Проектиране на вградени автомобилни електронни системи
Лабораторно упражнение №22

Работа с Xilinx Vitis HLS. Синтез на система от високо ниво (HLS – High Level Synthesis). Алгоритъм за детекция на контури със Собел оператор. VGA интерфейс.

==========

1. Превключете джъмпера вдясно на платката на позиция JTAG. Свържете µUSB кабел към PROG/UART USB куплунга. Включете платката от ключа ON/OFF. Включете USB-към-UART конвертор към сигнали JB1_P (T20), отговарящ на Uartlite_TxD, и JB1_N (U20), отговарящ на Uartlite_RxD. Модулът Uartlite ще бъде свързан към един MicroBlaze.

ЗАБЕЛЕЖКА: работната маса с платка Zybo Z7-10 трябва да използва куплунг JC и съответно сигналите Uartlite_TxD (V15), Uartlite_RxD (W15).

2. При нова инсталация на операционната система Ubuntu 20.04 трябва да се инсталира програма, с която може да се обработват изображения.

```
sudo apt install imagemagick
```

3. Копирайте изображението от директория 22_3 [1] във вашата директория за синтез от високо ниво. Нека тя бъде в workspaces/vitis_hls_workspace. Извлечете компонентите синьо, червено и зелено в некомпресиран (raw) вид с командата:

```
convert logo_en.jpg -interlace partition RGB:logo_en
```

Ако командата мине успешно, в директорията на проекта ви трябва да са се създали три файла със следните имена:

```
logo_en.R
logo_en.G
logo_en.B
```

За да преобразувате некомпресиран файл във файл с компресия, командата е:

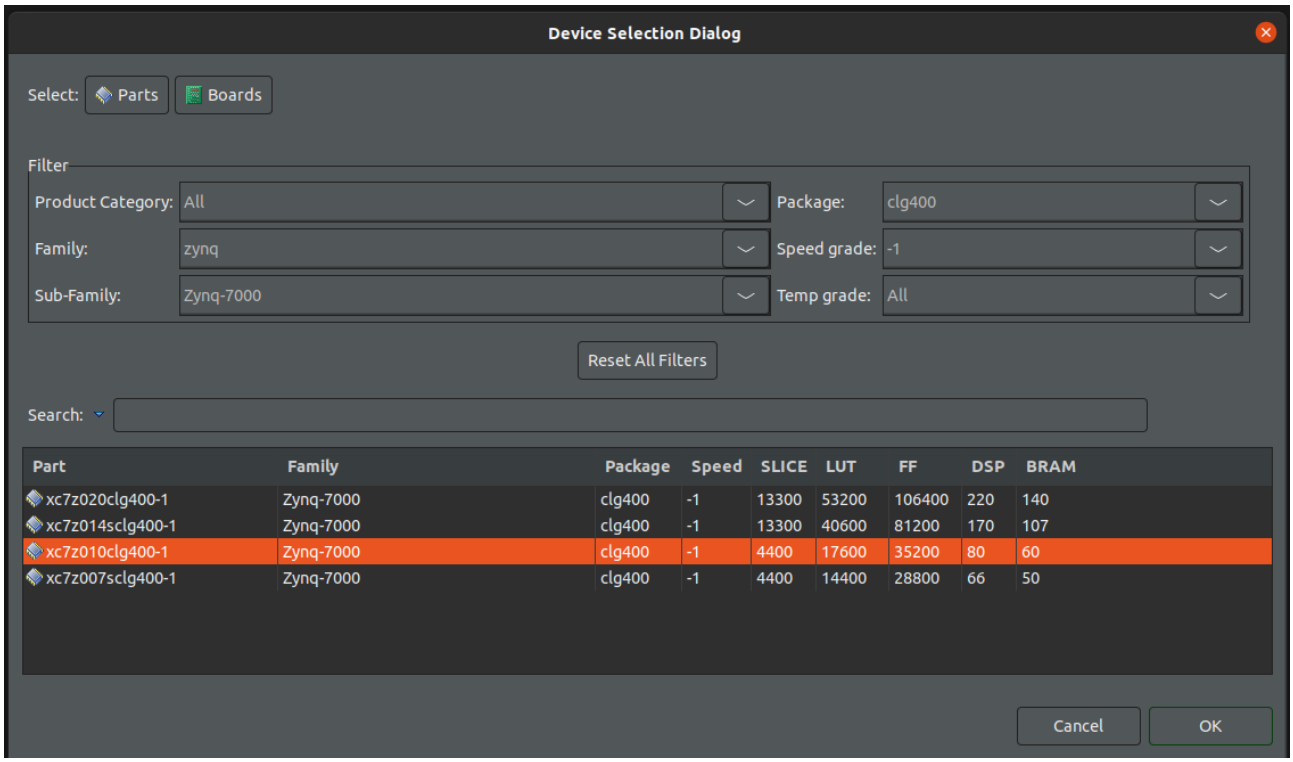
```
convert -depth 8 -size 320x240 gray:logo_en_converted.B
logo_en_converted.jpg
```

За да преобразувате некомпресиран файл в хедърен файл на C, съдържащ масив с байтовете на изображението, използвайте програмата bin_to_array предоставена в директорията на настоящото лабораторно:

```
bin_to_array /path/to/logo_en.B /path/to/logo_en.h
```

4. Създайте проект във Vitis HLS. За целта от страничния панел на Ubuntu изберете бутон “Show Applications”, след което в полето “Type to search” напишете Vitis HLS и натиснете с ляв бутон на мишката иконката на програмата.

5. Изберете File → New Project → Project name: sobel → Location: /home/user/hls_workspace/sobel → Next → Top Function: sobel → Next → Next → Part Selection: натиска се бутонът Browse (бутон с три точки) → Избира се Family: Zynq, Sub family: Zynq-7000, Package: clg400, Speed grade: -1 → от менюто с налични матрици се избира xc7z010clg400-1 → OK → Finish.



6. Вляво → Explorer → десен бутон върху Source → New Source File → sobel.c. В новосъздадения файл се въвежда следния сорс код [2]:

```
#include "sobel.h"

void gradient(const uint8_t *a1, const uint8_t *a2, const uint8_t *a3, const
uint8_t *a4, const uint8_t *a5, const uint8_t *a6, int8_t *output){
      *output = ( ((*a4)+((*a5)<<1)+(*a6)) - ((*a1)+((*a2)<<1)+(*a3)) );
}

void abs_val(const int8_t *x, const int8_t *y, uint8_t *output){
      *output = ((abs(*x)+abs(*y))/4);
}

void threshold(const uint8_t *x, uint8_t *output){
      if( *x > 16 )
            *output = 0;
      else
            *output = 255;
}

void sobel(uint8_t image_in[240][320], uint8_t image_out[240][320]){
#pragma HLS INTERFACE m_axi depth=76800 port=image_out offset=slave
bundle=DATA_IN_OUT_BUS
#pragma HLS INTERFACE m_axi depth=76800 port=image_in offset=slave
bundle=DATA_IN_OUT_BUS
#pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS

      int i, j;
      uint8_t av[240][320];
      int8_t Jx[240][320];
      int8_t Jy[240][320];

      for (j=1; j < 239; j++) {
            for (i=1; i < 319; i++) {
                  gradient(&image_in[j-1][i-1], &image_in[j][i-1],
&image_in[j+1][i-1], &image_in[j-1][i+1], &image_in[j][i+1], &image_in[j+1]
[i+1], &Jx[j][i]);
            }
      }

      for (j=1; j < 239; j++) {
            for (i=1; i < 319; i++) {
                  gradient( &image_in[j-1][i-1], &image_in[j-1][i], &image_in[j-
1][i+1], &image_in[j+1][i-1], &image_in[j+1][i], &image_in[j+1][i+1], &Jy[j]
[i] );
            }
      }

      for (j=1; j < 239; j++) {
            for (i=1; i < 319; i++) {
                  abs_val( &Jx[j][i], &Jy[j][i], &av[j][i] );
                  threshold( &av[j][i], &av[j][i] );
            }
      }

      for (j=1; j < 239; j++) {
            for (i=1; i < 319; i++) {
                  image_out[j][i] = av[j][i];
            }
      }
}
```

7. Вляво → Explorer → десен бутон върху Source → New Source File → sobel.h. В новосъздадения файл се въвежда следния сорс код:

```
#ifndef __SOBEL_H__
#define __SOBEL_H__

#include <stdint.h>
#include <stdlib.h>

void sobel(uint8_t image_in[240][320], uint8_t image_out[240][320]);

#endif
```

8. Вляво → Explorer → десен бутон върху Test Bench → New Test Bench File → sobel_test.c. В новосъздадения файл се въвежда следния сорс код:

```
#include "sobel_test.h"

int main(void){
	uint8_t img_in[240][320];
	uint8_t img_out[240][320];
	FILE *fin, *fout;
	long int file_size;

	fin = fopen("/home/user/workspaces/hls_workspace/sobel/logo_en.B","rb");

	if(fin == NULL){
		printf("Error reading file!");
		return 1;
	}

	fseek(fin, 0L, SEEK_END);
	file_size = ftell(fin);
	rewind(fin);

	printf("\n\n\nFile size is: %ld bytes\n\n\n", file_size);

	fread(&img_in, 1, file_size, fin);

	fclose(fin);

	fout =
fopen("/home/user/workspaces/hls_workspace/sobel/logo_en_converted.B","wb");

	if(fout == NULL){
		printf("Error creating a file for writing!\n");
		return 1;
	}

	sobel(img_in, img_out);

	fwrite(&img_out, 1, file_size, fout);

	fclose(fout);

	return 0;
}
```

9. Вляво → Explorer → десен бутон върху Test Bench → New Test Bench File → sobel_test.h. В новосъздадения файл се въвежда следния сорс код:

```
#ifndef __SOBEL_TEST_H__
#define __SOBEL_TEST_H__

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "sobel.h"

#endif
```

10. Копирайте в workspaces/hls_workspace/sobel файлът с некомпресираното изображение и със синята (например) компонента logo_en.B.

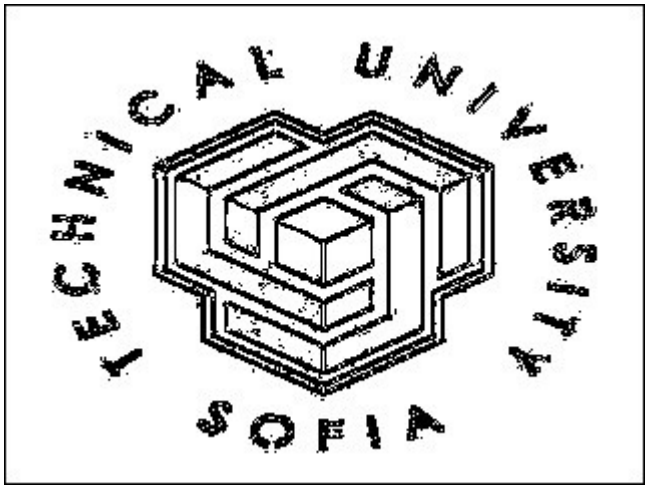
11. От падащото меню на бутона Run Flow (зелена стрелка надясно) → C Simulation → OK. Ако програмата се е изпълнила успешно, в sobel_csim.log трябва да има следните съобщения:

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling(apcc) ../../../../sobel_test.c in debug mode
INFO: [HLS 200-10] Running
'/home/user/programs/xilinx/Vitis_HLS/2022.2/bin/unwrapped/lnx64.o/apcc'
INFO: [HLS 200-10] For user 'user' on host 'computer' (Linux_x86_64 version
5.15.0-87-generic) on Sun Oct 29 21:17:12 EET 2023
INFO: [HLS 200-10] On os Ubuntu 20.04.6 LTS
INFO: [HLS 200-10] In directory
'/home/user/workspaces/hls_workspace/sobel/sobel/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_user/176457169860703228311 3
INFO: [APCC 202-1] APCC is done.
INFO: [HLS 200-112] Total CPU user time: 1.59 seconds. Total CPU system time:
0.11 seconds. Total elapsed time: 1.61 seconds; peak allocated memory: 98.965
MB.
   Compiling(apcc) ../../../../sobel.c in debug mode
INFO: [HLS 200-10] Running
'/home/user/programs/xilinx/Vitis_HLS/2022.2/bin/unwrapped/lnx64.o/apcc'
INFO: [HLS 200-10] For user 'user' on host 'dexter' (Linux_x86_64 version
5.15.0-87-generic) on Sun Oct 29 21:17:14 EET 2023
INFO: [HLS 200-10] On os Ubuntu 20.04.6 LTS
INFO: [HLS 200-10] In directory
'/home/user/workspaces/hls_workspace/sobel/sobel/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_user/176522169860703496578 1
INFO: [APCC 202-1] APCC is done.
INFO: [HLS 200-112] Total CPU user time: 1.59 seconds. Total CPU system time:
0.09 seconds. Total elapsed time: 1.56 seconds; peak allocated memory: 98.980
MB.
   Generating csim.exe

File size is: 76800 bytes

INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

12. Проверява се коректността на филтрираното изображение с:

```
convert -depth 8 -size 320x240 gray:logo_en_converted.B logo_en_converted.jpg
```

След това трябва да се отворят двете изображения и да се сравнят със следните примерни резултати:



13. Преминава се през стъпките [3] Run Flow → C Synthesis → OK, Run Flow → Cosimulation →отметка на Setup Only → OK, Export RTL → OK.

14. Затваря се Vitis HLS. Стартира се Vivado → от страничния панел на Ubuntu изберете бутон “Show Applications”, след което в полето “Type to search” напишете Vivado и натиснете с ляв бутон на мишката иконката на програмата. След това Create Project → Next → Project name: 22_hls → Project location: /home/user/workspaces/vivado_workspace → Next → Next → Boards → избира се демо платката Zybo → Next → Finish.

ЗАБЕЛЕЖКА: работната маса с платка Zybo Z7-10 (вдясно на Етернет куплунга трябва да има 2 HDMI конектора; ако има един HDMI и един VGA значи, че е само Zybo) трябва да избере Zybo Z7-10 от това меню.

15. Вляво → Flow Navigator → IP Catalog → десен бутон върху категорията Vivado Repository → Add Repository … → указва се пътя /home/user/workspaces/hls_workspace/sobel → Select → “One repository was added to the project” → OK.

16. Вляво → Flow Navigator → Create Block Design → OK.

17. Десен бутон в прозореца Diagram → Add IP → ZYNQ7 Processing System.

18. Десен бутон в прозореца Diagram → Add IP → AXI TFT Controller [4], [5], [7].

19. Designer Assistance Available → Run Block Automation → All Automation → OK.
20. Двукратно щракване върху блока ZYNQ7 → Page Navigator → PS-PL Configuration → AXI Non Secure Enablement → GP Master AXI Interface → проверява се дали е избрана опцията M AXI GP0 Interface.

Аналогично в подкатегорията GP Slave AXI Interface се слага отметка на S AXI GP0 Interface.

Page Navigator → Clock Configuration → PL Fabric Clocks → проверява се дали има отметка на FCLK_CLK0 и дали избраната честота е 100 MHz.

Аналогично се избира FCLK_CLK1 и се задава 25 MHz.

Затваря се прозорецът Re-customize IP с бутон OK.

21. Щраква се двукратно върху блок AXI TFT Controller. Избира се TFT Interface: VGA, AXI Data Width: 32, задава се начален адрес на frame buffer-a: Default TFT Base Address: 0x1fe0.0000 (2MB по-надолу от крайния адрес на DDR RAM – 0x1fff.ffff – 0x1f.ffff <2097152 байта> = 0x1fe00000, което позволява максимална резолюция 1920x1080 = 2073600 < 2097152). Натиска се OK.

ВНИМАНИЕ: блокът AXI TFT Controller е конфигуриран за работа с формат RGB666 (6 бита за червено, 6 бита за зелено и 6 бита за синьо), а на ZYBO е реализиран вариантът RGB565. Затова трябва най-старшият бит на червеното и синьото трябва да се игнорират. Това може да стане по два начина:
*след създаването на HDL Wrapper-a да се намалят броя на сигналите на съответния VGA порт;
*сигналите да бъдат мултиплексирани на изводи, които не се използват.
В настоящото лабораторно е избран втория подход, понеже първия има един съществен недостатък – HDL Wrapper-a е автоматично генериран файл и всички корекции, направени на ръка, в него изчезват при промяна на блоковата схема.

22. Добавя се блокът, проектиран с Vitis HLS → десен бутон върху прозорец Diagram → Add IP → sobel → двукратно щракване.

23. Designer Assistance Available → Run Connection Automation → All Automation → OK.

24. Натиска се бутон Regenerate Layout.

25. Свързва се сигналът FCLK_CLK1 на ZYNQ7 Processing System със сигналът sys_tft_clk на AXI TFT Controller.

26. Понеже sobel IP модулът използва масиви като входни и изходни портове, във Vitis HLS беше избрано да се използват регистри от външната DDR RAM памет. Ако матрицата съдържа много BRAM клетки, може масивите да са на самата матрица, и тогава ще се получи най-бърз хардуерен ускорител. Настоящата матрица, обаче, има само 60 BRAM клетки и за нея външната DDR DRAM е единствената опция. Портовете:
*s_axi_CTRL_BUS
*s_axi_control

са подчинени портове и се използват за контрол на IP модула. Това включва битове start, done, ready, idle, interrupt, interrupt enable и други, както и регистри за задаване на базови адреси от DDR DRAM, които ще се използват за начало на масивите. Портът:
*m_axi_DATA_IN_OUT_BUS

е главен порт и се използва за свързване с външна DDR DRAM. Той се използва за трансфер на данните, които ще бъдат обработвани от IP модула [6].

За да може sobel модулът да достъпва DDR DRAM, той трябва да е свързан със ZYNQ7 блока, откъдето да си комуникира с DDR DRAM контролера на FPGA чипа. Затова трябва да се щракне двукратно върху AXI Interconnect модулът, който свързва TFT контролера с S_AXI_GP0 порта на ZYNQ7 блока → Number of slave interfaces: 2 → OK.

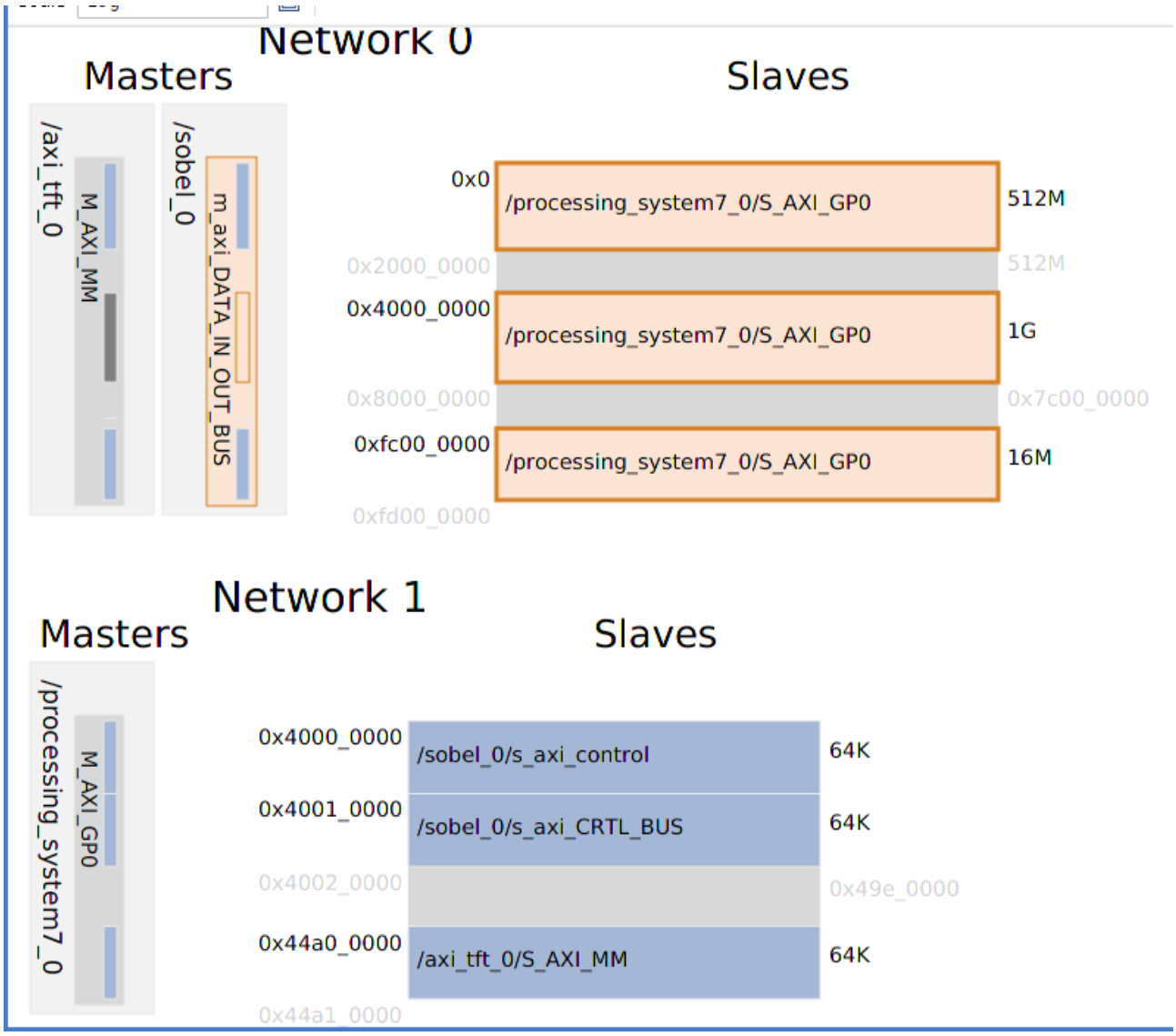
27. Свързва се m_axi_DATA_IN_OUT_BUS на sobel с S01_AXI на AXI Interconnect.

28. Изкарват се сигналите на VGA порта → щраква се еднократно върху порта VGA_INTF на AXI TFT Controller. След това се щраква с десен бутон и се избира Make External на следните сигнали:

tft_vga_b[5:0]
tft_vga_g[5:0]
tft_vga_r[5:0]
tft_hsync
tft_vsync

29. Отваря се таб Address Editor → Assign All → OK. Отваря се категорията Network 0 → sobel_0 → sobel_0/Data_m_axi_DATA_IN_OUT_BUS → Excluded → десен бутон върху GP0_M_AXI_GP0 → Include.

Картата на паметта трябва да изглежда така:



30. Натиснете бутон Validate Design в таб Diagram.

В някои версии на Vivado е възможно да се появят предупредителни съобщения, относно отрицателни стойности на параметрите DDR_DQS_TO_CLK_DELAY_x, но те могат да се игнорират в конкретния дизайн.

31. Таб Sources → десен бутон върху Constraints → Add Sources → Add or create constraints → Next → Create file → дава се името 22_hls_constraints → OK → Finish.

32. Отваря се току-що създадения файл от категория Constraints/constrs_1 чрез двукратно щракване върху 22_hls_constraints.xdc. Въвеждат се следните tcl команди:

```
##VGA Connector
##IO_L7P_T1_AD2P_35
set_property PACKAGE_PIN M19 [get_ports {tft_vga_r_0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_r_0[0]}]

##IO_L9N_T1_DQS_AD3N_35
set_property PACKAGE_PIN L20 [get_ports {tft_vga_r_0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_r_0[1]}]

##IO_L17P_T2_AD5P_35
set_property PACKAGE_PIN J20 [get_ports {tft_vga_r_0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_r_0[2]}]

##IO_L18N_T2_AD13N_35
set_property PACKAGE_PIN G20 [get_ports {tft_vga_r_0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_r_0[3]}]

##IO_L15P_T2_DQS_AD12P_35
set_property PACKAGE_PIN F19 [get_ports {tft_vga_r_0[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_r_0[4]}]

##IO_L14N_T2_AD4N_SRCC_35
set_property PACKAGE_PIN H18 [get_ports {tft_vga_g_0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_g_0[0]}]

##IO_L14P_T2_SRCC_34
set_property PACKAGE_PIN N20 [get_ports {tft_vga_g_0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_g_0[1]}]

##IO_L9P_T1_DQS_AD3P_35
set_property PACKAGE_PIN L19 [get_ports {tft_vga_g_0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_g_0[2]}]

##IO_L10N_T1_AD11N_35
set_property PACKAGE_PIN J19 [get_ports {tft_vga_g_0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_g_0[3]}]

##IO_L17N_T2_AD5N_35
set_property PACKAGE_PIN H20 [get_ports {tft_vga_g_0[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_g_0[4]}]
```

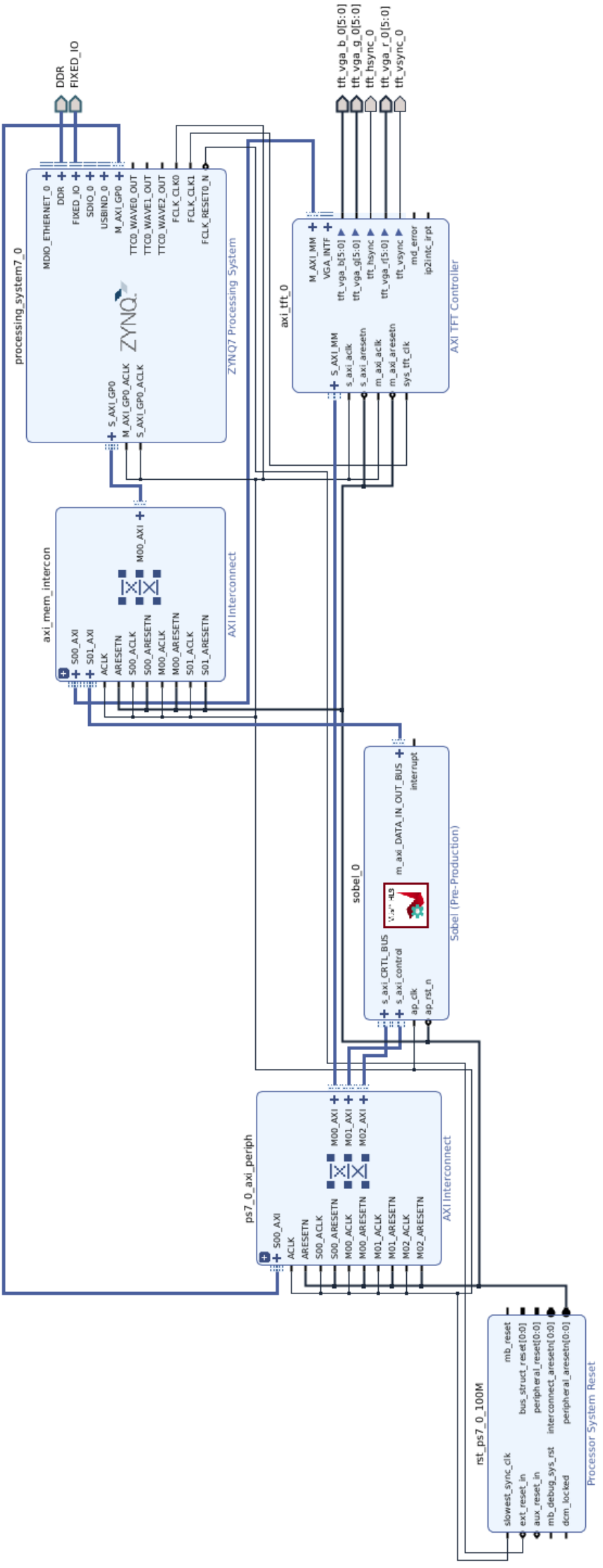
```
##IO_L15N_T2_DQS_AD12N_35
set_property PACKAGE_PIN F20 [get_ports {tft_vga_g_0[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_g_0[5]}]

##IO_L14N_T2_SRCC_34
set_property PACKAGE_PIN P20 [get_ports {tft_vga_b_0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_b_0[0]}]

##IO_L7N_T1_AD2N_35
set_property PACKAGE_PIN M20 [get_ports {tft_vga_b_0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_b_0[1]}]

##IO_L10P_T1_AD11P_35
set_property PACKAGE_PIN K19 [get_ports {tft_vga_b_0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_b_0[2]}]

##IO_L14P_T2_AD4P_SRCC_35
set_property PACKAGE_PIN J18 [get_ports {tft_vga_b_0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_b_0[3]}]

##IO_L18P_T2_AD13P_35
set_property PACKAGE_PIN G19 [get_ports {tft_vga_b_0[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_b_0[4]}]

##IO_L13N_T2_MRCC_34
set_property PACKAGE_PIN P19 [get_ports tft_hsync_0]
set_property IOSTANDARD LVCMOS33 [get_ports tft_hsync_0]

##IO_0_34
set_property PACKAGE_PIN R19 [get_ports tft_vsync_0]
set_property IOSTANDARD LVCMOS33 [get_ports tft_vsync_0]

#Route R and B most significant bits to the JB header
#just to pass the synthesis
set_property PACKAGE_PIN V20 [get_ports {tft_vga_r_0[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_r_0[5]}]

set_property PACKAGE_PIN W20 [get_ports {tft_vga_b_0[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {tft_vga_b_0[5]}]
```

Натиска се CTRL + s от клавиатурата.

33. Прозорец Sources → Design Sources → десен бутон върху design_1 → Create

HDL Wrapper → Let Vivado manage wrapper and auto-update → OK. Блоковата схема е показана на следващата страница.

34. Вляво → Flow navigator →



Generate bitstream → Yes → OK → изчаква се няколко минути (докато завърши синтеза) → View reports → OK

ВНИМАНИЕ: долу, централно, в таб Log може да наблюдавата съобщенията от синтеза. Най-горе, вдясно на Vivado прозореца ще видите иконка на въртящ се зелен часовник. Докато тя е видима, значи трябва да се изчака.

35. File → Export → Export hardware → Next → Include bitstream → Next → Next → Finish

==========
==========

36. Tools → Launch Vitis IDE

37. Избира се път до workspace за фърмуерния проект→ Launch

ВНИМАНИЕ: възможно е да има останали фърмуерни проекти от минали групи. В таб-а Explorer на средата Vitis със задържане на CTRL от клавиатурата изберете с ляв бутон на мишката всички проекти, след което натиснете десен бутон на мишката и Delete. Ако проектите ще се използват, махнете отметката от “Delete project contents on disk (cannot be undone)” и натиснете OK.

38. File → New → Platform project → Platform project name: 22_hls_pla → Next → таб "Create new platform from hardware" → Browse → избира се пътя до проекта 22_hls_pla, създаден от Vivado → design_1_wrapper.xsa → Open → избира се Processor: ps7_cortex_a9_0→ Finish.

39. Вляво→Project explorer → избира се 22_hls_pla → right-click → Build Project.

40. File → New → Application project → Next → "Select a platform from repository" → Избира се 22_hls_pla → Next → Application project name: 22_hls_app → Next → Next → “Hello World” → Finish.

41. Копирайте файлът logo_en.h (създаден с bin_to_array) в директорията, която съдържа helloworld.c файла.

42. Отворете helloworld.c и въведете следната програма:

```c
#include "platform.h"
#include "xil_printf.h"
#include "xil_cache.h"
#include "xtft.h"
#include "xparameters.h"
#include "xuartps_hw.h"
#include "sleep.h"
#include "logo_en.h"
#include "xsobel.h"

#define TFT_FRAME_ADDR      XPAR_PS7_DDR_0_S_AXI_HIGHADDR - 0x001FFFFF
#define IN_IMAGE_LOW_ADDR     TFT_FRAME_ADDR - 0x19000
#define OUT_IMAGE_LOW_ADDR    IN_IMAGE_LOW_ADDR - 0x19000

XTft TFT0;

volatile uint8_t *sobel_base_in = (volatile uint8_t *)IN_IMAGE_LOW_ADDR;
volatile uint8_t *sobel_base_out = (volatile uint8_t *)OUT_IMAGE_LOW_ADDR;

int main(){
      int i = 0;
      XTft_Config *tft_config;

      init_platform();

      print("Starting Sobel demo ...\n\r");

      Xil_DCacheDisable();

      tft_config = XTft_LookupConfig(XPAR_TFT_0_DEVICE_ID);
      XTft_CfgInitialize(&TFT0, tft_config, tft_config->BaseAddress);

      while (XTft_GetVsyncStatus(&TFT0) != XTFT_IESR_VADDRLATCH_STATUS_MASK);

      XTft_SetFrameBaseAddr(&TFT0, TFT_FRAME_ADDR);
      XTft_ClearScreen(&TFT0);

      memset((uint32_t *)0x1fe00000, 0xff, 0x1fffff);

      XTft_EnableDisplay(&TFT0);

      for(u32 y_coord = 0; y_coord < 240; y_coord++){
            for(u32 x_coord = 0; x_coord < 320; x_coord++){
                  XTft_SetPixel(&TFT0, x_coord, y_coord, logo_en[i++]);
            }
      }

      usleep(20000000);

      XTft_DisableDisplay(&TFT0);

      while(1){ }

      cleanup_platform();

      return XST_SUCCESS;
}
```

43. Свържете монитор към VGA куплунга на ZYBO. Заредете програмата във FPGA с десен бутон върху 22_hls_app_system → Debug As → Launch Hardware. Натиснете бутон F8 (Resume). На монитора би трябвало да се появи нефилтрираното изображение.

44. Извикайте функциите за инициализация, стартиране и изчакване за завършване на Sobel филтрацията. Тествайте дали IP модула работи. За целта разгледайте функциите на драйвера за вашия custom IP модул, който се намира в 22_hls_pla/export/22_hls_pla/hw/drivers/sobel_v1_0/src/xsobel.c

* * *

[1] https://oriona.bg/431/vektorizirane-na-logo-na-tu-sofiya
https://github.com/narendiran1996/vga_controller

[2] L. Bogdanov, H. Nikolov, T. Stefanov, “Embedded Multi-processor Systems-on-Chip: Laboratory Experiments and Users Manual”, ISBN: 978-619-167-034-5, Technical University of Sofia, 2013.

[3] Chathura Rajapaksha, “Xilinx Vivado HLS Beginners Tutorial : Custom IP Core Design for FPGA”, online, 2017.

https://medium.com/@chathura.abeyrathne.lk/xilinx-vivado-hls-beginners-tutorial-custom-ip-core-design-for-fpga-59876d5a4119

https://medium.com/@chathura.abeyrathne.lk/xilinx-vivado-hls-beginners-tutorial-integrating-ip-core-into-vivado-design-7ddea40c9b7e

[4] https://wiki.nus.edu.sg/display/ee4218/%5BOptional%5D+Adding+a+Display

[5] Mohamad Oussayran, “Video Processing 2: Display pattern via VGA output on ZYBO”, online, 2023.

[6] https://byu-cpe.github.io/ecen427/labs/hls-accelerator/
[7] https://narendiran1996.github.io/project-blogs/jekyll/update/2020/08/14/vgaController.html

доц. д-р инж. Любомир Богданов, 2024 г.