Проектиране на вградени автомобилни електронни системи
Лабораторно упражнение №13

Работа с Xilinx Vivado и Vitis. Споделяне на модул посредством хардуерен мютекс (mutex) в многопроцесорна система.

==========

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

2. Стартирайте терминал с CTRL + ALT + T и изпълнете командите:

```
source ~/programs/xilinx/Vivado/2020.2/settings64.sh
vivado
```

3. Create Project → Next → Project name: 13_mutex → 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.

4. Вляво → Flow navigator → Create block design → OK.

5. Вдясно → Diagram → right-click → Add IP → Search → ZYNQ7 Processing System → double click.

6. 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.

7. Вдясно → Diagram → right-click → Add IP → Search → Processor System Reset → double click.

8. Вдясно → Diagram → зелена лента → Designer Assitance available → Run Block Automation → Слага се отметка на "All Automation".

9. Вдясно → Diagram → right-click → Add IP → Search → AXI Uartlite → double click.

10. Щракнете два пъти върху блока AXI Uartlite → Baudrate → 115200 → OK.

11. Вдясно → Diagram → right-click → Add IP → Search → AXI Interconnect → double click.

12. Щракнете два пъти върху блока AXI Interconnect → Number of Slave Interfaces = 2 → Number of Master Interfaces = 3 → OK.

13. Вдясно → Diagram → right-click → Add IP → Search → MicroBlaze → double click.

14. Вдясно →Diagram → зелена лента → Designer Assitance available -> Run Block Automation → Слага се отметка на "microblaze_0". В полето Options се избира Local memory: 64 kB и Cache configuration: None. Натиска се OK.

15. Щракнете два пъти върху блока MicroBlaze → Predefined configurations → Select Configuration: Microcontoller preset → OK.

16. Натиска се и се задържа ляв бутон върху M_AXI_DP порта и се свързва с порта S00_AXI на AXI Interconnect блокът, след това се пуска левия бутон. Аналогично се свързва портът M00_AXI с порта S_AXI на AXI Uartlite модула. Аналогично се свързва M_AXI_GP0 порта на ZYNQ7 блокът с порта S01_AXI на AXI Interconnect блокът.

17. Вдясно → Diagram → right-click → Add IP → Search → Mutex → double click [1].

18. Натиска се и се задържа ляв бутон върху M01_AXI порта на AXI Interconnect блокът и се свързва с порта S0_AXI на mutex_0 блока, след това се пуска левия бутон. Аналогично се свързва портът M02_AXI с порта S1_AXI на mutex_0 .

19. Щраква се двукратно върху ZYNQ Processing System → “Page navigator” → MIO Configuration → махат се отметките в раздел I/O Peripherals на ENET0, USB0, SD0 и UART1. Натиска се OK.

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

21. В основния прозорец на Vivado, до таб Diagram, се избира Address Editor → натиска се бутон Assign All.

22. Подрежда се блоковата схема с бутон Regenerate Layout.

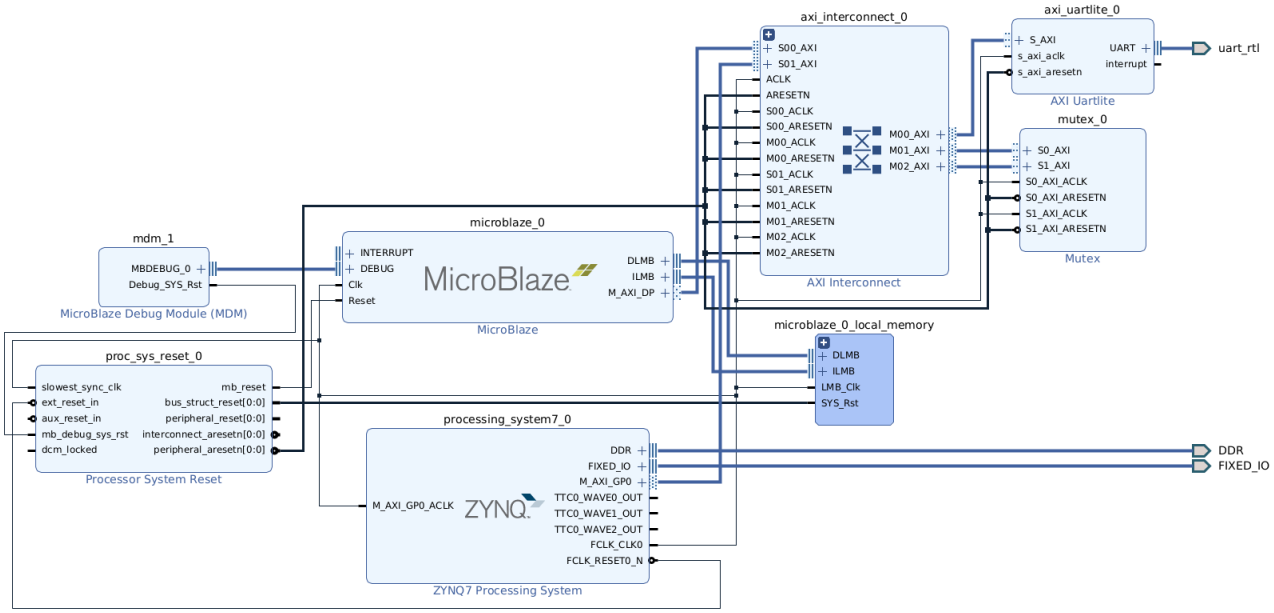
23. Вдясно→Diagram→лента с бутони → Validate Design (F6) → "Validation successful. There are no errors or critical warnings in this design." → OK

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.

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

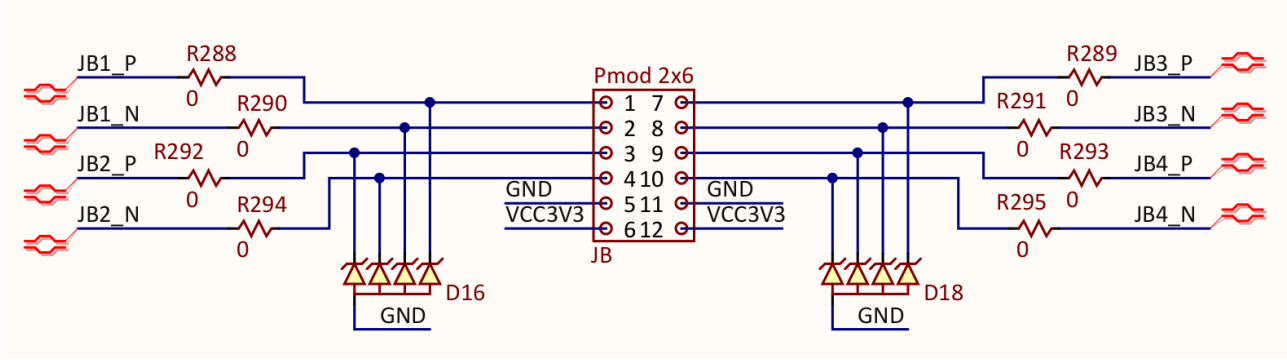
Блоковата схема на системата е показана по-долу.

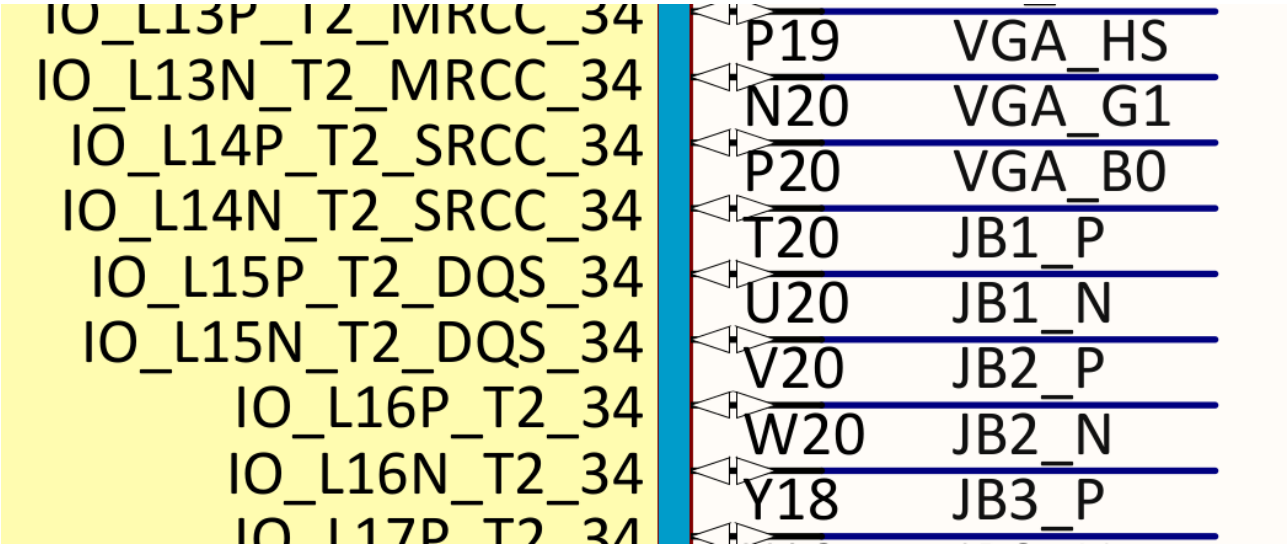


25. File → Add Sources → Add or create constraints → Next → Add files → укажете пътя до 00_ZYBO_Master.xdc (има го в директорията на настоящото лабораторно) → OK → сложете отметка на “Copy constraints files into project” → Finish.

26. Горе, вляво → таб Sources → Constraints → constrs_1 → щракнете двукратно върху 00_ZYBO_Master.xdc, за да се отвори constraints файла. В него се описват връзките между вътрешните сигнали на FPGA и изводите на корпуса на FPGA.

Гледайки принципната схема на ZYBO, трябва да се проследят връзките на порт JB с изводите на FPGA.





В constraints файла се записват следните редове:

#UART TX
set_property PACKAGE_PIN T20 [get_ports uart_rtl_txd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_txd]

#UART RX
set_property PACKAGE_PIN U20 [get_ports uart_rtl_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_rxd]

след което се натиска Ctrl + s от клавиатурата, за да се запазят промените.

ВНИМАНИЕ! Имената на сигналите в дадения constraints файл са примерни (uart_rtl_rxd, uart_rtl_txd). Реалните имена могат да се видят от Design Sources → design_1_wrapper → design_1_i → двукратно щракване върху design_1.v

Пример – в настоящото упражнение Vivado е генерирал в wrapper-а следните сигнали:

```
module design_1
  (DDR_addr,
   DDR_ba,
   DDR_cas_n,
   DDR_ck_n,
   DDR_ck_p,
   DDR_cke,
   DDR_cs_n,
   DDR_dm,
   DDR_dq,
   DDR_dqs_n,
   DDR_dqs_p,
   DDR_odt,
   DDR_ras_n,
   DDR_reset_n,
   DDR_we_n,
   FIXED_IO_ddr_vrn,
   FIXED_IO_ddr_vrp,
   FIXED_IO_mio,
   FIXED_IO_ps_clk,
   FIXED_IO_ps_porb,
   FIXED_IO_ps_srstb,
   uart_rtl_rxd,
   uart_rtl_txd);
```

затова четирите Tcl команди дадени като пример съдържат get_ports uart_rtl_rxd и get_ports uart_rtl_txd.

27. Свържете USB-UART конвертор към горния ред изводи на куплунга JB на Zybo платката.

28. 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.

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

==========

==========

30. Tools → Launch Vitis IDE

31. 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.

32. File → New → Platform project → Platform project name: 13_mutex_mb_pla → Next → таб "Create new platform from hardware" → Browse → избира се пътя до проекта 13_mutex, създаден от Vivado → design_1_wrapper.xsa → Open → Поле Software specification → Operating system: standalone (това означава bare-metal firmware) и Processor: microblaze_0 → Finish.

33. Вляво → Project explorer → избира се 13_mutex_mb_pla → right-click → Build Project.

34. File → New → Application project → Next → "Select a platform from repository" → Избира се 13_mutex_mb_pla → Next → Application project name: 13_mutex_mb_app → Next → Next → “Hello World” → Finish.

35. Вляво в таб Explorer → отваря се 13_mutex_mb_app_system / 13_mutex_mb_app / src → щраква се двукратно върху helloworld.c.

36. В текстовия редактор на Vitis и във файлът helloworld.c на MicroBlaze фърмуера се въвежда следната програма:

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

int main(){
    init_platform();

    while(1){
      print("MicroBlaze\n\r");
    }

    cleanup_platform();

    return 0;
}
```

37. Вляво → Project explorer → избира се 13_mutex_mb_app → right-click → Build Project.

38. Вляво → Project explorer → избира се 13_mutex_mb_app_system → right-click → Build Project.

39. File → New → Platform project → Platform project name: 13_mutex_cortex_pla → Next → таб "Create new platform from hardware" → Browse → избира се пътя до проекта 13_mutex, създаден от Vivado → design_1_wrapper.xsa → Open → Поле Software specification → Operating system: standalone и Processor: ps7_cortexa9_0 → Finish.

40. Вляво → Project explorer → избира се 13_mutex_cortex_pla → right-click → Build Project.

41. File → New → Application project → Next → "Select a platform from repository" → Избира се 13_mutex_cortex_pla → Next → Application project name: 13_mutex_cortex_app → Next → Next → “Hello World” → Finish.

42. Щраква се двукратно с ляв бутон върху директорията src в проекта 13_mutex_cortex_app_system/ 13_mutex_cortex_app → src → helloworld.c

43. Пренасочва се printf библиотеката към Uartlite. За целта се отваря платформеният проект 13_mutex_cortex_pla → двукратно щракване върху platform.spr → ще се отвори нов таб в Eclipse с настройки на платформения проект (исторически наричан още Board Support Package – BSP). В категорията Standalone on ps7_cortexa9_0 се щраква подкатегорията Board Support Package → Modify BSP Settings … → категория Overview → standalone → поле Configuration for OS: → Value на категориите stdin и stdout се променя на axi_uartlite_0 → OK → затваря се табът.

44. В текстовия редактор на Vitis и във файлът main.c на ARM Cortex A9 фърмуера се въвежда следната програма:

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

int main(){
    init_platform();

    while(1){
      print("ARM Cortex A9\n\r");
    }

    cleanup_platform();

    return 0;
}
```

45. Вляво, Project explorer → избира се 13_mutex_cortex_app → right-click → Build project.

46. Вляво, Project explorer → избира се 13_mutex_cortex_app_system → right-click → Build project.

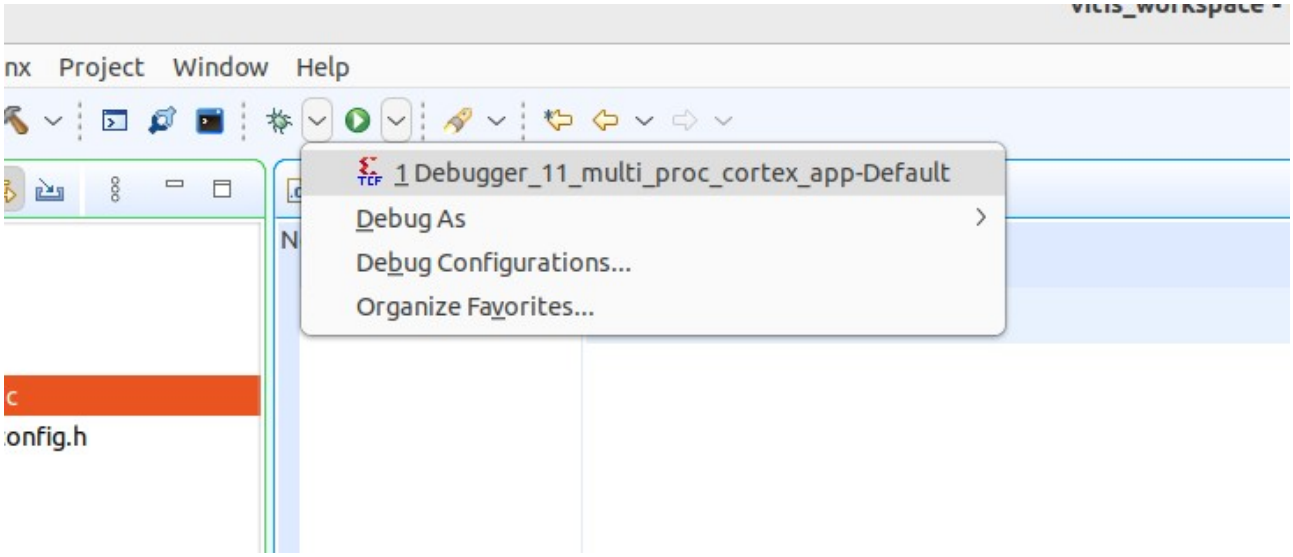
47. В основния прозорец на Vitis до бутонът Debug има стрелка надолу → натиска се → Debug configurations … → щраква се двукратно върху Single Application Debug → вдясно ще се появи нова конфигурация на дебъг сесия. Избира се таб Application и се слагат отметки на двата процесора: microblaze_0

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 (11_multi_proc_mb_app.elf). Сега в полетата Application на Summary трябва да се вижда:

microblaze_0 Debug/13_mutex_mb_app.elf
ps7_cortexa9_0 Debug/13_mutex_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.

48. 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.

49. Отваря се терминал в Ubuntu с CTRL + ALT + T → Пише се ls /dev/tty и се натиска tab → "Display all 100 possibilities? (y or n)" въвежда се 'y' → **търси се системния файл, отговарящ на виртуалния RS232 порт** за дебъг съобщения.

Сега се търси номера на виртуалния порт на USB-UART конвертора. След като се намери, в същия терминал се стартира RS232 терминал чрез командата:

```
cutecom
```

50. В cutecom → Device: избира се съответния порт за дебъг съобщения /dev/ttyUSBx → Settings → 115200-8-N-1, no flow control -> Open

51. Във Vitis: натиска се бутон Resume (F8) за Cortex A9 (ядро 0). След това в Cutecom трябва да се изпише:

```
ARM Cortex A9
ARM Cortex A9
ARM Cortex A9
ARM Cortex A9
ARM Cortex A9
ARM Cortex A9
```

52. Във Vitis: натиска се бутон Resume (F8) за MicroBlaze. След това в Cutecom трябва да се изпише:

```
MicroBlaze
MicroBlaze
MicroBlaze
MicroBlaze
MicroBlaze
MicroBlaze
MicroBlaze
```

53. Натиснете бутон Suspend на ARM Cortex A9 ядрото. Cortex-ът би трябва да е зациклил във функцията void Xil_DataAbortHandler(void *CallBackRef) в xil_exception.c файла. Това е защото и двата процесора се опитват да достъпят един и същи ресурс едновременно.

54. За да спрете debug сесията във Vitis, натиснете Disconnect.

55. Напишете програма, която защитава print съобщенията (и в MicroBlaze, и в ARM Cortex A9) с мютекс като първо заключите мютекса, после принтирате съобщението, после отключвате мютекса. Използвайте следните функции:

XMutex_LookupConfig()
XMutex_CfgInitialize()
XMutex_Lock()
XMutex_Unlock()

ВНИМАНИЕ: хардуерният мютекс заключва и отключва ресурса като взима под внимание ID-то на процесорът, който се опитва да отключи/заключи ресурса [2]. Проверете в xparameters.h и на двата процесора какво ID е дал Vivado като число. Ако ID-та са еднакви, мютексът няма да работи.

Ако успешно сте написали програмата, в терминала трябва да се виждат следните съобщения:

```
ARM Cortex A9
MicroBlaze
ARM Cortex A9
MicroBlaze
ARM Cortex A9
MicroBlaze
ARM Cortex A9
MicroBlaze
```

56. Пренапишете програмата от миналата точка, но използвайте неблокиращата функция XMutex_Trylock( ).

* * *

[1] Adam Tylor, “MicroZed Chronicles: Inter Processor Communication (Part 3)”, online, https://www.hackster.io/news/microzed-chronicles-inter-processor-communication-part-3-f007cddc3af5, 2023.
[2] LogiCORE IP Product Guide, “Mutex v2.1”, PG117 (v2.1), Vivado Design Suite, November 21, 2019.

доц. д-р инж. Любомир Богданов, 2023 г.