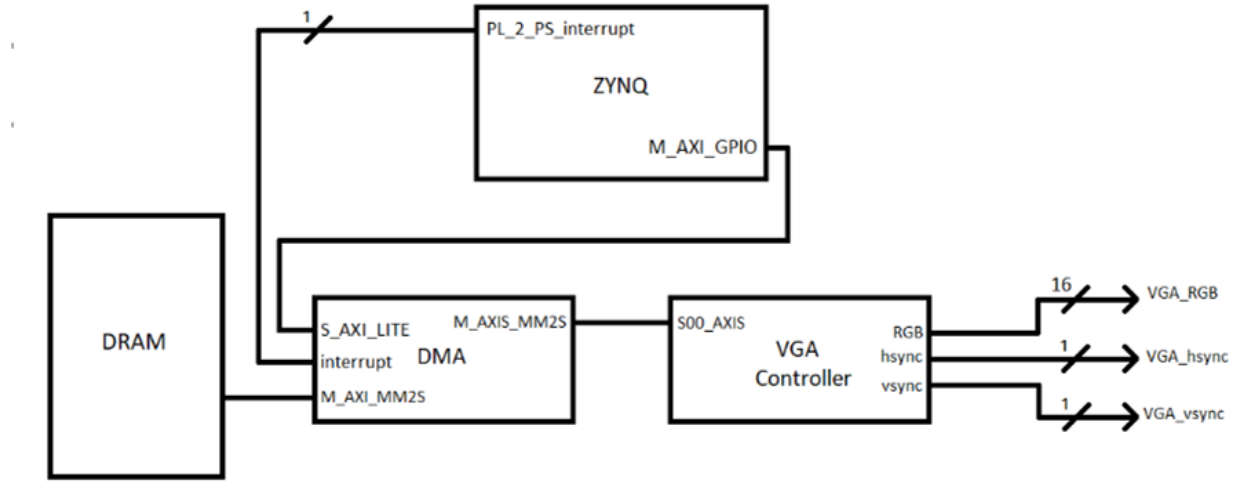
Vežba 12 : VGA DMA kontroler

1. Uvod

Sistem opisan na prethodnim vežbama omogućavao je isrctavanje slike rezolucije 256x144 piksela. U njemu, da bi se pikseli slike pojavili na ekranu, bilo je neophodno upisati ih pojedinačno, preko AXI_GPIO modula, u BRAM memoriju. Tako smeštenu sliku VGA modul je čitao iz memorije i prikazivao na ekranu. Problem ovakvog sistema jeste što on sliku iscrtava samo u gornjem levom uglu ekrana, odnosno može da oboji samo piksele koji se po horizontali nalaze u opsegu 0-255 i po vertikali u opsegu 0-143. Uzrok toga nije bio VGA modul, već BRAM memorija, koje na *Zybo* razvojnoj ploči ima "samo" 240KB, te nije moguće smestiti u nju sliku veće rezolucije[1]. Način da se ovaj problem reši jeste da sliku koju prikazujemo na ekranu VGA modul čita direktno iz DDR RAM memorije[2] (na *Zybo* razvojnoj ploči na raspolaganju imamo 512MB RAM memorije). No, to zahteva malo kompleksniji sistem koji će imati direktan pristup memorijskim lokacijama RAM memorije u kojima se nalazi slika. U nastavku je prikazana pojednostavljena blok šema jednog takvog sistema:



Slika 1. Blok šema sistema koji omogućava iscrtavanja slike na ekranu preko VGA interfejsa

[1] Ukoliko bismo hteli da isrctamo 640*480 = 307200 piksela bilo bi nam potrebna BRAM memorija veličine 614400 bajtova, jer je svaki piksel veličine 2 bajta.
[2] Do sada smo sliku koja se nalazi u RAM memoriji piksel po piksel smeštali u BRAM.

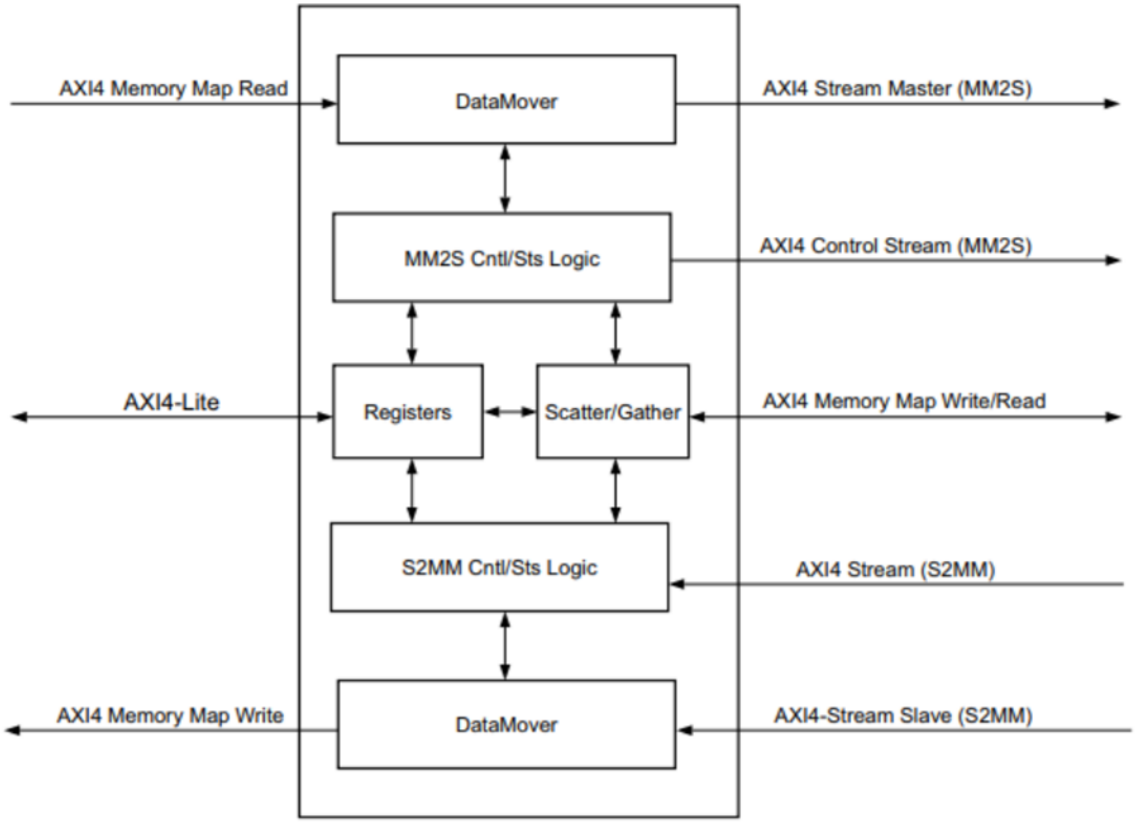
Sa slike se može videti da se sistem sastoji iz sledećih komponenti:

- DMA (Direct Memory Access)
- VGA Controller

VGA kontroler ima istu ulogu kao i u prethodnom sistemu, s tim što je sada proširen AXI *stream* interfejsom preko kojega prima pojedinačne piksele slike. Ta promena neće uticati na pisanje drajvera za ovaj sistem (jer procesor nema direktan pristup VGA kontroleru), ali ono što hoće uticati jeste AXI DMA kontroler.

2. AXI DMA (Direct Memory Access)

AXI DMA (Direct Memory Access) kontroler omogućava ulazno/izlaznom uređaju (u ovom slučaju VGA kontroler) 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 određene memorijski mapirane registre) i na taj način mu kaže iz kog memorijskog opsega RAM memorije da šalje podatke. Na sledećoj slici je prikazana blok šema onoga što se nalazi unutar AXI DMA kontrolera:



Slika 2. AXI DMA kontroler

Ono što će biti nama od interesa jeste blok označen sa “Registers”. Tom bloku se pristupa preko AXI lite interfejsa, i modifikacijom memorijski mapiranih registara unutar tog modula se konfiguriše AXI DMA.

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 od sledeće navedenih režima rada želimo da bude:

- **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.
- **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.

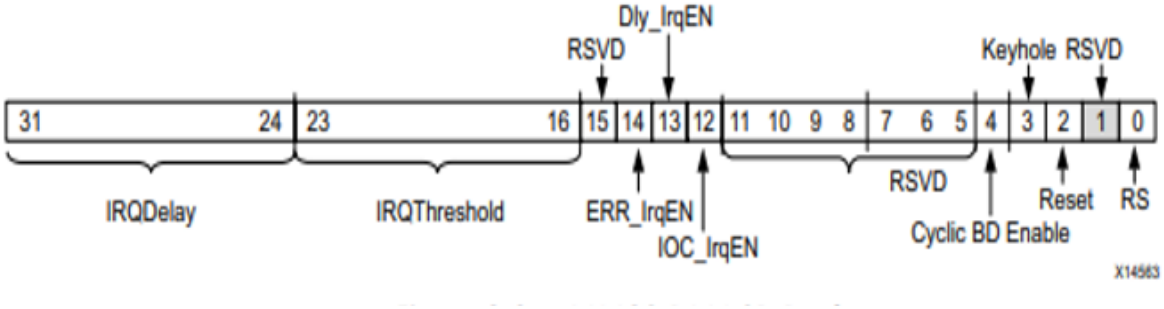
Da bi se realizovao sistem sa slike 1, AXI DMA kontroler je potrebno konfigurisati da radi u direktnom režimu, i on će biti detaljno opisan, dok se za ostale režime upućuje na sledeći pdf:
https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf

Ovako realizovan sistem kod koga AXI DMA direktno šalje piksele slike iz RAM memorije VGA kontroleru ima jednu manu, a to je da sliku iz memorije moramo konstantno slati VGA kontroleru, odnosno kada se pošalju svi pikseli, odmah treba krenuti iznova. Ovo je problem jer svako novo slanje zahteva intervenciju procesora koji mora ponovo preko AXI *lite* interfejsa da pošalje instrukciju AXI DMA kontroleru, jer on neće automatski krenuti ispočetka. Da bi se procesoru olakšao posao, AXI DMA će generisati prekid svaki put kada pošalje sve piksele i kada procesor detektuje prekid poslaće instrukciju DMA komponenti da krene sa ponovnim slanjem (Sa slike 1 se može videti da je *interrupt* port DMA komponente povezan na *interrupt* port procesora).

Direct Register Mode (režim direktnog adresiranja)

U ovom režimu DMA može da šalje podatke iz memorije prema nekom IP jezgru (VGA kontroleru 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 postaviti bit na poziciji 2 na logičku jedinicu.
2. Pokrenuti *MM2S* (memory to stream) kanal upisivanjem logičke jedinice na *run/stop* bit unutar memorijski mapiranog *MM2S_DMACR* (slika 3) registra koji se nalazi na adresi 0. Odnosno bit na poziciji 1 postaviti na logičku jedinicu.
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 bite na pozicijama 14 i 12 postaviti na logičke jedinice.
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!**

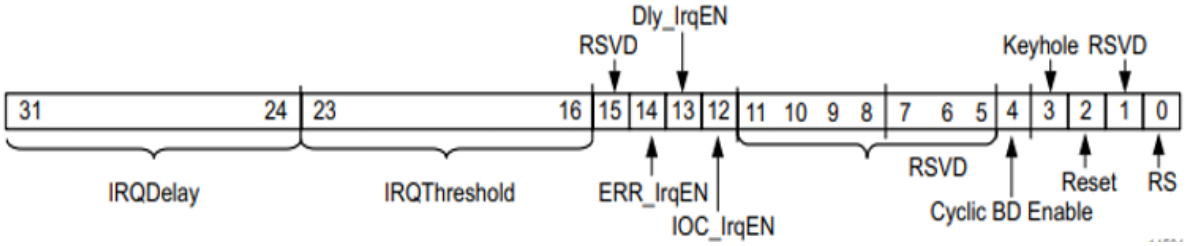


Slika 3. MM2S_DMACR registar

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 0x30. Odnosno postaviti bit na 2. poziciji na logičku jedinicu.

2. Pokrenuti *S2MM* (stream to memory) kanal upisivanjem logičke jedinice na run/stop bit unutar memorijski mapiranog *S2MM_DMACR* (Slika 4) registra koji se nalazi na adresi 0x30. Odnosno bit na poziciji 1 postaviti na logičku jedinicu..
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 bite na pozicijama 14 i 12 postaviti na logičke jedinice.
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!**



Slika 4. S2MM_DMACR registar

Napomena:

Prilikom modifikacije pojedinačnih bita memorijski mapiranih registara voditi računa da ne izbrišete prethodne modifikacije. Takođe prethodno pomenute adrese preko kojih 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.

3. Drajver

Ovaj drajver je sličan LED drajveru opisanom na vežbi 8, i u nastavku će biti opisane samo razlike medju njima.

3.1. Module init funkcija

```
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;
```

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 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).

3.2. Module exit funkcija

U njoj je neophodno osloboditi memoriju zauzetu 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.

3.3. Probe funkcija

U sistemu sa slike 1 DMA komponenta prekidom obaveštava procesor da je poslala jednu sliku, te odatle sledi da u *probe* funkciji, kao i kod *axi timera* obrađivanog na vežbi 10, treba saznati jedinstveni broj koji dodeljen prekidu koji izaziva AXI DMA. Takođe pored toga treba inicijalizovati DMA u režim direktnog adresiranja.

```
vp->irq_num = platform_get_irq(pdev, 0);
if(!vp->irq_num)
{
       printk(KERN_ERR "vga_dma_probe: Could not get IRQ resource\n");
       rc = -ENODEV;
       goto error2;
}

if (request_irq(vp->irq_num, dma_isr, 0, DEVICE_NAME, NULL))
{
      printk(KERN_ERR "vga_dma_probe: Could not register IRQ %d\n", vp->irq_num);
      return -EIO;
}
else
{
    printk(KERN_INFO "vga_dma_probe: 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);
```

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 i 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
// upisivanje u MM2S_DMACR registar. Postavljanje reset bita na logičku jedinicu
(3. bit)
iowrite32(reset, base_address);
// čitanje iz MM2S_DMACR registra DMA kontrolera
MM2S_DMACR_reg = ioread32(base_address);
// postavljanje na logičku jedinicu 13. i 15. bita u MM2S_DMACR
en_interrupt = MM2S_DMACR_reg | IOC_IRQ_EN | ERR_IRQ_EN;
iowrite32(en_interrupt, base_address); // upisivanje u MM2S_DMACR registar
return 0;
}
```

Funkcija je realizovana tako da se ispoštuju prvi i treći korak koji su navedeni kada je objašnjen način konfigurisanja DMA kontrolera u režimu direktnog adresiranja. Odnosno sistem je resetovan i omogućeni su prekidi upisivanjem prave vrednost u *MM2S_DMACR* registar. Parametri *dma_init* funkcije su:

- **void __iomem *base_address** - pokazivač na baznu 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 drugi, četvrti 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

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

   // Upisivanje u MM2S_SA registar vrednost odakle DMA kontroler da krene.
```

```
   iowrite32((u32)TxBufferPtr, base_address + 24);

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

Nju 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* funkcija (pogledati kodni listing init funkcije, 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 baznu adresu preko koje procesor pristupa memorijski mapiranim registrima unutar DMA kontrolera preko AXI_LITE interfejsa.

3.4. Funkcija za opsluživanje prekida (ISR)

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;
  //čitanje irq_status bita iz MM2S_DMASR registra
  IrqStatus = ioread32(vp->base_addr + 4);
  //clear irq_status bita u MM2S_DMASR registru. (To se radi
  //upisivanjem logičke 1 na 13. bit u MM2S_DMASR (IOC_Irq).
  iowrite32(IrqStatus | 0x00007000, vp->base_addr + 4);
  /*Slanje transakcije*/
  dma_simple_write(tx_phy_buffer, MAX_PKT_LEN, vp->base_addr);
  return IRQ_HANDLED;;
}
```

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 izvrši 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.

3.5. Funkcija za upis podataka

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.

```
sscanf(buff,"%d,%d,%s", &xpos, &ypos, rgb_buff);
ret = kstrtoull(rgb_buff, 0, &rgb);
tx_vir_buffer[640*ypos + xpos] = (u32)rgb;
```

3.6. Funkcija za memorijsko mapiranje

Mana korišćenja prethodno opisanog 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. Za ovu svrhu 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. U našem slučaju bilo bi dobro da aplikacija ima direktan pristup memorijskom prostoru u koji se smešta slika koju DMA šalje iz RAM-a. Ukoliko bi to bilo omogućeno aplikacije ne bi morala prvo da šalje piksel *write* funkciji drajvera, već bi direktno bez posrednika mogla da menja te memorijske lokacije.

```
 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;
}
```

Unutar *vga_dma_mmap* funkcije poziva se *dma_mmap_from_coherent* koja u stvari vrši memorijsko mapiranje adresnog prostora uređaja na adresni prostor procesa. Njen prototip je sledeći:

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

Parametri ove funkcije imaju sledeće značenje:

- **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 dat u nastavku.

```
int main()
{
  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;
  }
  return 0;
}
```

U aplikaciji iz prethodnog kodnog listinga funkcija *mmap* se koristi 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** - je pokazivač na memorijski mapiran prostor.

MEMCPY funkcija se koristi kako bi se podaci kopirali na adresni prostor mapiran pomoću mmap funkcije(u ovom slučaju u taj adresni prostor kopira se slika).

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, odnosno nakon što se ta funkcija izvrši aplikacija više nema pristup memoriji kojoj je imala pristup nakon mapiranja. 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. 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.