr + 4);//clear-ovanje irq_status bita u
MM2S_DMASR registru. (To se radi upisivanjem logičke  1 na 13. bit u MM2S_DMASR (IOC_Irq).

   /*Slanje transakcije*/
  dma_simple_write(tx_phy_buffer, MAX_PKT_LEN, vp->base_addr);
  return IRQ_HANDLED;;
}
```

Slika 9.

Ova funkcija se poziva svaki put kada se desi prekid, odnosno svaki put kada DMA kontroler pošalje jedan paket(sliku). Način na koji funkcija mora da se izvši je sledeći:

1) Prvo je potrebno pročitati statusni bit iz MM2S_DMASR registra koji se nalazi na adresi pomerenoj za 4 u odnosu na baznu adresu (**base_addr + 4**), i koji govori da li se prekid desio.
2) Nakon toga potrebno je očistiti (*eng. clear*) statusni bit, tako što će se u registar MM2S_DMASR upisati vrednost *irqStatus* | *0x00007000*.
3) Kada su se prethodna dva koraka izvršila poziva se funkcija *dma_simple_write* koja kaže DMA kontroleru da ponovo pošalje sliku.

Write funkcija:

Write funkcija je napisana tako da se preko nje na proizvoljnu lokaciju na ekranu može upisati određeni piksel. String koji write funkcija očekuje mora biti napisan u formatu “x,y,rgb”, gde x predstavlja poziciju na horizontalnoj osi gde piksel treba da se nađe, y predstavlja poziciju na vertikalnoj osi i rgb predstavlja vrednost piksela koja treba da se upiše. Vrednost rgb mora da bude 16-bitna, pri čemu biti 15-11 predstavljaju crvenu boju, 10-5 zelenu i 4-0 plavu boju. Sledeći kod listing predstavlja primer aplikacije koja prosleđuje write funkciji drajvera poziciju i vrednost piksela:

```
fp = fopen("/dev/vga_dma", "w");
if(fp == NULL)
{
   printf("Cannot open /dev/vga for write\n");
   return -1;
}
fprintf(fp,"%d,%d,%#04x\n",50,250,0xf100);
fclose(fp);
if(fp == NULL)
{
   printf("Cannot close /dev/vga\n");
   return -1;
}
```

Slika 10.

Fprintf funkcija u ovom slučaju prosleđuje write funckiji drajvera piksel na poziciji x=50 i y=250, sa vrednošću 0xf100(crveni piksel).

Za detalje o tome kako je napisana write funkcija pogledati *vga_driver.c* izvorni kod.

MMAP funkcija:

Mana korišćenja drajvera kod prenosa velikih količina podataka je da se gubi većina procesorskog vremena na prebacivanje podataka u string i na parsiranje istog stringa u drajveru. Ukoliko pokušamo da pošaljemo sliku 640x480 preko write funkcije, možemo primetiti da će biti potrebno 5-10 sekundi da se iscrta čitava slika. U ove svrhe postoji alternativan način upisa u memoriju - memorijsko mapiranje. Memorijsko mapiranje (*mmap*) se koristi kako bi se deo adresnog prostora procesa mapirao na određeni adresni prostor uređaja. Na ovaj način neki proces (aplikacija) može da ima direktan pristup memorijskim lokacijama uređaja. Ova funkcija se nakon implementacije postavlja na odgovarajući pokazivač u file_operations strukturi.

```
static ssize_t vga_dma_mmap(struct file *f, struct vm_area_struct *vma_s)
{
   int ret = 0;
   long length = vma_s->vm_end - vma_s->vm_start;
   //printk(KERN_INFO "DMA TX Buffer is being memory mapped\n");

   if(length > MAX_PKT_LEN)
   {
     return -EIO;
     printk(KERN_ERR "Trying to mmap more space than it's allocated\n");
   }

   ret = dma_mmap_coherent(NULL, vma_s, tx_vir_buffer, tx_phy_buffer, length);
   if(ret<0)
   {
      printk(KERN_ERR "memory map failed\n");
      return ret;
   }
   return 0;
}
```

Slika 11.

Prototip *dma_mmap_from_coherent* je sledeći:

```
int dma_mmap_from_coherent (        struct device * dev,
        struct vm_area_struct * vma,
        void * vaddr,
        dma_addr_t  * dma_handle,
        size_t size);
```

Parametri ove funkcije su:

- ***struct device **** *dev* – uređaj za koji se alocira memorija (u ovom slučaju tu se prosleđuje 0).
- **struct vm_area_struct * vma** – je kernel struktura koja sadrži informacije o virtuelnom adresnom prostoru procesa koji poziva *mmap* funkciju drajvera. Pomoću *dma_mmap_coherent* funkcije drajver mapira taj virtuelni adresni prostor na fizički prostor alociran pomoću *dma_alloc_coherent* funkcije. U ovom slučaju polja ove strukture koja su od interesa su vma->start, vma->end, i na osnovu njih može da se zaključi kolika je veličina adresnog prostora procesa koji je potrebno mapirati. Za više informacija o ovoj strukturi pogledati 15. poglavlje knjige *Linux Device Drivers.*
- **void * vaddr** – pokazivač na kernel virtuelnu adresu. To je povratna vrednost dma_alloc_coherent funkcije.
- ***dma_addr_t*** *dma_handle* – pokazivač na fizičku adresu koja se prosleđuje DMA komponenti.
- ***size_t size*** – broj podataka koji se mapiraju na alociranu memoriju.

Primer aplikacije (procesa) koja koristi mmap funkciju kako bi prosledila sliku koja će se prikazati na monitoru je sledeći:

```
int fd;
int *p;
fd = open("/dev/vga_dma", O_RDWR|O_NDELAY);
if (fd < 0)
{
        printf("Cannot open /dev/vga for write\n");
        return -1;
}
p=(int*)mmap(0,640*480*4, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
memcpy(p, image, MAX_PKT_SIZE);
munmap(p, MAX_PKT_SIZE);
close(fd);
if (fd < 0)
{
        printf("Cannot close /dev/vga for write\n");
        return -1;
}
```

Slika 12

U aplikaciji sa slike 12 koristi se ***mmap*** funkcija kako bi se određeni opseg adresa iz korisničkog prostora povezao sa memorijom uređaja (drajvera). Odnosno svaki put kada aplikacija čita iz ovog adresnog prostora ili upisuje u njega ona u stvari čita, ili upisuje u adresni prostor uređaja(drajvera).

Prototip te funkcije je:

*void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)*

Parametri funkcije su:

- ***void *start*** – označava početnu adresu od koje podaci započinju, i uglavnom je 0.
- ***size_t length*** – predstavlja veličinu memorije koju je potrebno memorijski mapirati.
- ***int prot*** – govori koja vrsta pristupa memorijski mapiranom prostoru je dozvoljena. Može da bude PROT_READ, PROT_WRITE, PROT_EXEC.
- ***int flags*** – informacija o prirodi memorijski mapiranog prostora. Uglavnom je MAP_SHARED, što znači da će svaka promena tog memorijskog prostora biti uočena od strane procesa koji mapira taj prostor.
- ***int fd*** – referenca na fajl (drajver).
- ***off_t offset*** – predstavlja vrednost koja se dodaje na parametar *start*. U ovom slučaju je 0 iz razloga što je potrebno da adresni prostor krene od nulte adrese.
- ***Povratna vrednost*** – jeste pokazivač na memorijski mapiran prostor.

MEMCPY funkcija se koristi kako bi se podaci kopirali na adresni prostor mapiran pomoću mmap funkcije.

void *memcpy(void *str1, const void *str2, size_t n)

Parametri funkcije su:

- ***void *str1*** – pokazivač na memorijski mapiran prostor.
- ***const void *str2*** – pokazivač na podatke koje je potrebno kopirati u memorijski mapiran prostor. U ovom slučaju to je pokazivač na niz koji sadrži u sebi piksele slike.
- ***size_t n*** – Količina podataka koje je potrebno smestiti u memorijski mapiran prostor.

MUNMAP funkcija uklanja željeni opseg memorijski mapiranog prostora. Ovo je potrebno uraditi kada proces ne koristi više memorijski mapirani prostor, i u ovom slučaju to se radi nakon što se slika pomoću *memcpy* kopira u taj prostor. Prototip funkcije je:

int munmap (void *addr, size_t length)

Parametri funkcije su:

- ***void *addr –*** pokazivač na memorijski mapirani adresni prostor
- ***size_t length –*** veličina memorijski mapiranog adresnog prostora.