Проектиране на вградени автомобилни електронни системи
Лабораторно упражнение №14

Работа с Xilinx Vivado и Vitis. Споделяне на външна DDR RAM памет 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. 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: 14_shared_ram → 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. 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 → MicroBlaze → double click.

10. 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.

11. Щракнете два пъти върху блока MicroBlaze → Predefined configurations → Select Configuration: Microcontoller preset → OK [1].

12. Щракнете два пъти върху блока ZYNQ7 → “Page navigator” → PS-PL Configuration → GP Slave AXI Interface → сложете отметка на S AXI GP0 interface [2]. “Page navigator” → MIO Configuration → махат се отметките в раздел I/O Peripherals на ENET0, USB0, SD0. На UART1 се проверяват връзките MIO48 ↔ tx, MIO49 ↔ rx. Натиска се OK.

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

14. Щракнете два пъти върху блока AXI Interconnect → Number of Slave Interfaces = 1 → Number of Master Interfaces = 1 → OK.

15. 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_GP0 порта на ZYNQ7 блокът.

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

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

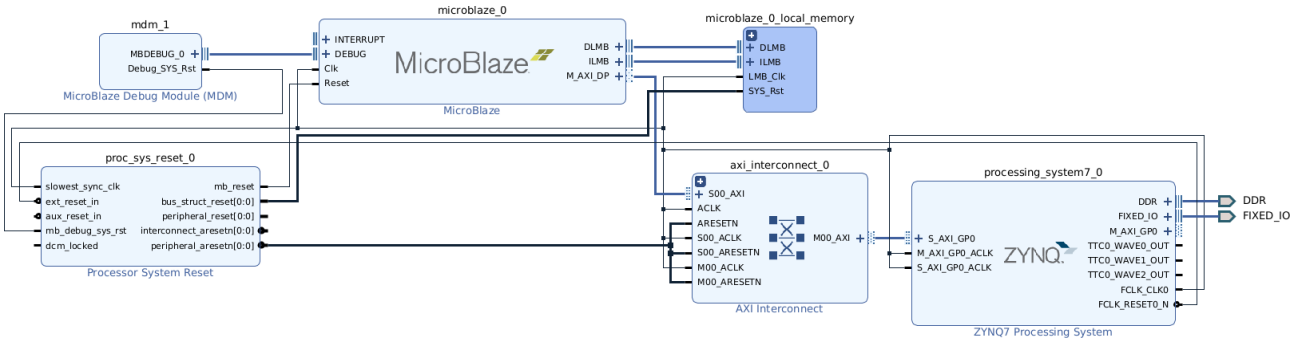
18. Подрежда се блоковата схема с бутон Regenerate Layout.

19. 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.

20. 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.



21. 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.

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

==========

==========

23. Tools → Launch Vitis IDE

24. 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.

25. File → New → Platform project → Platform project name: 14_shared_ram_mb_pla → Next → таб "Create new platform from hardware" → Browse → избира се пътя до проекта 14_shared_ram, създаден от Vivado → design_1_wrapper.xsa → Open → Поле Software specification → Operating system: standalone (това означава bare-metal firmware) и Processor: microblaze_0 → Finish.

26. Вляво → Project explorer → избира се 14_shared_ram_mb_pla → right-click →

Build Project.

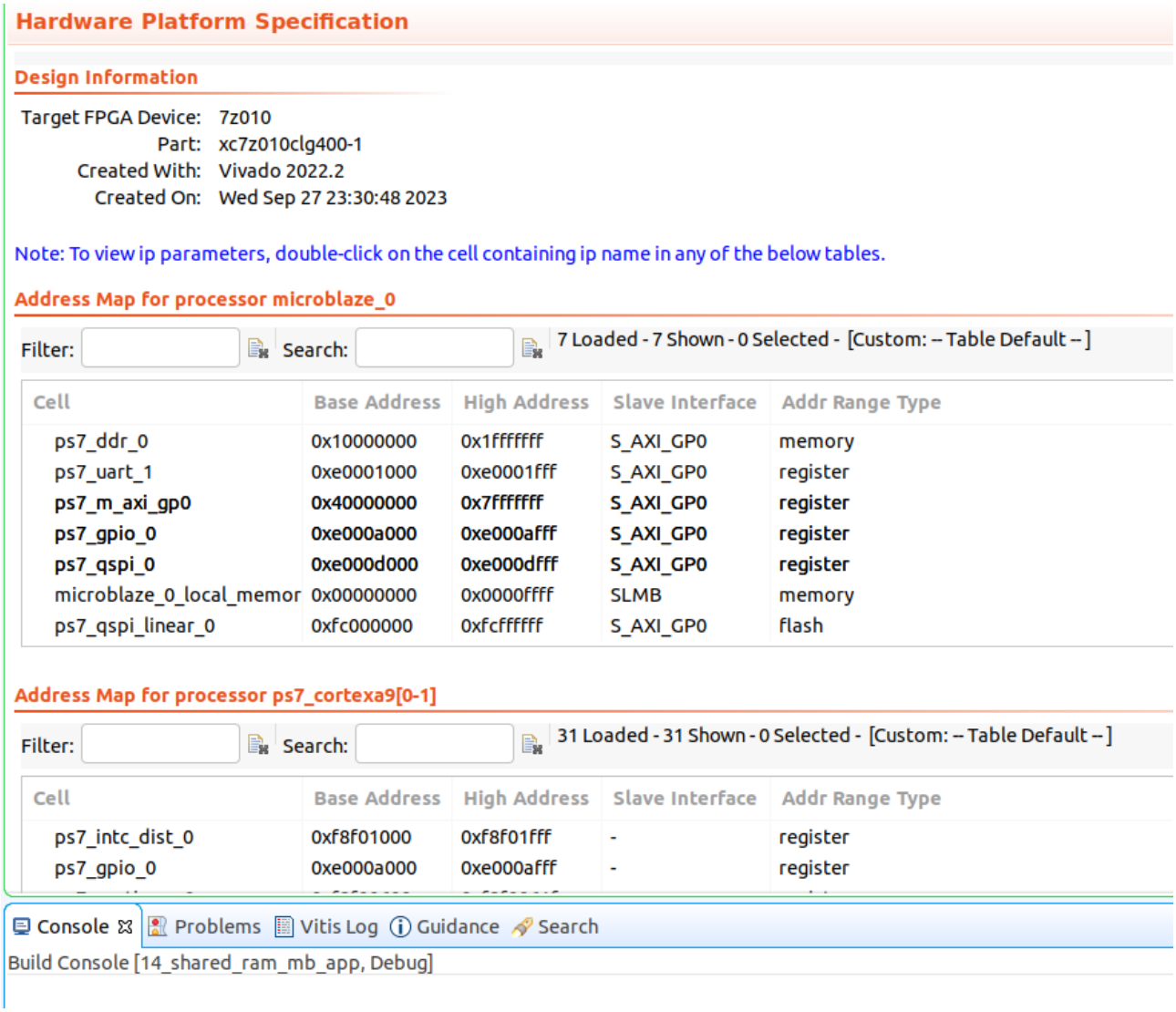
27. File → New → Application project → Next → "Select a platform from repository" → Избира се 14_shared_ram_mb_pla → Next → Application project name: 14_shared_ram_mb_app → Next → Next → “Empty application (C)” → Finish.

28. File → New → Application project → Next → "Select a platform from repository" → Избира се 14_shared_ram_mb_pla → Next → Application project name: 14_shared_ram_mb_app → Next → Next → “Hello World” → Finish.

29. Щраква се двукратно с ляв бутон върху директорията src в проекта 14_shared_ram_mb_app_system/ 14_shared_ram_mb_app → src → helloworld.c

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

volatile uint32_t *ddr_start_register = (volatile uint32_t *)0x10000000;

int main(){
    init_platform();

    while(1){
      strcpy((char *)ddr_start_register, "Hello, this is MicroBlaze!\n\r");
      usleep(1000000);
      strcpy((char *)ddr_start_register, "The messages are in the shared DDR
SDRAM.\n\r");
      usleep(1000000);
      strcpy((char *)ddr_start_register, "The example is not safe, we have no
mutex.\n\r");
      usleep(1000000);
    }

    cleanup_platform();

    return 0;
}
```

30. Вляво→Project explorer → избира се 14_shared_ram_mb_pla → директория hw → щраква се двукратно върху design_1_wrapper.xsa. Разгледайте картата на паметта и се уверете, че началният адрес на DDR е 0x10000000. Ако това не е така, коригирайте адреса на указателя ddr_start_register [3], [4], [5].



31. Вляво → Project explorer → избира се 14_shared_ram_mb_app → right-click → Build Project.

32. Вляво → Project explorer → избира се 14_shared_ram_mb_app_system → right-click → Build Project.

33. File → New → Platform project → Platform project name: 14_shared_ram_cortex_pla → Next → таб "Create new platform from hardware" → Browse → избира се пътя до проекта 14_shared_ram, създаден от Vivado → design_1_wrapper.xsa → Open → Поле Software specification → Operating system: standalone и Processor: ps7_cortexa9_0 → Finish.

34. Вляво → Project explorer → избира се 14_shared_ram_cortex_pla → right-click → Build Project.

35. File → New → Application project → Next → "Select a platform from repository" → Избира се 14_shared_ram_cortex_pla → Next → Application project name: 14_shared_ram_cortex_app → Next → Next → “Hello World” → Finish.

36. Щраква се двукратно с ляв бутон върху директорията src в проекта 14_shared_ram_cortex_app_system/ 14_shared_ram_cortex_app → src → helloworld.c

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

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

volatile uint32_t *ddr_start_register = (volatile uint32_t *)0x10000000;

int main(){
    init_platform();

    Xil_DCacheDisable();

    usleep(100000);

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

    while(1){
      xil_printf("The message: %s\n\r", ddr_start_register);
      usleep(200000);
    }

    cleanup_platform();

    return 0;
}
```

38. Вляво, Project explorer → избира се 14_shared_ram_cortex_app → right-click → Build project.

39. Вляво, Project explorer → избира се 14_shared_ram_cortex_app_system → right-click → Build project.

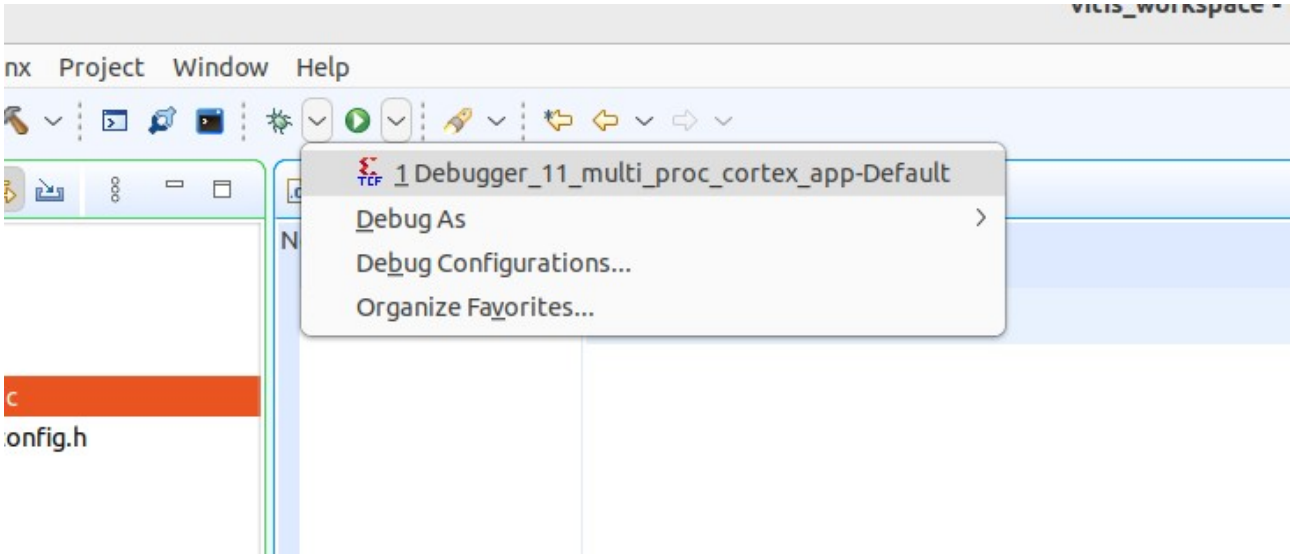
40. В основния прозорец на Vitis до бутонът Debug има стрелка надолу → натиска се → Debug configurations … → щраква се двукратно върху Single Application Debug → вдясно ще се появи нова конфигурация на дебъг сесия. Избира се таб Application и се слагат отметки на двата процесора: microblaze_0 и ps7_cortexa9_0. 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/14_shared_ram_mb_app.elf
ps7_cortexa9_0 Debug/14_shared_ram_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.

41. 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.

42. 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
```

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

44. Във Vitis: натиска се бутон Resume (F8) за MicroBlaze.

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

```
Starting ...
The message: Hello, this is MicroBlaze!
The message: Hello, this is MicroBlaze!
The message: Hello, this is MicroBlaze!
The message: Hello, this is MicroBlaze!
The message: Hello, this is MicroBlaze!
The message: The messages are in the shared DDR SDRAM.
The message: The messages are in the shared DDR SDRAM.
The message: The messages are in the shared DDR SDRAM.
The message: The messages are in the shared DDR SDRAM.
The message: The messages are in the shared DDR SDRAM.
The message: The example is not safe, we have no mutex.
The message: The example is not safe, we have no mutex.
The message: The example is not safe, we have no mutex.
The message: The example is not safe, we have no mutex.
The message: The example is not safe, we have no mutex.
```

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

47. Закоментирайте редът Xil_DcacheDisable(); от фърмуера на ARM Cortex A9. Пуснете отново дебъг сесия. Наблюдавайте в терминала Cutecom некохерентността на кеша.

48. Използвайки функциите в xil_cache.h напишете програма, която използва кеш на ARM Cortex A9 и работи правилно (като в точка 45).

49. Напишете програма, която принуждава ARM Cortex A9 ядрото да чака, докато не получи стартов сигнал от MicroBlaze микропроцесора.

* * *

[1] Caglayan Dokme, “A Shared BRAM Example with Microblaze and Zynq SOC”, online, https://medium.com/@caglayandokme/a-shared-bram-example-with-microblaze-and-zynq-soc-949495b5f540 , 2021.

[2] Adam Taylor, “MicroZed Chronicles: Combining MicroBlaze & the Zynq MPSoC”, online, https://www.hackster.io/news/microzed-chronicles-combining-microblaze-the-zynq-mpsoc-94501295dd3 , 2018.

[3] Sandeep Gundlupet Raju, “Accessing BRAM in Linux”, online, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842412/Accessing+BRAM+In+Linux , 2020.

[4] "Zynq UltraScale+ MPSoC: Embedded Design Tutorial: A Hands-On Guide to Effective Embedded System Design", User’s Guide, Xilinx Inc, UG1209 (v2018.3), 2018.

[5] John McDougall, “Simple AMP: Zynq SoC Cortex-A9 Bare-Metal System with MicroBlaze Processor”, Xilinx Application Note, XAPP1093 (v1.0.1), 2014.

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