



Проектиране на вградени автомобилни електронни системи

Лабораторно упражнение №15

Работа с Xilinx Vivado и Vitis. Многопроцесорна система с персонализиран IP модул на Verilog. Синхронизация чрез прекъсвания.

=====

1. Превключете джъмпера вдясно на платката на позиция JTAG. Свържете μ USB кабел към PROG/UART USB куплунга. Включете платката от ключа ON/OFF.

2. 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.

3. Create Project → Next → Project name: 15_custom_ip_verilog → Next → RTL Project + "Do not specify sources at this time" → Next → таб Boards: избира се Zybo (не Zybo Z7-10, не Zybo Z7-20, а само 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.

4. Tools → Create and Package New IP → Next → Create a new AXI4 peripheral → Next → Name: abs_gradient_out →

Name: S00_AXI

Interface Type: Lite

Interface Mode: Slave

Data Width (Bits): 32

Memory Size (Bytes): 64

Number of Registers: **16**

→ Next → Edit IP → Finish

5. Ще се проектира IP модул с 3 функции:

* изчисление на градиент в монохромни изображения (рег.: 6 входа + 1 изход)

* изчисление на средна стойност и модул на две числа (рег.: 3 входа + 1 изход)

* GPIO-lite модул с няколко изхода (1 регистър)

* Възможност за генериране на прекъсване (1 регистър)

Общо: 13 регистъра

6. В новоотвореният се прозорец на Vivado → таб Sources → двукратно щракнете върху abs_gradient_out_v1_0 (abs_gradient_out_v1_0.v)(1) → след това двукратно щракнете върху abs_gradient_out_v1_0_S00_AXI_inst : abs_gradient_out_v1_0_S00_AXI (abs_gradient_out_v1_0_S00_AXI.v) →

* abs_gradient_out_v1_0 – top-level описание на новия модул;

* abs_gradient_out_v1_0_S00_AXI_inst – същинското описание на новия модул, където се въвежда кода на Verilog.

7. В abs_gradient_out_v1_0_S00_AXI_inst се търси коментатър *// Users to add ports here* и се добавя [1], [2]:

```
// Users to add ports here
output wire interrupt_0,
output wire interrupt_1,
output wire gpio_out_0,
output wire gpio_out_1,
// User ports ends
```

8. В abs_gradient_out_v1_0_S00_AXI_inst се търси коментатър *// Add user logic here* и се добавя:

```
// Add user logic here
assign interrupt_0 = slv_reg11[0:0];
assign interrupt_1 = slv_reg11[1:1];
assign gpio_out_0 = slv_reg12[0:0];
assign gpio_out_1 = slv_reg12[1:1];

reg [C_S_AXI_DATA_WIDTH-1:0] buffer_0;
reg [C_S_AXI_DATA_WIDTH-1:0] buffer_1;

always @(posedge S_AXI_ACLK)
begin
    slv_reg6 <= (slv_reg3 + (slv_reg4 << 1) + slv_reg5) - (slv_reg0 + (slv_reg1 << 1) + slv_reg2);

    if(slv_reg7[31] == 1'b1)
    begin
        buffer_0 <= -slv_reg7;
    end
    else
    begin
        buffer_0 <= slv_reg7;
    end

    if(slv_reg8[31] == 1'b1)
    begin
        buffer_1 <= -slv_reg8;
    end
    else
    begin
        buffer_1 <= slv_reg8;
    end

    slv_reg10 = (buffer_0 + buffer_1)/slv_reg9;
```

```
end
// User logic ends
```

9. Регистри 6 и 10 се използват само като изходи. В темплейтът, който Vivado е генерирал, те се достъпват от няколко места. Това ще доведе до хардуерен конфликт – при синтеза ще излезе грешката “slv_reg6 has multiple drivers”. Затова се налага **да се коментират** всички редове в `abs_gradient_out_v1_0_S00_AXI_inst.v`, където тези два регистъра участват в лявата страна на присвояване. Това са следните редове:

```
//slv_reg6 <= 0;
//slv_reg10 <= 0;

//slv_reg6[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
//slv_reg10[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];

//slv_reg6 <= slv_reg6;
//slv_reg10 <= slv_reg10;
```

Натиснете CTRL + s, за да се запазят промените във Verilog описанието.

10. Таб `abs_gradient_out_v1_0.v` → четирите сигнала на модула за прекъсване и GPIO изходи се записват в top-level описанието със същите имена. Това става като в *//Users to add ports here* се добавят редовете:

```
// Users to add ports here
output wire interrupt_0,
output wire interrupt_1,
output wire gpio_out_0,
output wire gpio_out_1,
```

както и в *// Instantiation of Axi Bus Interface S00_AXI* се добавят:

```
// Instantiation of Axi Bus Interface S00_AXI
abs_gradient_out_v1_0_S00_AXI # (
    .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
    .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
) abs_gradient_out_v1_0_S00_AXI_inst (

    ....
    .interrupt_0(interrupt_0),
    .interrupt_1(interrupt_1),
    .gpio_out_0(gpio_out_0),
    .gpio_out_1(gpio_out_1)
);
```

Натиснете CTRL + s, за да се запазят промените във Verilog top-level описанието.

11. Таб Package IP – abs_gradient_out → Packaging Steps: File Groups → натиснете синият ред текст “Merge changes from File Groups Wizard”. Аналогично в Packaging Steps: Customization parameters → “Merge changes from Customization Wizard”.
Аналогично в Packaging Steps: Review and Package → бутон “Re-Package IP” → Do you want to close the project → Yes.
12. Вляво → Flow navigator → Create block design → OK.
13. Вдясно → Diagram → right-click → Add IP → Search → ZYNQ7 Processing System → double click.
14. Вдясно → Diagram → натиска се и се задържа ляв бутон върху FCLK_CLK0 сигнала и се свързва с M_AXI_GP0_ACLK, след това се пуска левия бутон.
15. Вдясно → Diagram → right-click → Add IP → Search → Processor System Reset → double click.
16. Вдясно → Diagram → right-click → Add IP → Search → MicroBlaze → double click.
17. Вдясно → Diagram → зелена лента → Designer Assistance available -> Run Block Automation → Predefined configurations → Select Configuration: Microcontroller preset → слага се отметка на "microblaze_0" и “zynq_0” → Local memory: 64 kB и Cache configuration: None.
18. Слага се отметка на Interrupts. Натиска се OK.
19. Вдясно → Diagram → right-click → Add IP → Search → AXI GPIO → double click.
20. Щракнете двукратно върху axi_gpio_0 → IP Interface GPIO → Board Interface : leds 4bits → OK.
21. Щракнете двукратно върху AXI Interconnect, който свързва MicroBlaze с контролерът му на прекъсванията. Изберете Number of master interfaces: 2.
22. Свържете S_AXI на axi_gpio_0 с M01_AXI на microblaze_0_axi_periph interconnect-a.
23. Вдясно → Diagram → right-click → Add IP → Search → abs_gradient_out → double click.

24. Свързват се M_AXI_GP0 на processing_system7_0 с S00_AXI на abs_gradient_out_0 през блок AXI Interconnect (сложете го с десен бутон → Add IP → AXI Interconnect → двойно щракване, след това двойно щракване върху самия блок → избира се Number of Slave interfaces: 1 и Number of Master interfaces: 1 → OK).

25. Свързват се сигналите interrupt_0 и interrupt_1 на abs_gradient_out_0 към новопоявилият се (по време на Run Block Automation на MicroBlaze) контролер на прекъсванията, към който има свързан блок Concat с 2 входа. Ако входовете не са два, трябва да се щракне двукратно върху него и да се избере Number of ports: 2.

26. Десен бутон върху gpio_out_0 на abs_gradient_out_0 → Make External. Аналогично се прави и за gpio_out_1.

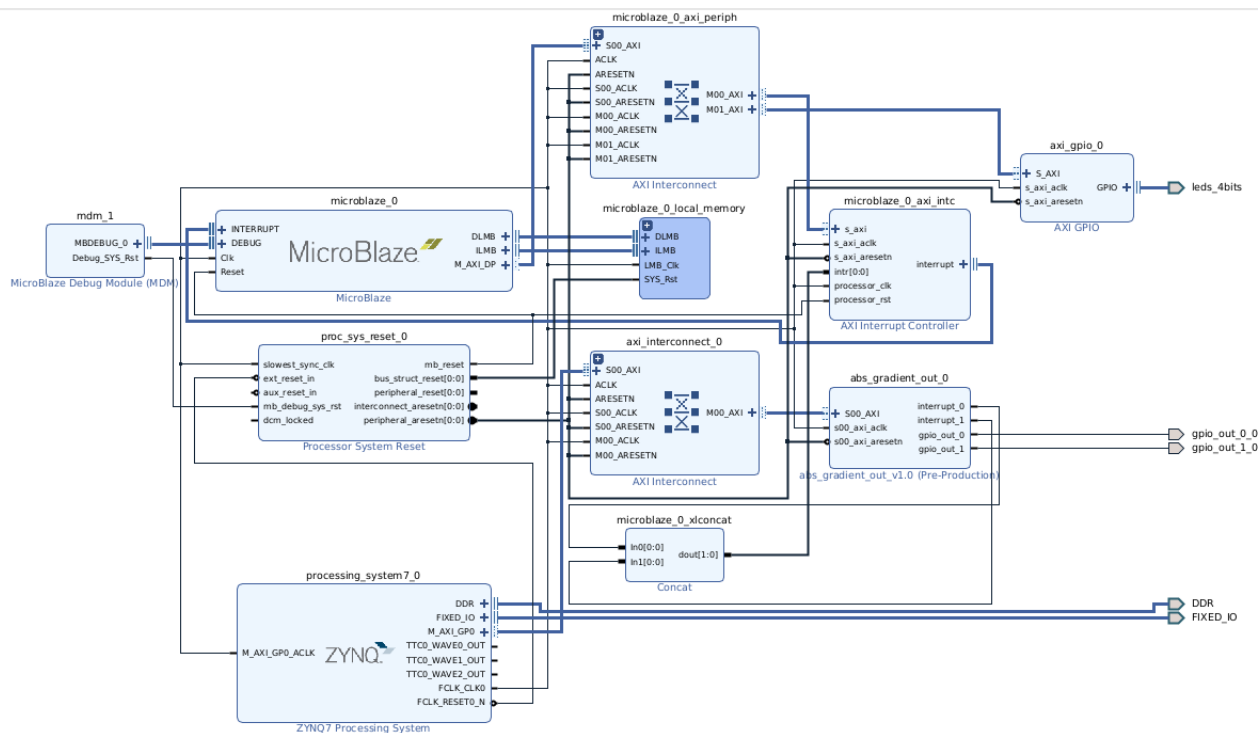
27. Вдясно → Diagram → зелена лента → Designer Assistance available -> Run Connection Automation → Слага се отметка на "All Automation". Натиска се OK.

28. Щраква се два пъти върху блока "ZYNQ7 Processing System" → в "Page navigator" → MIO Configuration → в раздел I/O Peripherals → UART1 се проверяват връзките MIO48 ↔ tx, MIO49 ↔ rx.

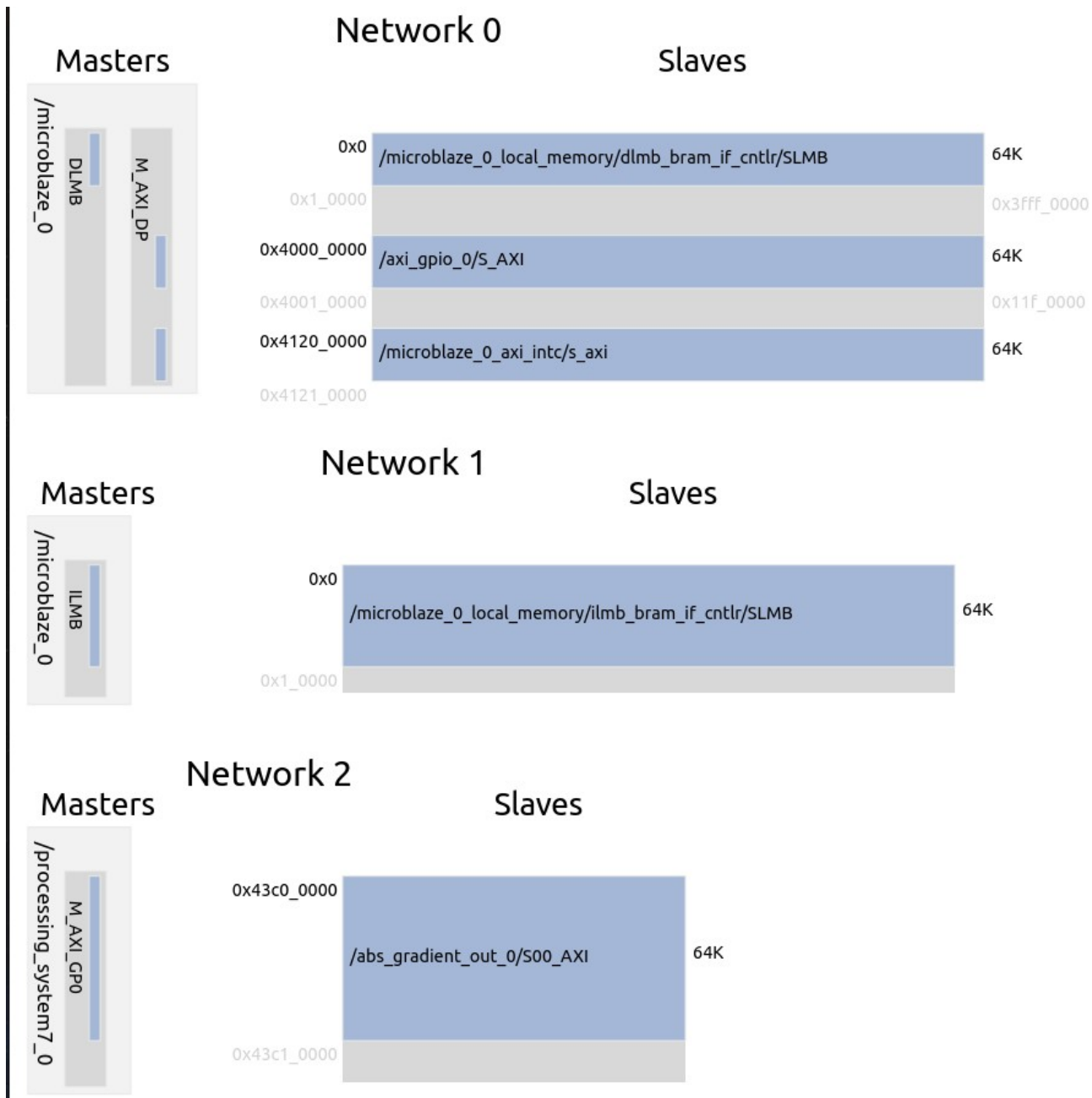
29. В същия прозорец → "Page navigator" → MIO Configuration → маха се отметката на I/O Peripherals → ENET0, USB0 и SD0.

30. Подрежда се блоковата схема с бутон Regenerate Layout. Блоковата схема на системата е показана на следващата страница. Структурата на персонализираният модул е следната:

slv_reg0 – gradient_0 [in]
slv_reg1 – gradient_1 [in]
slv_reg2 – gradient_2 [in]
slv_reg3 – gradient_3 [in]
slv_reg4 – gradient_4 [in]
slv_reg5 – gradient_5 [in]
slv_reg6 – gradient_out [out]
slv_reg7 – abs_in0 [in]
slv_reg8 – abs_in1 [in]
slv_reg9 – abs_div [in]
slv_reg10 – abs_out [out]
slv_reg11 – interrupt [in]
slv_reg12 – GPIO [out]



31. В основния прозорец на Vivado, до таб Diagram, се избира Address Editor → натиска се бутон Assign All. Тази стъпка разполага периферните модули на системата в адресното поле на съответните микропроцесори. Отворете таб Address Map и се уверете, че персонализираният модул `abs_gradient_out` е поместен в картата на паметта на ARM Cortex A9.



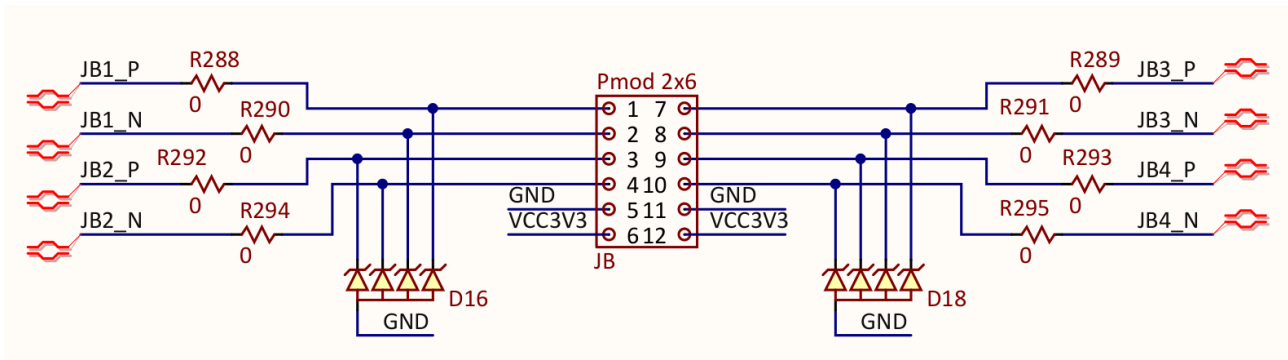
ВНИМАНИЕ: ако по време на автоматизацията на блоковете Vivado е избрал 8 kB за програмна и даннова памет на MicroBlaze, променете тези стойности на 64 kB от таб Address editor (съседен на Address Map).

32. Вдясно → Diagram → лента с бутони → Validate Design (F6) → "Validation successful. There are no errors or critical warnings in this design." → OK.

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.

33. Централно → в Block design прозореца, натиска се таб-а Sources → Design sources → right-click на design_1.bd → Create HDL Wrapper (създава VHDL описание на новосъздадената система) → Let Vivado manage wrapper and auto-update → OK

34. Добавя се constraints file, в който ще се опише извеждането на сигналите gpio_out_0 и gpio_out_1 на изводи съответно V20 и W20, които излизат на JB конектора под имената JB2_P и JB2_N.



IO_L13P_T2_MRCC_34	P19	VGA_HS
IO_L13N_T2_MRCC_34	N20	VGA_G1
IO_L14P_T2_SRCC_34	P20	VGA_B0
IO_L14N_T2_SRCC_34	T20	JB1_P
IO_L15P_T2_DQS_34	U20	JB1_N
IO_L15N_T2_DQS_34	V20	JB2_P
IO_L16P_T2_34	W20	JB2_N
IO_L16N_T2_34	Y18	JB3_P
IO_L17P_T2_34		

Вляво → таб Sources → десен бутон върху категорията Constraints → Add Sources → Add or create constraints → Next → Create file → File name: abs_gradient_out → OK → Finish.

Отворете категорията Constraints → constrs_1 (1) → двукратно щракване на abs_gradient_out.xdc → въвежда се:

```
#GPIO 0
```

```
set_property PACKAGE_PIN V20 [get_ports gpio_out_0_0]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports gpio_out_0_0]
```


#GPIO 1

```
set_property PACKAGE_PIN W20 [get_ports gpio_out_1_0]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports gpio_out_1_0]
```

ЗАБЕЛЕЖКА: работната маса с платка Zybo Z7-10 трябва да използва куплунг JC и съответно сигналите gpio_out_0_0 (T11), gpio_out_1_0 (T10).

#ZYBO Z7-10, GPIO 0

```
set_property PACKAGE_PIN T11 [get_ports gpio_out_0_0]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports gpio_out_0_0]
```

#ZYBO Z7-10, GPIO 1

```
set_property PACKAGE_PIN T10 [get_ports gpio_out_1_0]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports gpio_out_1_0]
```

Проверете в design_1_wrapper.v дали наистина имената на сигналите са gpio_out_0_0 и gpio_out_0_1.

35. 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 наблюдавате 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.

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

=====

37. От страничния панел на Ubuntu изберете бутон “Show Applications”, след което в полето “Type to search” напишете Vitis Classic и натиснете с ляв бутон на мишката иконката на програмата.

38. 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.

39. File → New → Platform project → Platform project name: 15_custom_ip_verilog_mb_pla → Next → таб "Create new platform from hardware" → Browse → избира се пътя до проекта 15_custom_ip_verilog, създаден от Vivado → design_1_wrapper.xsa → Open → избира се Processor: microblaze_0 → Finish.

40. Вляво → Project explorer → избира се 15_custom_ip_verilog_mb_pla → right-click → Build Project.

41. File → New → Application project → Next → "Select a platform from repository" → Избира се 15_custom_ip_verilog_mb_pla → Next → Application project name: 15_custom_ip_verilog_mb_app → Next → Next → "Empty Application (C)" → Finish.

42. Щраква се двукратно с ляв бутон върху директорията src в проекта 15_custom_ip_verilog_mb_app_system/15_custom_ip_verilog_mb_app → десен бутон върху src директорията → New → Other → C/C++ → Source File → Next → Source file: main_mb.c → Finish.

43. Щракнете двукратно върху lscript.ld от src директорията. Въведете в полето Stack size: 0x1000 → CTRL + s → затворете табът на линкерния редактор.

44. Разберете базовия адрес на персонализираното IP от 15_custom_ip_verilog_mb_pla/hw/design_1_wrapper.xsa → abs_gradient_out_0 → поле Base Address. В настоящия пример, този адрес е 0x43c00000.

45. Въведете в main_mb.c на микропроцесора MicroBlaze следната програма (ка:

```
#include <stdio.h>
#include <xgpio.h>
#include "xparameters.h"
#include "sleep.h"
#include "xil_exception.h"
#include "xintc.h"

typedef struct {
    UINTPTR BaseAddress;    /**< Device base address */
    u32 IsReady;            /**< Device is initialized and ready */
    int InterruptPresent;    /**< Are interrupts supported in h/w */
    int IsDual;             /**< Are 2 channels supported in h/w */
} abs_gradient_out_t;

XGpio output;
abs_gradient_out_t abs_module_0;
XIntc intc_0;

void abs_gradient_interrupt_0(void){
    static int flag = 1;

    if(flag){
```

```

        flag = 0;
        XGpio_DiscreteWrite(&output, 1, 0x01);
    }
    else{
        flag = 1;
        XGpio_DiscreteWrite(&output, 1, 0x00);
    }
}

void abs_gradient_interrupt_1(void){
    static int flag = 1;

    if(flag){
        flag = 0;
        XGpio_DiscreteWrite(&output, 1, 0x02);
    }
    else{
        flag = 1;
        XGpio_DiscreteWrite(&output, 1, 0x00);
    }
}

int main(void){
    abs_module_0.BaseAddress = (volatile uint32_t)0x43C00000;
    abs_module_0.IsReady = 1;
    abs_module_0.IsDual = 0;
    abs_module_0.InterruptPresent = 1;

    XGpio_Initialize(&output, XPAR_AXI_GPIO_0_DEVICE_ID);
    XGpio_SetDataDirection(&output, 1, 0x0);

    XIntc_Initialize(&intc_0, XPAR_INTC_0_DEVICE_ID);
    XIntc_SelfTest(&intc_0);
    XIntc_Connect(&intc_0,
XPAR_MICROBLAZE_0_AXI_INTC_ABS_GRADIENT_OUT_0_INTERRUPT_0_INTR,
(XInterruptHandler)abs_gradient_interrupt_0, &abs_module_0);
    XIntc_Connect(&intc_0,
XPAR_MICROBLAZE_0_AXI_INTC_ABS_GRADIENT_OUT_0_INTERRUPT_1_INTR,
(XInterruptHandler)abs_gradient_interrupt_1, &abs_module_0);
    XIntc_Start(&intc_0, XIN_REAL_MODE);
    XIntc_Enable(&intc_0,
XPAR_MICROBLAZE_0_AXI_INTC_ABS_GRADIENT_OUT_0_INTERRUPT_0_INTR);
    XIntc_Enable(&intc_0,
XPAR_MICROBLAZE_0_AXI_INTC_ABS_GRADIENT_OUT_0_INTERRUPT_1_INTR);

    Xil_ExceptionInit();
    Xil_ExceptionEnable();
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
(Xil_ExceptionHandler)XIntc_InterruptHandler, &intc_0);

    //Clear the AXI interrupt controller's pending flag
    XIntc_Acknowledge(&intc_0,
XPAR_MICROBLAZE_0_AXI_INTC_ABS_GRADIENT_OUT_0_INTERRUPT_0_INTR);
    XIntc_Acknowledge(&intc_0,
XPAR_MICROBLAZE_0_AXI_INTC_ABS_GRADIENT_OUT_0_INTERRUPT_1_INTR);

    while(1){

    }
}

```

```
    return 0;
}
```

46. Вляво, Project explorer -> избира се 15_custom_ip_verilog_mb_app -> right-click -> Build project.

47. Вляво, Project explorer -> избира се 15_custom_ip_verilog_mb_app_system -> right-click -> Build project.

48. File → New → Platform project → Platform project name: 15_custom_ip_verilog_cortex_pla → Next → таб "Create new platform from hardware" → Browse → избира се пътя до проекта 15_custom_ip_verilog, създаден от Vivado → design_1_wrapper.xsa → Open → Finish.

49. Вляво → Project explorer → избира се 15_custom_ip_verilog_cortex_pla → right-click → Build Project.

50. File → New → Application project → Next → "Select a platform from repository" → Избира се 15_custom_ip_verilog_cortex_pla → Next → Application project name: 15_custom_ip_verilog_cortex_app → Next → Next → "Hello world" → Finish.

51. Щраква се двукратно с ляв бутон върху директорията src в проекта 15_custom_ip_verilog_cortex_app_system/15_custom_ip_verilog_cortex_app → отваря се helloworld.c.

52. На мястото на темплейт програмата, копирайте следния код за ARM Cortex A9:

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "sleep.h"

volatile uint32_t *abs_interrupt = ((volatile uint32_t *)0x43C00000)+11;

int main(){
    init_platform();

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

    while(1){
        *abs_interrupt = 0x01;
        *abs_interrupt = 0x00;
        usleep(1000000);
        *abs_interrupt = 0x02;
        *abs_interrupt = 0x00;
        usleep(1000000);
    }
}
```

```

cleanup_platform();
    return 0;
}

```

53. Вляво, Project explorer → избира се 15_custom_ip_verilog_cortex_app → right-click → Build project.

54. Вляво, Project explorer → избира се 15_custom_ip_verilog_cortex_app_system → right-click → Build project.

55. В основния прозорец на Vitis до бутонът Debug има стрелка надолу → натиска се → Debug configurations ... → щраква се двукратно върху Single Application Debug → вдясно ще се появи нова конфигурация на дебъг сесия. Избира се таб Application и се слагат отметки на двата процесора: microblaze_0 и ps7_cortexa9_0. Проверяват се полетата Application, указващи фърмуерния .elf файл за всеки процесор. Полето на MicroBlaze може да бъде празно. Затова с ляв бутон в полето Summary се избира целият ред на microblaze_0 и се натиска бутон Search срещу полето Application. Средата Vitis ще предложи всички .elf файлове, които са достъпни. С ляв бутон се натиска двукратно върху съответния файл на MicroBlaze (14_shared_ram_cortex_app.elf). Сега в полетата Application на Summary трябва да се вижда:

microblaze_0	Debug/15_custom_ip_verilog_mb_app.elf
ps7_cortexa9_0	Debug/15_custom_ip_verilog_cortex_app.elf

Натиска се Apply → Debug

ВНИМАНИЕ: Всяко следващо стартиране на Debug сесия може да стане с бутон надолу до Debug бутона от основния прозорец на Vitis, при условие, че поне веднъж е била стартирана дебъг сесия от Debug configurations... прозореца (в конкретния случай това е станало, когато сме натиснали Apply → Debug).

ВНИМАНИЕ: не трябва да се натиска самият бутон Debug понеже това създава нова дебъг сесия, която по подразбиране зарежда фърмуер само на едно Cortex A9 ядро.

ВНИМАНИЕ: при промяна на сорс кода трябва да се натисне Build на фърмуерния проект (_app) и на системния проект (_app_system), иначе дебъг сесията ще зареди старата версия на .elf файлът.

56. Дебъгването на отделните микропроцесори става като се избере с ляв бутон съответния процесор от таб Debug. Дебъг бутоните и всички дебъг табове се

присвояват автоматично на избрания процесор, т.е. въпреки че процесорите са два, наборът от дебъг инструменти е един.

Ако дизайнът е бил успешен, светодиодите LD0 и LD1 трябва да започнат да премигват на 1 секунда последователно. Забележете, че паузата когато и двата светодиода са изгасени е по-дълга, защото функцията `XGpio_DiscreteWrite(&output, 1, 0x00);` се вика два пъти.

57. Напишете програма, която тества сигналите `gpio_out_0` и `gpio_out_1`. Свържете осцилоскоп на сигналите JB2_P и JB2_N от куплунга JB.

58. Напишете програма, която тества модулта за събиране на две числа по абсолютна стойност и делене с трето число. Формулата е:

$$\text{slv_reg10} = (\text{slv_reg7} + \text{slv_reg8}) / \text{slv_reg9};$$

59. Напишете програма, която тества модулта за изчисляване на градиент на 6 числа по формулата:

$$\text{slv_reg6} = (\text{slv_reg3} + (\text{slv_reg4} * 2) + \text{slv_reg5}) - (\text{slv_reg0} + (\text{slv_reg1} * 2) + \text{slv_reg2});$$

60. Напишете програма, която описва персонализираният модул чрез структура на C (виж документ `07_struct_map.pdf` в директорията на настоящото лабораторно упражнение). Нека програмата тества всичките 4 подмодула – прекъсване, GPIO, градиент и изчисляване на модул.

*

*

*

[1] https://github.com/k0nze/zedboard_pl_to_ps_interrupt_example

[2] “Zynq-7000 SoC: Embedded Design Tutorial – A Hand-On Guide to Effective Embedded System Design”, UG1165 (v2020.1), Xilinx, June 10, 2020.

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