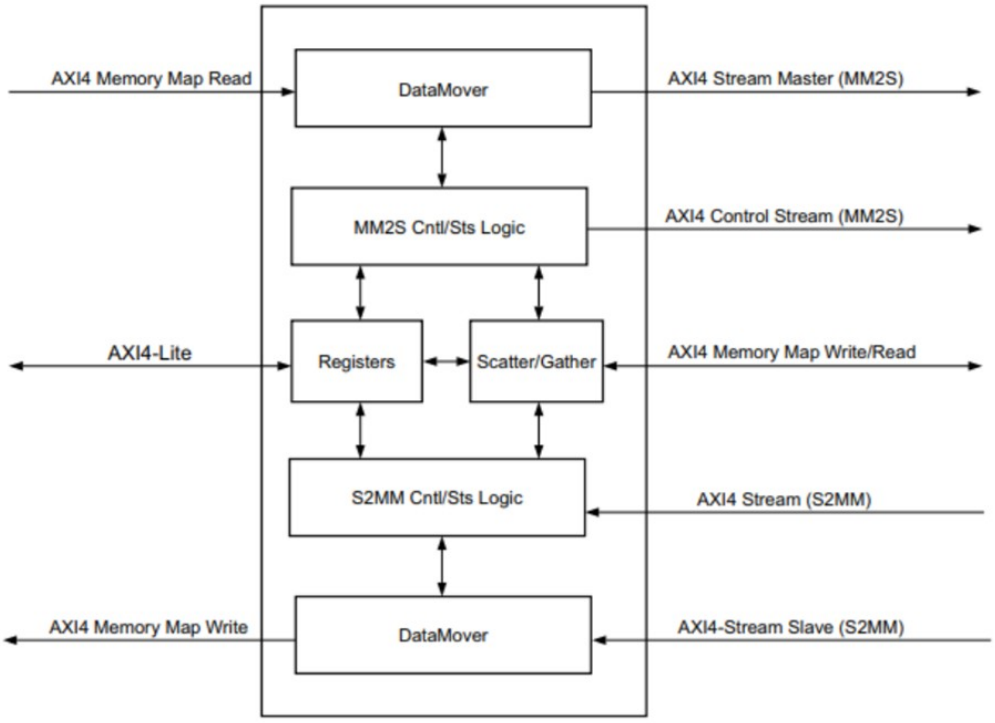
1. AXI DMA (Direct Memory Access)

AXI DMA (Direct Memory Access) kontroler omogućava ulazno/izlaznom uređaju 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 1. 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:

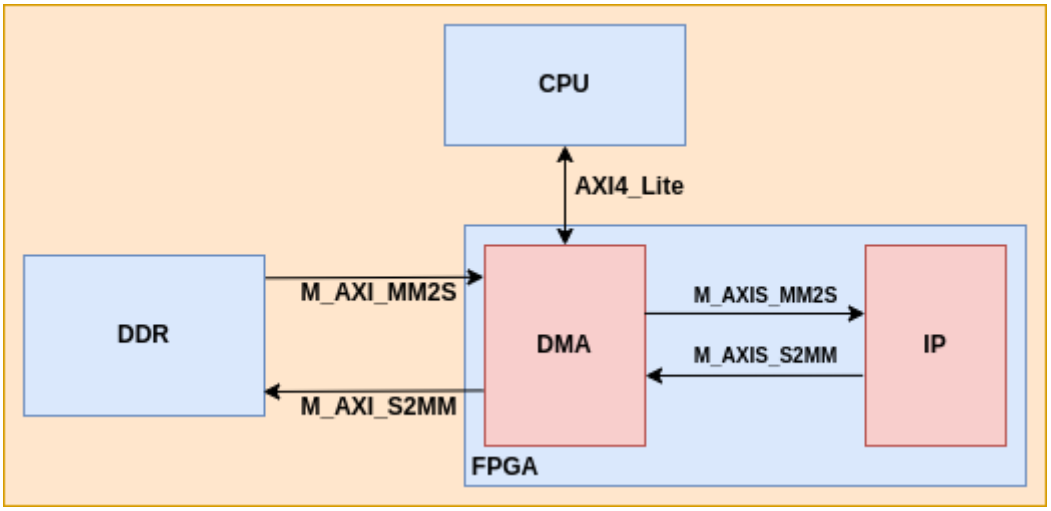
- **Direct Register Mode** (režim direktnog adresiranja) se koristi kada su podaci kojima se pristupa kontinualni u memoriji, odnosno svi podaci se nalaze jedan za drugim.

- **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

Direct Register Mode (režim direktnog adresiranja)

Slika ispod prikazuje generalnu strukturu sistema koji koristi DMA komponentu konfigurisanu u *direct mode* režimu za prenos podataka između DDR memorije i logike na FPGA delu čiša:



Slika 2. Generalna struktura sistema sa DMA komponentom.

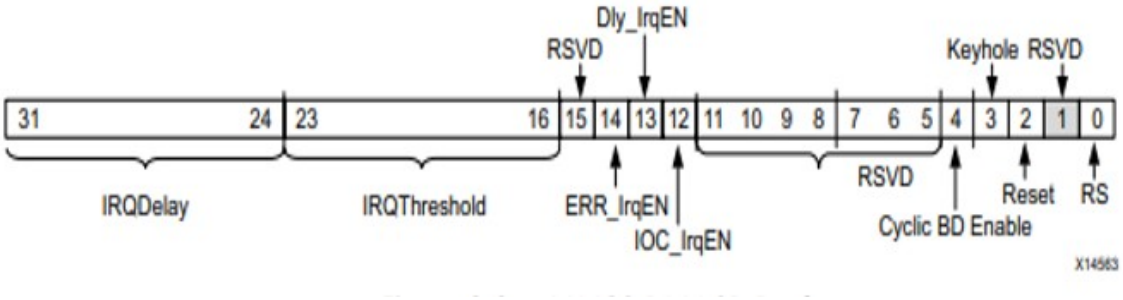
Na slici je prikazano da DMA poseduje sledeće interfejse:

- AXI4_Lite - preko njega procesor konfiguriše DMA jezgro (opisano detaljno u nastavku).
- M_AXI_MM2S - AXI FULL READ kanal preko koga DMA čita podatke iz DDR memorije
- M_AXIS_MM2S - AXI stream master interfejs preko koga DMA prosleđuje podatke pročitane iz DDR memorije logici na FPGA delu čipa.
- M_AXIS_S2MM - AXI stream slave interfejs preko koga DMA prima podatke od logike na FPGA delu čipa.
- M_AXI_S2MM - AXI FULL write kanal preko koga DMA upisuje u DDR memorije podatke prihvaćene od logike na FPGA delu čipa.

U ovom režimu DMA može da šalje podatke iz memorije prema nekom IP jezgru 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.

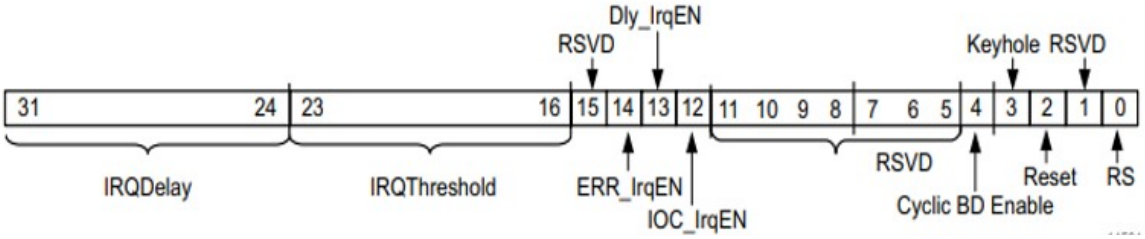
4. Nakon toga potrebno je upisati validnu početnu adresu u RAM memoriji od koje DMA komponenta treba da krene sa prenosom podataka ka IP jezgru. Ta informacija se upisuje u *MM2S_SA* registar, koji je memorijski mapiran na adresu 0x18.
5. Poslednje što je potrebno uraditi jeste da se u registar MM2S_LENGTH 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!**



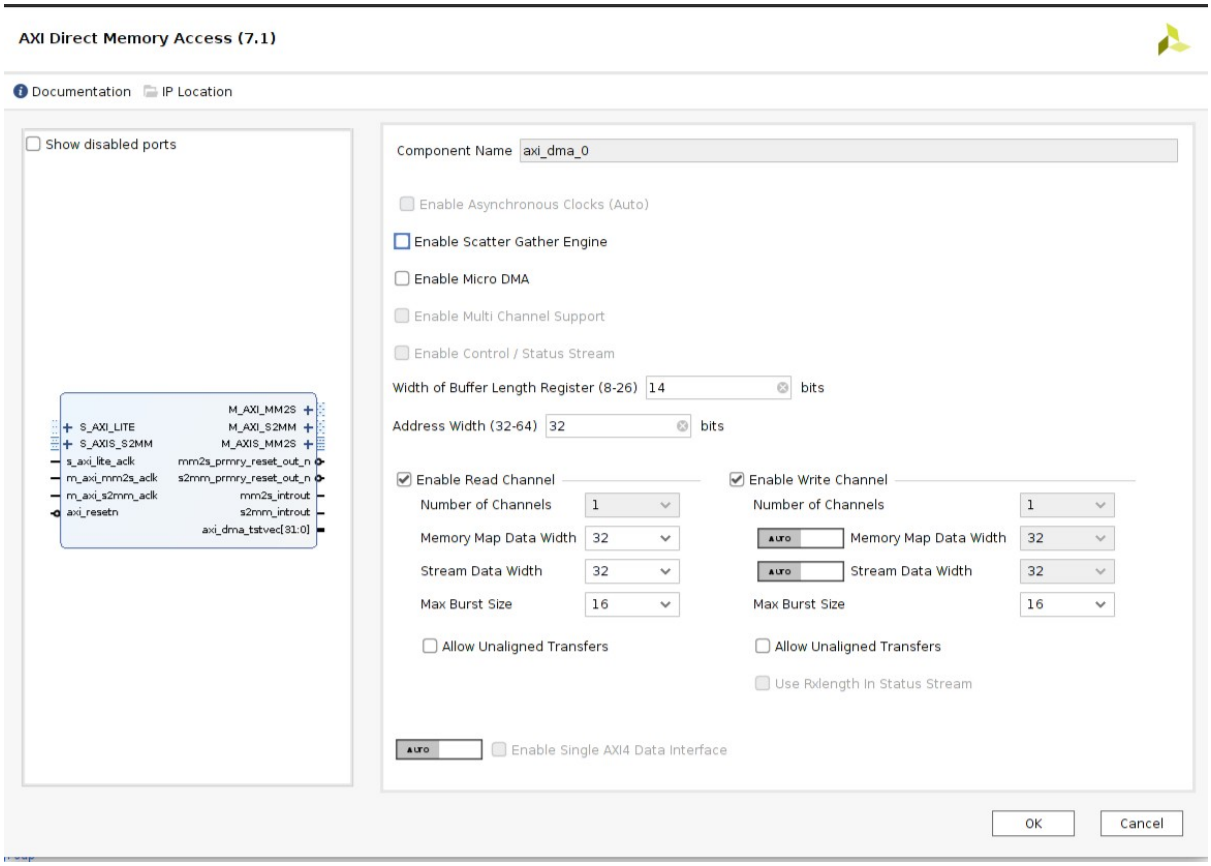
Slika 4. S2MM_DMACR registar

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.

IP integrator

I u ovoj vežbi koristimo već gotov blok (AXI Direct Memory Access) koji dodajemo na isti način kao blokove pomenute na prethodnim vežbama. Da bi on radio u *Direct* modu neophodno ga je konfigurisati. Dvoklikom na blok otvoriti podešavanja i podesiti ga kao na sledećoj slici:

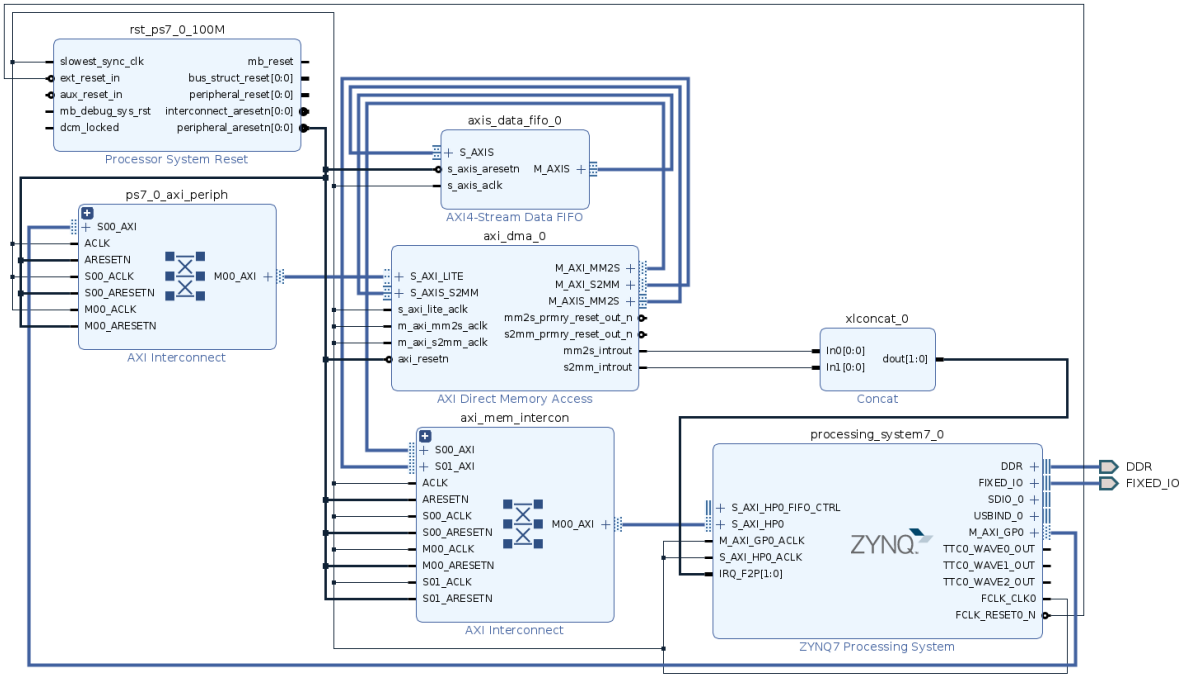


Slika 5. Konfiguracija AXI DMA bloka

Nakon podešavanja, IP blok treba da ima interfejs kao na sledećoj slici:

Slika 6. Interfejs DMA bloka konfigurisano u *Direct mode* režimu

Sledeći primer demonstrira kako se koristi AXI DMA. U ovom primeru je DMA komponenta povezana sa *AXI-Stream Data FIFO* blokom. Ideja je da DMA pročita nešto iz DDR memorije, smesti to u fifo i onda to što je smestio u FIFO opet vrati u DDR.



Slika 7. Povezivanje DMA komponente sa Fifo blokom i Procesorom.

Fifo blok poseduje *AXI stream slave* i *AXI stream master* interfejse i preko njih je povezan sa DMA komponentom. Preko prvog prihvata podatke koje DMA šalje a preko drugog prosleđuje podatka DMA komponenti. Da bi se olakšale stvari, slično kao i ne prethodnim vežbama realizovana je .tcl skripta koja automatski generiše sistem sa prethodne slike. Ta skripta se nalazi u priloženom materijalu na sledećoj putanji:

Lab11_DMA/Vivado_project/

Ukoliko implementirate sistem na ZyboZ710 ploči pokrenuti skriptu **Master_zybo_z7_10.tcl,** a ukoliko implementirate na Zybo ploči pokrenuti **Master_zybo.tcl** skriptu.

Kako bi se ova skripta pokrenula neophodno je otvoriti Vivado i izabrati opciju:

Tools -> Run Tcl Script

I podesiti putanju do skripte.

Vitis

U nastavku je samo *main* deo koda koji konfiguriše DMA tako da pošalje nešto ka FIFO baferu i koji čeka da FIFO nešto vrati. **U priloženom materijalu imate ceo kod u direktorijumu *Vitis_test_files***

```
#define MEM_BASE_ADDR      (DDR_BASE_ADDR + 0x1000000)             // MEM START
ADDRESS
#define TX_BUFFER_BASE     (MEM_BASE_ADDR + 0x00001000)
#define RX_BUFFER_BASE     (MEM_BASE_ADDR + 0x00002000)

u32 *TxBufferPtr = (u32 *)TX_BUFFER_BASE;
u32 *RxBufferPtr = (u32 *)RX_BUFFER_BASE;

int main()
{
  int status;
  int xfer_missmatches =0;
  Xil_DCacheDisable();
  Xil_ICacheDisable();
  init_platform();

  xil_printf("\r\nStarting simulation\r\n");

  //status = Initialize_System();
  //Setup Tx interrupt
  status=Setup_Interrupt(INTC_DEVICE_ID,
(Xil_InterruptHandler)Tx_Interrupt_Handler, TX_INTR_ID);
  //Setup Rx interrupt
  status=Setup_Interrupt(INTC_DEVICE_ID,
(Xil_InterruptHandler)Rx_Interrupt_Handler, RX_INTR_ID);
  DMA_init_interrupts();
  if (status != XST_SUCCESS) return status;

  //Copy image to TxBuffer
  for(int i=0; i<XFER_LENGTH/4; i++){
    TxBufferPtr[i] = i;
    RxBufferPtr[i] = 0;
  }
  //Start first DMA transaction
  DMA_Simple_Write(TxBufferPtr, XFER_LENGTH); //My function that starts a DMA
write transaction
  DMA_Simple_Read (RxBufferPtr, XFER_LENGTH); //My function that starts a DMA
read  transaction

  while (!tx_intr_done && !rx_intr_done)
    {
      xil_printf("Waiting... Interrupt Status -> TX:%d | RX:%d\n\r",
tx_intr_done, rx_intr_done);
      sleep(1);
    }
  //Change values of TxBuffer
  for(int i=0; i<XFER_LENGTH/4; i++){
    if(      TxBufferPtr[i] != RxBufferPtr[i])
      xfer_missmatches ++;
  }
```

```
  if(xfer_missmatches > 0)
    xil_printf("Transfer failed with %d\n\r",xfer_missmatches);
  else
    xil_printf("Transfer successful!!! \n\r");


  sleep(1); //1 sec delay

  cleanup_platform();
  Disable_Interrupt_System();

  return 0;
}
```

Da bi DMA mogao da prebacuje podatke neophodno mu je proslediti odgovarajuće pokazivače. U ovom slučaju su definisana dva pokazivača, TxBufferPtr i RxBufferPtr, jedan pokazuje na niz koji treba poslati iz DDR-a a drugi na niz u koji DMA smešta podatke. **Obratite pažnju u prethodnom kodu da je ovim pokazivačima neophodno dati fizičke bazne adrese.**

U nastavku su opisane funkcije koje se pozivaju kako bi se prethodno pomenuto omogućilo:

- Xil_DCacheDisable - isključuje keširanje podataka. Ovo je nephodno uraditi jer bi u suprotnom DMA upisivao podatke u keš i morali bi da pozivamo dodatne funkcije za kako bi se podaci iz keša prebacili u DDR.
- Xil_ICacheDisable - isključuje keširanje instrukcija.
- Setup_Interrupt - Funkcija koda podešava prekide. U slučaju DMA komponente treba registrovate prekide izazvane i prilikom čitanja podataka iz DDR-a i prilikom upisa podataka u DDR.
- DMA_init_interrupts - Podešava interne registre DMA komponente kako bi se omogućili prekidi. **Ovu funkciju možete modifikovati.**
- DMA_Simple_Write - Pozivom ove funkcije inicira se prenos podataka iz DDR memorije ka logici u FPGA delu čipa. **Ovu funkciju možete modifikovati.**
- DMA_Simple_Read - Pozivom ove funkcije inicira se prenos podataka iz FPGA dela lipa ka DDR memoriji. **Ovu funkciju možete modifikovati.**

Pored ovih funkcija postoje još dve koje se ne vide u main delu koda i on su: Rx_Interrupt_Handler i Tx_Interrupt_Handler.

```
static void Rx_Interrupt_Handler(void *Callback)
{
  //Read irq status from MM2S_DMASR register
  u32 S2MM_DMASR_reg = Xil_In32(DMA_BASEADDR + S2MM_DMASR_OFFSET);

  //Clear irq status in MM2S_DMASR register
  //(clearing is done by writing 1 on 13. bit in MM2S_DMASR (IOC_Irq)
  Xil_Out32(DMA_BASEADDR + S2MM_DMASR_OFFSET, S2MM_DMASR_reg | DMASR_IOC_IRQ);

  rx_intr_done = 1;
  xil_printf("RX Interrupt Happened!\r\n");

}


static void Tx_Interrupt_Handler(void *Callback)
```

```
{
  //In every interrupt handler send the next transaction to continue the cycle

  //Read irq status from MM2S_DMASR register
  u32 MM2S_DMASR_reg = Xil_In32(DMA_BASEADDR + MM2S_DMASR_OFFSET);

  //Clear irq status in MM2S_DMASR register
  //(clearing is done by writing 1 on 13. bit in MM2S_DMASR (IOC_Irq)
  Xil_Out32(DMA_BASEADDR + MM2S_DMASR_OFFSET, MM2S_DMASR_reg | DMASR_IOC_IRQ);

  tx_intr_done = 1;
  xil_printf("TX Interrupt Happened!\r\n");}
```

To su prekidne routine koje se pozivaju nakon što DMA izvrši čitanje podataka iz DDR memorije i upis podataka iz DDR memorije. Unutar njih se podešavaju određeni registri DMA komponente kako bi se omogućilo da se sledeći put desi prekid (Ovo je opisano na početku).