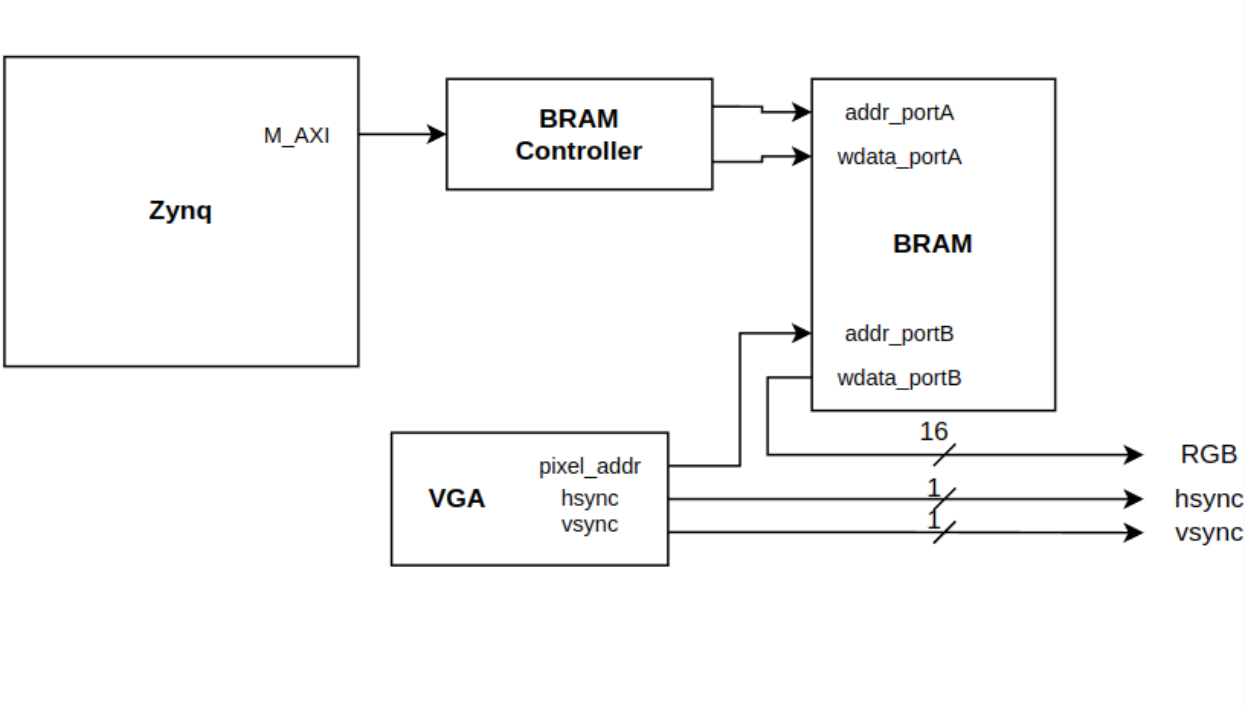
Vežba 11 : VGA BRAM kontroler

1 Opis sistema za generisanje slike na VGA izlazu

Da bi se napisao drajver za sistem čija je uloga da generiše sliku na VGA izlazu, mora se razumeti od kojih komponenti se taj sistem sastoji, kako se sa njim komunicira i koja su mu ograničenja. Na sledećoj slici je prikazana pojednostavljena blok šema jednog takvog sistema:



Slika 1. Blok šema sistema koji sadrži VGA kontroler

Da bi se omogućio ispis slike na ekranu preko VGA izlaza upotrebljene su sledeće komponente:

- BRAM
- BRAM Controller
- AXI_GPIO
- VGA

U **BRAM** memoriju se smešta slika koju je potrebno prikazati na monitoru. Dubina BRAM memorije iznosi 36864 32-bitne lokacije. Razlog za to je rezolucija slike koja iznosi 256*144 piksela. Kako je jedan piksel predstavljen sa 16 bita, to znači da se 16 bita od 32 prilikom čitanja piksela iz bram memorije odbacuje. Zbog ovakvog načina organizacije memorije i zbog toga što je BRAM memorija bajt adresabilna, pojedinačni piksel je pomeren za četiri adresne lokacije u odnosu na piksele oko sebe, odnosno prvi piksel se upisuje na adresu 0x0000, drugi piksel se upisuje na adresu 0x0004, treći na 0x0008 itd. U ovom slučaju BRAM

komponenta je konfigurisana kao dvopristupna memorija (portA i portB), pri čemu se slika upisuje preko porta A, a čita preko porta B.

BRAM Controller komponenta omogućava upis/čitanje iz BRAM memorije. On je neophodan posrednik između Zynq Processing System komponente koja generiše AXI transakcije, i same blok memorije sa kojom se komunicira preko BRAM memorijskog interfejsa. Sa slike 1 se može videti da je BRAM Controller povezan na port A blok memorije, te se koristi samo za upis podataka u memoriju. Blok memorija je uz pomoć BRAM kontrolera memorijski mapirana, te da bi upisali podatak u BRAM, upisujemo na adresu koja je jednaka zbiru početne adrese BRAM kontrolera i ofseta za željeni podatak. T.j. nulta lokacija u blok memoriji je bazna adresa BRAM kontrolera, a svaka sledeća je pomerena za 4 u odnosu na baznu.

VGA komponenta je zadužena za generisanje hsync i vsync signala koji se prosleđuju na *hsync* i *vsync* portove VGA konektora. Ti signali su sinhronizacioni signali koji su neophodni VGA konektoru kako bi znao na kom mestu na monitoru da postavi određeni piksel. VGA komponenta ima i jedan port povezan sa BRAM-om koji generiše adrese piksela koje treba postaviti na određenu lokaciju na monitoru, a BRAM na osnovu tih adresa daje vrednosti piksela na portu data_portB i prosleđuje ih na RGB_out port VGA konektora kao što se može videti na slici 1. VGA komponenta generiše adrese piksela u skladu sa hsync i vsync signalima. Detaljno poznavanje rada ove komponente nije neophodno za pisanje drajvera ovog sistema, te se ova skripta neće upuštati u to. Za one koji žele da znaju više, to mogu naći na sledećem linku:

https://www.elektronika.ftn.uns.ac.rs/uvod-u-mikroracunarsku-elektroniku/wp-content/uploads/sites/134/2018/03/Vezba-11-Projektovanje-slo%C5%BEenih-digitalnih-sistema-VGA.pdf

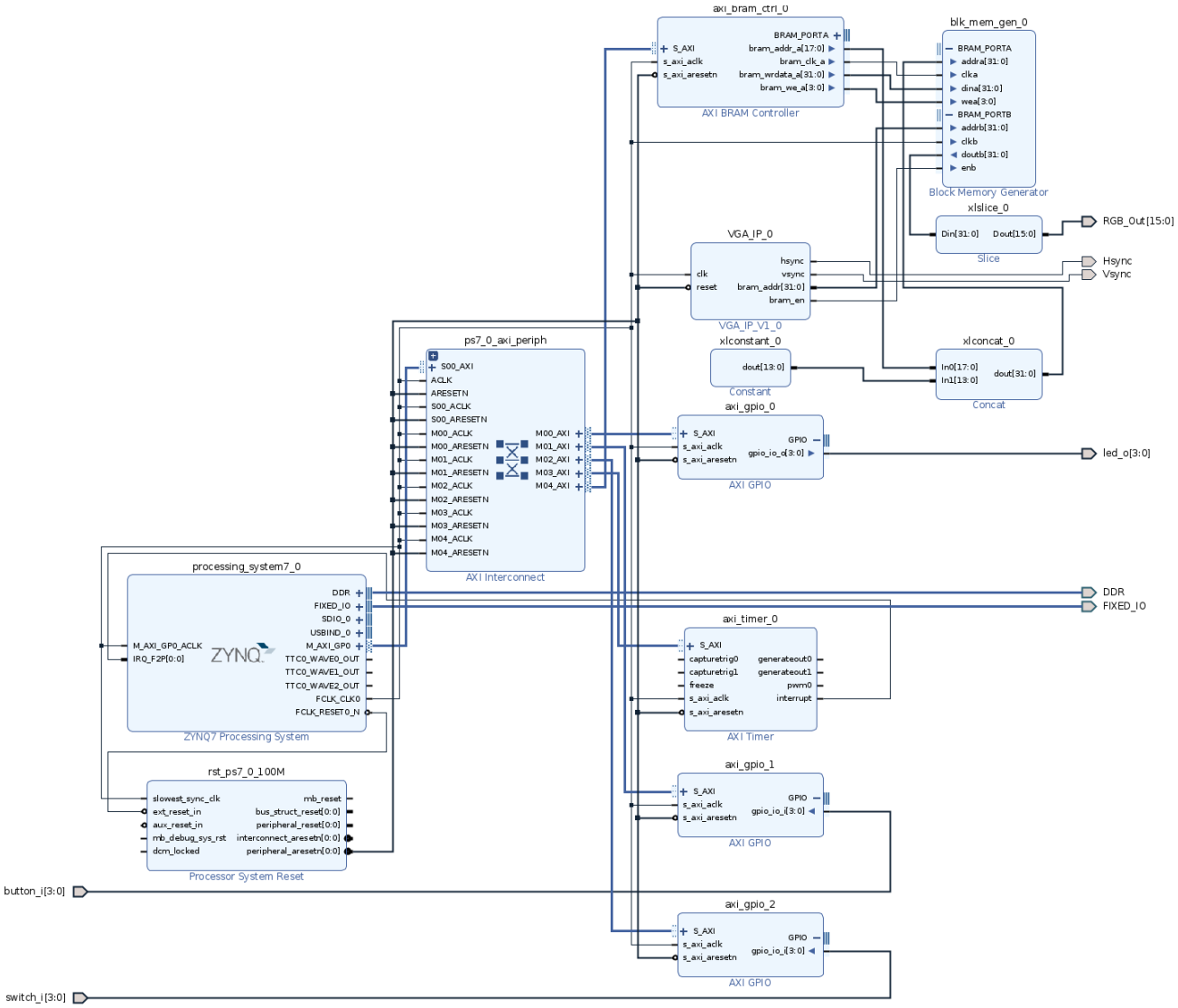
Na slici 1 je prikazana još jedna komponenta označena sa *Zynq* i ona predstavlja sistem za procesiranje, naime ARM procesore sa sistemom za keširanje i ostalim komponentama koje su fiksne i ne pripadaju FPGA delu čipa. Može se primetiti da je on direktno povezan sa BRAM kontrolerom, odnosno da on preko BRAM kontrolera vrši upis u BRAM.

1.

Ograničenje ovog sistema je to da ne može da predstavi sliku čija je rezolucija veća od 256x144 piksela, jer u programabilnoj logici zybo razvojne ploče ne postoji dovoljna količina BRAM-a. Tako da iako VGA kontroler generiše vsync i hsync signale tako da se može predstaviti slika rezolucije 640x480, količina BRAM memorije na zybo razbojnoj ploči ograničava tu rezoluciju na 256x144. Tako da prilikom pisanja drajvera treba voditi računa da se ne prosleđuje slika čiji je broj piksela veći od 36864.

Napomena:

Prethodna blok šema (slika 1) predstavlja pojednostavljeni izgled sistema gde su određeni signali (klok, reset, ...), komponetne i portovi izostavljeni jer nisu bitni za razumevanje njegovog rada. Kompletan blok dijagram prikazan je na slici 2:



Slika 2. Blok šema prethodno opisanog sistema napravljena pomoću Vivado IP integratora

U ovom projektu pored VGA modula, BRMA kontrolera i BRAM memorije, postoje i:

- AXI_GPIO0 - povezan sa četiri LED diode na zybo razvojnoj ploči i upisom u njegov memorijski mapiran registar na adresi 0x0000 menja se stanje na diodama(upisom 0x000f vrednosti sve četiri diode će se upaliti).
- AXI_GPIO1 - povezan sa četiri tastera na zybo razvojnoj ploči i čitanjem iz njegovog registra memorijski mapiranog na adresi 0x0000 se čita stanje tastera(npr. Ukoliko se pročita 0x0000 svi su pušteni, a 0x000f svi su pritisnuti).
- AXI_GPIO2 - povezan sa četiri prekidača na zybo razvojnoj ploči i čitanjem iz njegovog registra memorijski mapiranog na adresi 0x0000 se čita stanje prekidača(npr. Ukoliko se pročita 0x0000 svi su isključeni, a 0x000f svi su uključeni).
- AXI_TIMER_0 - opisan na prethodnim vežbama.

- Ostale komponente su neopohodne za ispravan rad sistema i njih automatski generiše Vivado alat.

Opis GPIO komponente se može naći na sledećem linku: https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf

2 Pisanje Driver-a

Zbog velike sličnosti *led* driver-u (opisanog u vežbi 9), u nastavku je samo prikazan kodni listing *vga_write* funkcije koja se poziva svaki put kada je potrebno postaviti piksel na ekran.

```
static ssize_t vga_write(struct file *f, const char __user *buf, size_t length,
loff_t *off)
{
  char buff[BUFF_SIZE];
  int ret = 0;
  unsigned int xpos=0,ypos=0;
  unsigned long long rgb=0;
  unsigned char rgb_buff[10];
  ret = copy_from_user(buff, buf, length);
  if(ret){
    printk("copy from user failed \n");
    return -EFAULT;
  }
  buff[length] = '\0';


  sscanf(buff,"%d,%d,%s", &xpos, &ypos, rgb_buff);
  ret = kstrtoull(rgb_buff, 0, &rgb);

  if(ret != -EINVAL)//checking if input had rigth format
  {
    if (xpos > 255)
    {
      printk(KERN_WARNING "VGA_write: X_axis position exceeded, maximum is 255 and
minimum 0 \n");
    }
    else if (ypos > 143)
    {
      printk(KERN_WARNING "VGA_write: Y_axis position exceeded, maximum is 143 and
minimum 0 \n");
    }
```

```
    else
    {
      iowrite32(rgb, vp->base_addr + (256*ypos + xpos)*4);

    }
  }
  else
  {
    printk(KERN_WARNING "VGA_write: Wrong write format, expected
\"xpos,ypos,rgb\"\n");
    // return -EINVAL;//parsing error
  }
  return length;

}
```

Pozivom *vga_write* funkcije pojaviće se jedan piksel na ekranu ukoliko je *write* funkciji prosleđen string u ispravnom formatu. Taj format je “x,y,rgb”, gde “x” predstavlja poziciju piksela na x osi, “y” poziciju piksela na y osi i rgb predstavlja vrednost(boju) piksela. Bitno je napomenuti da “x” i “y” parametri moraju biti decimalni brojevi, dok rgb može biti i heksadecimalan i decimalan. *Sscanf* funkcija parsira prosleđeni string, i smešta vrednosti stringa u *x*,*y* i *rgb_buff* promenljive. Može se primetiti da je *rgb_buff* niz karaktera, za razliku od *x* i *y* koji su *integer* tipa i to je urađeno jer *rgb* parametar može biti u heksadecimalnom ili decimalno formatu, te je nad tim stringom potrebno uraditi malo drugačije parsiranje. Za to je zadužena *kstrtoull* funkcija koja vrši konvertovanje *string-a* u *unsigned long long* tip. Njoj je kao drugi parametar prosleđena 0, zbog toga će funkcija sama da shvati koji je format prosleđeng stringa(decimalni, heksadecimalni, ...) i nakon konverzije će rezultat svog izvršavanja smestiti u promenljivu *rgb.* Nakon toga slede provere ispravnosti prosleđenog stringa, odnosno prva provera da li su karakteri stringa pravilni (ako neko pošalje “10, 5b, 0xfff,!” ispisuje se poruka greške), a druga provera je da li su *“*x*”* i “y” u pravilnom opsegu (veličina slike je 256*144, odnosno x mora da bude manji od 256, a y manji od 144). Ukoliko je sve kako treba pozvaće se *iowrite32* funkcija, čime se vrednost piksela u RGB formatu upisuje na pravilnu adresu u blok memoriji.

3 Pisanje test aplikacije

Testiranje ispravnog rada sistema i drajvera možemo uraditi pomoću *echo* komande tako što bi smo slali piksel po piksel i iscrtavali ih na monitoru (npr *echo 20,20,0xf00 >/dev/vga*). Pošto je to malo naporno, napisaćemo aplikaciju koja će iscrtati sledeću sliku na monitoru:

Slika 3. Slika koja se iscrtava na displeju

Naredni kodni listing predstavlja aplikaciju koja iscrtava sliku 3 na monitoru:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "test.h"

int main(void)
{
  FILE* fp;
  int x,y;
  int ret = 0;

  fp = fopen("/dev/vga", "w");
  if(fp == NULL)
    {
      printf("Cannot open /dev/vga for write\n");
```

```
      return -1;
    }
  for(y=0; y<144; y++)
    {
      for(x=0; x<256; x++)
       {
         fprintf(fp,"%d,%d,%#04x\n",x,y,image[y*256+x]);
         fflush(fp);
       }
    }
  if(fclose(fp) == EOF)
    {
      printf("Cannot close /dev/vga\n");
      return -1;
    }
  return 0;
}
```

Slika se nalazi u *test.h* fajlu, i može se videti da je taj fajl uključen u aplikaciju. Na samom početku se otvara *node /dev/vga* preko koga se pikseli šalju drajveru. Nakon toga se pomoću dve *for* petlje šalju pikseli, pri čemu spoljašnja *for* petlja predstavlja “y” koordinatu piksela, a unutrašnja “x” koordinatu. Samo slanje piksela se obavlja pomoću *fprintf* funkcije, pri čemu se prilikom slanja poštuje format koji drajver očekuje. *Fflush* funkcija je zadužena da čim *fprintf* smesti string u tok(*stream*), taj string bude poslat drajveru. Da nje nema string ne bi bio poslat odmah, već bi čekao da se u toku (*stream*) nakupi određena količina informacija i onda bi se sve odjednom poslalo drajveru. U toj situaciji ne bi bio ispoštovan format stringa koji drajver očekuje i slanje ne bi uspelo. Kada se prođe kroz sve iteracije *for* petlji (pošalju svi pikseli) *node /dev/vga* se zatvara i aplikacija se završava.