ELE8307 - Laboratoire 5
Conception d'un système multiprocesseurs

Introduction

Ajouter du matériel dédié à un processeur permet une augmentation conséquente des performances, nous l'avons observé dans les laboratoires précédents. Une autre approche consiste tout simplement à augmenter le nombre de processeurs du système. En théorie, la limite maximale d'accélération correspond à l'augmentation du nombre de processeurs qui se partageront la ou les tâches. Lorsqu'une seule tâche est accélérée par l'ajout de processeurs au système, la grande problématique de conception est de décider comment diviser ou d'organiser cette dernière afin qu'elle soit exécutée en parallèle, par plusieurs processeurs. Dépendamment de l'application ciblée, cette tâche peut être plus ou moins difficile, voir impraticable.

Objectifs

Dans ce laboratoire, nous apprendrons à réaliser une plate-forme multiprocesseur et à modifier une application pour accélérer son exécution sur cette dernière.

- ✓ Création d'une plateforme multiprocesseur avec Qsys.
- ✓ Introduction aux considérations logicielles liées à l'utilisation de plusieurs processeurs.
- ✓ Utilisation d'une plateforme multiprocesseur pour accélérer l'exécution d'une application.
- ✓ Introduction to the HPS system and SoC EDS environment

Documentation

La documentation de référence est disponible sur Moodle. Vous trouverez notamment les deux fichiers :

1. Guide pour mutex
2. Altera HPS multiprocessor tutorial
3. Guide for pthread
4. Guide for memory map
5. Guide for SoCkit

Introduction

Bien qu'il soit possible de produire des systèmes multiprocesseurs où chaque processeur est indépendant des autres, sans aucun partage de ressource, il est souvent plus intéressant de produire des systèmes avec des ressources partagées, comme une mémoire, permettant une communication entre les processeurs. Sous Qsys, l'ajout d'un processeur implique l'ajout d'un

port maître de données et d'instructions (section « connection » du système), et le partage est spécifié en connectant par exemple les ports de données (et/ou d'instructions) de plusieurs processeurs à une même ressource.

Partage de mémoire

Une mémoire peut être utilisée aussi bien pour stocker des données lors de l'exécution d'un programme que pour contenir les programmes à exécuter. Si une mémoire partagée contient les instructions programmes, chaque processeur doit utiliser un espace mémoire distinct, qui contiendra notamment les sections .text, .rodata, .rwdata, .heap, et .stack. Ces espaces ne peuvent pas être partagés. On peut spécifier le début de l'espace mémoire du programme d'un processeur en ajustant les décalages des adresses mémoire des champs « Reset Vector » et « Exception Vector » d'un processeur Nios II.

Lorsqu'une mémoire est utilisée pour contenir des données partagées entre différents processeur, il suffit d'y relier les bus de données des processeurs concernés et le logiciel s'occupe des interconnexions.

Utilisation d'un Mutex

Afin d'assurer la protection de certaines ressources partagées, le processeur Nios II peut utiliser le composant « Mutex » disponible dans Qsys. Un mutex permet aux processeurs d'obtenir l'accès exclusif à une ressource partagée. Lorsque plusieurs processeurs tentent d'obtenir l'exclusivité sur une ressource via le mutex (on parle de verrou sur une ressource), ce dernier donne l'accès à un seul propriétaire. Lorsque le processeur a terminé d'utiliser la ressource, il relâche le mutex et le verrou est alors donné à un processeur en attente.

Le mutex ne protège pas physiquement les ressources partagées, mais est uniquement une sécurité logicielle que l'on associe à chaque ressource partagée. Le programmeur doit donc s'assurer de verrouiller le mutex avant d'accéder à la ressource à laquelle il est associé, à chaque accès à cette ressource dans le code. La librairie logicielle du mutex (altera_avalon_mutex.h) met à disposition les fonctions pertinentes à son utilisation :

Fonction	Description
altera_avalon_mutex_open()	Initialise le mutex, requis avant d'utiliser les autres fonctions.
altera_avalon_mutex_lock()	Attend de verrouiller le mutex (fonction bloquante !)
altera_avalon_mutex_trylock()	Essaie d'obtenir le verrou sur le mutex, retourne immédiatement, même si le verrou n'est pas obtenu.
altera_avalon_mutex_unlock()	Libère le mutex.
altera_avalon_mutex_is_mine()	Détermine si le processeur possède le verrou sur le mutex.
altera_avalon_mutex_first_lock()	Teste si le mutex a déjà été verrouillé puis libéré depuis l'initialisation du système. Retourne 1 s'il n'a pas été relâché depuis la réinitialisation, 0 sinon.

Il n'est pas toujours requis d'utiliser un mutex lorsqu'une ressource est partagée dans un système multiprocesseur. Il est possible de se réserver un espace mémoire pour conserver des drapeaux de
communication pour les processeurs. Toutefois, lorsque l'exclusivité est requise, le mutex facilite grandement les choses. Utilisez la documentation indiquée en début d'énoncé pour bien comprendre son fonctionnement.
Le mutex sera utilisé dans ce laboratoire, et il pourrait fort bien être utile lors du projet final…

System on Chip :

The SoCkit is a more advanced version of the DE-2 kit which you have been using in the previous labs. The main difference is that the SoCkit has a Hard core processor (HPS) ARM A9 chip along with the FPGA. Since the HPS works at a much higher frequency (1 Ghz) as compared to 300 MHz of the FPGA, it is very advantageous to include an HPS in your design. Another main difference is that the sockit contains a Cyclone V FPGA which can work at 400 Mhz. The FPGA and HPS are connected through 2 different buses namely – i) AXI Bus ii) Lightweight AXI bus. As the name suggests the axi bus is used when large amount of data transfer is required, and the lightweight bus is used when small amount of data transfer is required. The overall architecture can be found in the below link (hardware spec. C):

https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=816

Travail à réaliser

In this lab, you need to accomplish the following tasks:

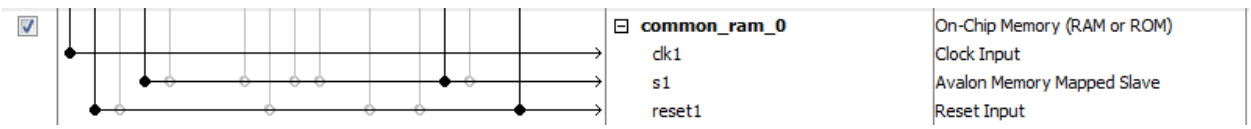
- ✓ Write 10 values [0 - 9] in the common onchip memory using NIOS.
- ✓ Using HPS (Hard core processor- ARM A9), multiply each of the values by 3 and store the 10 new values in another portion of the same common onchip memory.
- ✓ Read the values from the new location using NIOS and display it on the NIOS Console.

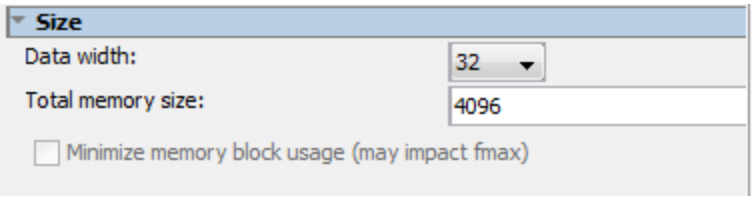
Procedure

1) Open the given project in Quartus 15.1

2) Open Qsys -> soc_system.qsys

3) Add an onchip RAM having a single port (32 bits, 4096 Bytes). Connect the hps_afi_axi_bus and NIOS_1 datamaster bus to the onchip ram.



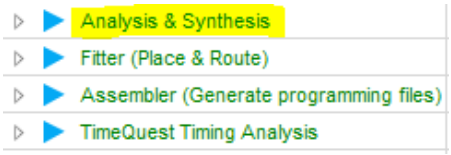


4) Remove the conflicts in address, by manually assigning the address. NOTE: The address space is same at common_ram for the hps and nios 1.

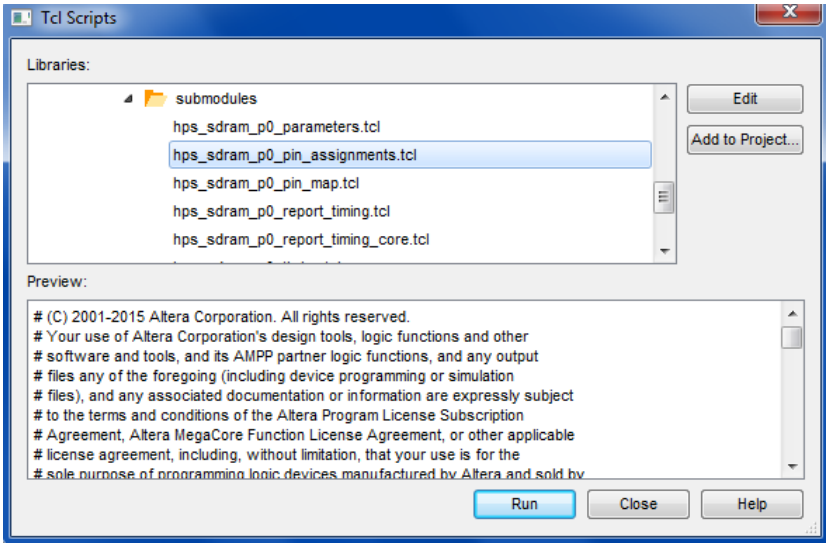
	hps_0.h2f_axi_master	nios_1.data_master
address_span_extender_0.windowed_...		0x1000_0000 - 0x1fff_ffff
common_ram_0.s1	0x0002_0000 - 0x0002_0fff	0x0002_0000 - 0x0002_0fff
dipsw_pio.s1		
hps_0.f2h_axi_slave		
led_gpio_0.s1		0x0003_0000 - 0x0003_000f
mbox_arm2nios_0.avmm_msg_sender		
mbox_arm2nios_0.avmm_msg_receiver		
mbox_arm2nios_1.avmm_msg_sender		
mbox_arm2nios_1.avmm_msg_receiver		0x0007_8000 - 0x0007_800f
mbox_nios2arm_0.avmm_msg_sender		
mbox_nios2arm_0.avmm_msg_receiver		
mbox_nios2arm_1.avmm_msg_sender		0x0007_0000 - 0x0007_000f
mbox_nios2arm_1.avmm_msg_receiver		
nios_0.debug_mem_slave		
nios_1.debug_mem_slave		0x000a_0000 - 0x000a_07ff
nios_buttons_0.s1		
nios_buttons_1.s1		0x0005_0000 - 0x0005_000f
nios_ram_0.s1		
nios_ram_0.s2	0x0000_0000 - 0x0000_ffff	
nios_ram_1.s1		0x0000_0000 - 0x0000_ffff
nios_ram_1.s2	0x0001_0000 - 0x0001_ffff	
sysid_qsys.control_slave		
hps_0.f2h_axi_slave via address_span...		0x2000_0000 - 0x1fff_ffff

5) Generate the qsys.

6) Just run “Analysis and Synthesis” and not entire compilation.



7) Now go to Tools -> TCL script



Run the above hps_sdram_p0_pin_assignment.tcl script to generate the timing constraints for the hps pins.

8) Now, run the entire compilation.

9) For generating the device trees follow the below guide

https://rocketboards.org/foswiki/Documentation/GSRDDeviceTreeGenerator141ArrowSoCKitEdition

```
Siva@SIVA-LT /cygdrive/c/All_Siva/Polytechnique/TA/Wanted_lab/sockit_ghrd_lab2-4
/sockit_ghrd_lab4
$ cd C:\\All_Siva\\Polytechnique\\TA\\Wanted_lab\\sockit_ghrd_lab2-4\\sockit_gh
rd_lab4_
```

*Double-backslash

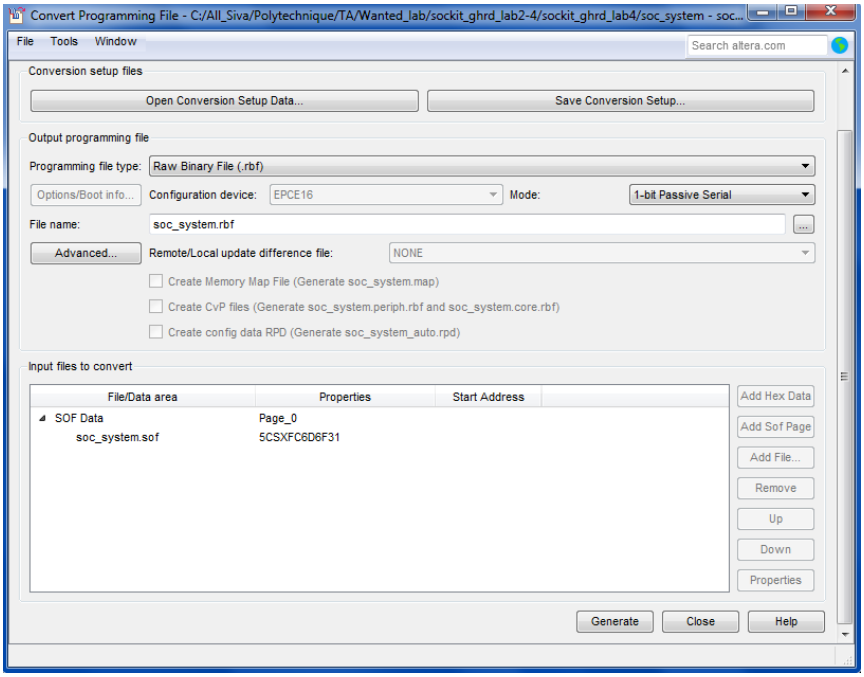
```
Siva@SIVA-LT /cygdrive/c/All_Siva/Polytechnique/TA/Wanted_lab/sockit_ghrd_lab2-4
/sockit_ghrd_lab4
$ sopc2dts --input soc_system.sopcinfo --output socfpga.dts --type dts --board
soc_system_board_info.xml --board hps_common_board_info.xml --bridge-removal al
l --clocks
```

*a blank space before –

soc_system.dts and soc_system.dtb files are generated

10) Create .rbf file so that the hps can program the FPGA. In main quartus window -> files -> Convert programming files

10.1) Select programming file type as .rbf . Rename the file type as soc_system.rbf. The rbf file programs the FPGA on boot which is way more convenient than programming the HPS and FPGA separately (using .sof file). The programming of the FPGA can be confirmed by the FPGA_CONF_D led, present near the HSMC connector (on the right).



10.2) Click “add file” and add the soc_system.sof file

10.3) Click generate

11) An image file is provided in the ELE_8307/2018/labo5. Please burn that image into your sdcard using any of the free software tools such as Win32.

https://sourceforge.net/projects/win32diskimager/

To format the sd card use any of the free software tools such as:

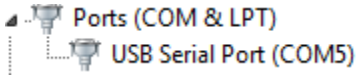
https://www.sdcard.org/downloads/formatter_4/

[If you are unable to burn the image, contact me immediately and I can do it for you]

Take a moment to observe how the sd-card is partioned [observable in ubuntu/mac, for windows need to install appropriate drivers depending on the version of windows which can be found online]. For reference:

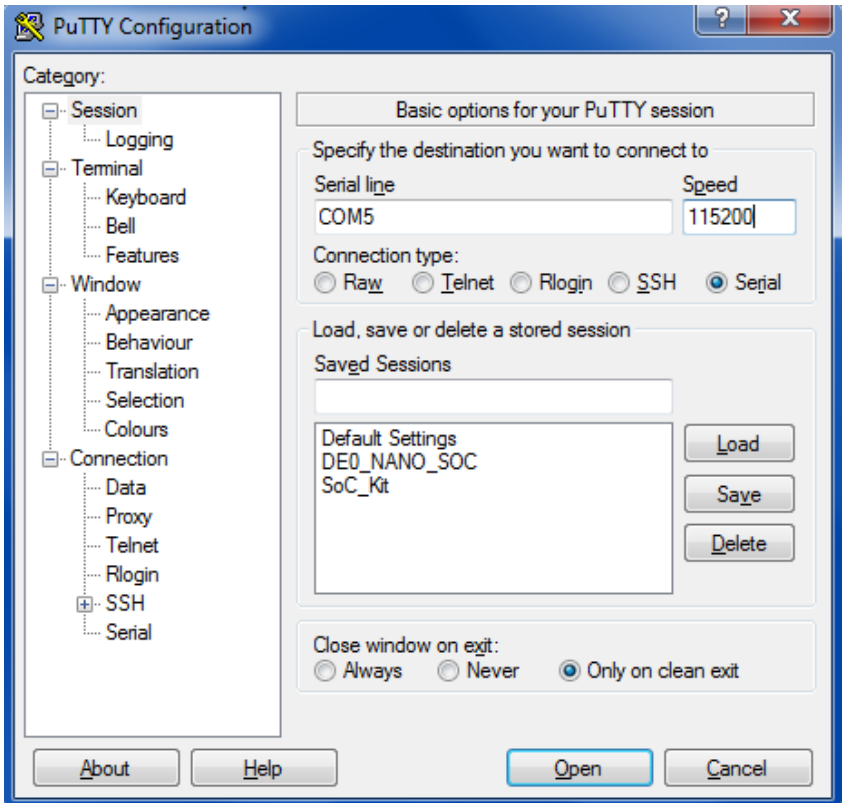
https://rocketboards.org/foswiki/Documentation/GSRD141SDCardArrowSoCKitEdition

12) Insert the sdcard into the sdcard slot (as indicated on the board) in the SoCkit. Connect the USB cable from the USB to UART pin (Next to RJ-45 ethernet connector) to the lab computer. Turn on the board. Open device manager in the computer. Wait until the system recognizes the device. You will observe that the device is recognized as a COMx under COM & LPT.



Now open putty (present in 8307/Putty directory). Select serial. Update the serial line field with COMx as observed in device managers. Set speed to 115200. You can save this configuration for future use. Click open. If nothing appears, close the putty session. Turn off the board. Again turn on the board and open an putty session immediately after turning on the board. Type root for login. And just hit enter if it asks for password.

Congrats! You successfully booted an HPS.



12) Play around with the example labs 1-3 present in the sdcard image. Refer to the manual provided by altera (available in moodle ELC_multi processor FPGA). Also, take a moment to understand the ".c" application codes. (Might be useful while developing you own application files)

Mmap, code snippet: (Hint: The address offsets can be obtained from Qsys)

```
fd = open("/dev/mem", O_RDWR);
led_address   = mmap(NULL, 0x10000, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0xff230000);
mbox0_address = mmap(NULL, 0x20000, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0xff260000);
load0_address = mmap(NULL, 0x10000, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0xc0000000);
load1_address = mmap(NULL, 0x10000, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0xc0010000);
close(fd);
```

14) Copy the soc_system.rbf, soc_system.dts and soc_system.dtb into the sdcard (preferably in a folder lab 4). Follow the final step [Updating individual elements on sdcard] in the following guide [Ubuntu/Mac]:

https://rocketboards.org/foswiki/Documentation/GSRD141SDCardArrowSoCKitEdition

Similar instructions to update any SD card can be found online for Windows. Do NOT copy and paste directly.

15) Using the above instructions copy the “u-boot-script.txt” and “make file” from lab1 to lab4, and any “.c” application file to lab4. Make the appropriate modifications in both the files:

Change the following from:

```
fatload mmc 0:1 $fpgadata lab1/soc_system.rbf;
```

to:

```
fatload mmc 0:1 $fpgadata lab4/soc_system.rbf;
```

And, from:

```
set fdtimage lab1/soc_system.dtb;
```

to:

```
set fdtimage lab4/soc_system.dtb;
```

Ensure the name is soc_system.dtb and soc_system.rbf.

In the makefile change the name to your “.c” application file name.

16) To create the u-boot script with u-boot header:

type “mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "U-Boot Script" -d uboot_script.txt ../u-boot.scr” at the prompt from lab4 directory

17) Reboot the board

– Before rebooting, ensure we remove mount points by typing the following at the prompt:

-- “cd /; umount /media/fat”

– At the prompt type “halt”

– Power-cycle the board by pressing the power button twice

– Note: You may have to reconnect terminal emulator after the board has been power cycled

18) Boot Strap NIOS

– Wait for the login prompt to reappear, and re-log in. (Username: root; no password)

– At Linux prompt, change directory to the first lab by typing “cd /media/fat/lab1”

– Ensure that the Altera GPIO kernel module is loaded by typing “modprobe gpio-altera”.

19) Change Software & Rerun

– Edit the “.c” application file. Please read the “nios_button code.c” in lab3 to understand how mmap() function works. Mmap function maps the physical memory to virtual memory. Understand the functioning of pthread.h (contains mutex), and Avalon threading.

– Save the .c file and recompile by typing “make clean all”

– Rerun by typing “./<name of .c file>” at the prompt

20) The NIOS processor can be programmed as usual (as seen in the earlier 3 labs).

21) In this lab, you need to accomplish the following tasks:

- ✓ Write 10 values [0 - 9] in the common onchip memory using NIOS.
- ✓ Using HPS, multiply each of the values by 3 and store the 10 new values in another portion of the common onchip memory.
- ✓ Read the values from the new location using NIOS and display it on the NIOS Console.

22) BONUS: Connect the datamaster of NIOS 0 to the same peripheral and demonstrate the information transfer using threading between the two NIOS processors and HPS.

Grille d’évaluation

Ce laboratoire compte pour 6% de la session

Utilisation du mutex : 3/6

Fonctionnement de l’application : 3/6

BONUS : 1 point

References

1. https://rocketboards.org/foswiki/Documentation/BuildingMultiProcessorSystems

2. https://rocketboards.org/foswiki/Documentation/GSRD141UserManualArrowSoCKitEdition

3. https://moodle.epfl.ch/pluginfile.php/1680498/mod_resource/content/10/SoC-FPGA%20Design%20Guide%20%5BDE0-Nano-SoC%20Edition%5D.pdf

4.https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu_nii51020.pdf

5.http://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/HPS_peripherials/pthreads_index.html

6. https://www.ee.ryerson.ca/~courses/coe838/labs/HPS-FPGA-Interconnect.pdf