HW/SW Codesign Projet with lowRISC/Rocket-Chip platform

I. Objectives

In this project, students will develop a demonstrator based on a RISC-V softcore, implemented on a FPGA. To do so, the students will have to develop, test and integrate different software components, inspired by the practical work done on the Rocket Chip platform in the implemented softcore.

II. Skills to be developed

- Putting into practice the knowledge acquired in the Embedded Systems Design Course and in the Rocket Chip TPs.
- Revisiting the HW/SW Co-Design flow of softcores on FPGA.
- Better understanding the HW/SW link through a hardware implementation on FPGA instead of a software simulation.
- Understanding the different subsystems of a typical embedded system.
- Developing and integrate specific software components adapted to an embedded target.

III. Overview of the expected end result

The aim of the project is to build a demonstrator highlighting the different skills acquired during the course and the practical work.

A.Demonstration

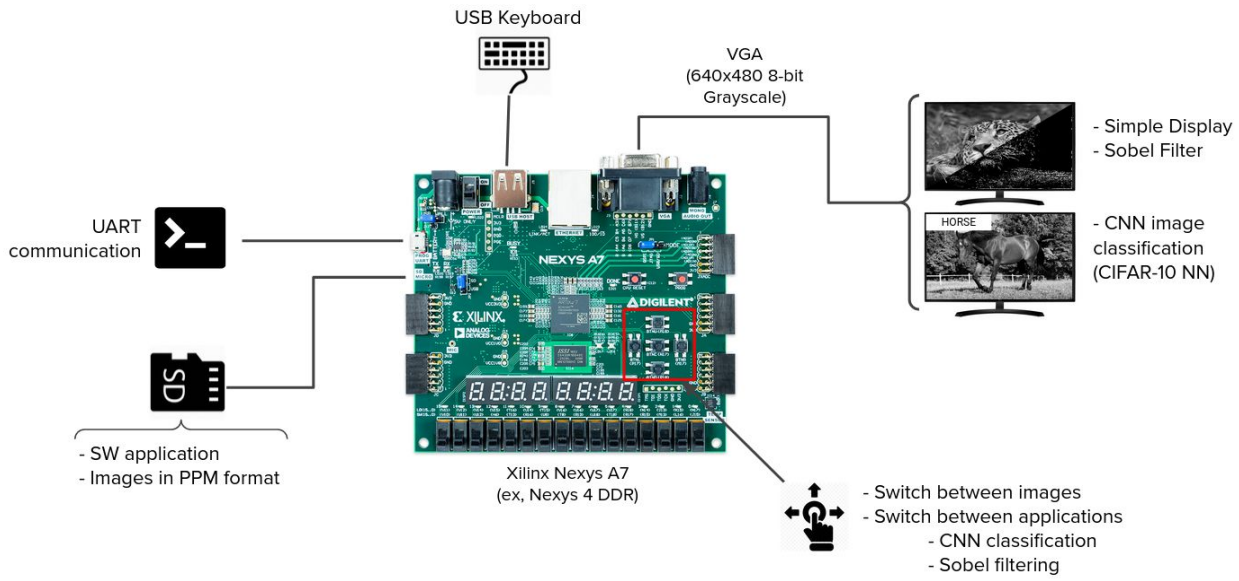
To get an idea of what is expected from you, we have prepared a video demonstration that you can watch on the following links, to get an idea of the deliverable expected.

Video: LeaRnV_RISC-V_based_embedded_system_AIT_SAID.mp4
Presentation: LeaRnV_RISC-V_based_embedded_system_AIT_SAID.pdf

B.General Description

- The demonstrator consists of an FPGA, connected to a display by a VGA cable, and an SD card containing the application and the processed images. An optional keyboard can be added to the set.

- The FPGA board is flashed by a containing bitstream:
    - A RISC-V softcore, of an RV64IMAFD architecture, with a "DefaultConfig" configuration and a hardware FPU, generated from a modified version of Rocket Chip and having a JTAG port to allow external debugging tools (OpenOCD, GDB, ...) to connect to it.
    - A set of hardware IPs in Verilog/SystemVerilog, developed by the lowRISC foundation are also included, namely: a UART module, an SD card controller, a keyboard controller.
    - A VGA controller (homemade) is also included. It has a BRAM memory capable of holding a 640x480 black and white (greyscale, 8-bit) image.
- The course of the demo is explained in the included video.



IV. Technical aspects

A.Documentation

To implement the project, a lowRISC technical documentation has been elaborated, accessible on the following link: lowrisc_project.pdf

You will find there:

- A presentation of the lowRISC open source project, developed within lowRISC.
- A presentation of the **Nexys A7** card (which was called **Nexys 4 DDR**).
- The stages of :
    - Bitstream generation (hardware) if required (**already done**)
    - Compilation and generation of the bootloader @0x4000_0000 (**already done**)
    - Compilation of the application @0x8000_0000: your demonstration will be in the form of an application, so it is mainly this part that you will have to see.

B. Pre-compiled hardware and project resources

A bitsream (hardware + bootloader) is provided. You can download it from here.
You will need to flash it on the FPGA board provided. The order will be sent to you later.

Bitstream: chip_top.new.bit.mcs
Archive of the entire project: lowrisc-chip-DATE2020-DEMO.zip

Command to flash the bitstream:

To flash the bitstream on the FPGA board, **Vivado 2018.1** must first be sourced. To do so, download and source the following configuration file:

Vivado 2018.1 configuration file: settings_vivado_2018_1.sh
Flash script: program_cfgmem.tcl

Then, on the host machine, run the following command.

```
vivado -mode batch -source program_cfgmem.tcl -tclargs "xc7a100t_0"
chip_top.new.bit.mcs
```

C. Files concerned

For this 2021 demonstrator, you will have to manipulate only the software part, whose files are available in the folder **<DOSSIER_LOWRISC>/fpga/bare_metal**
The archive can be accessed at the following link:
<https://drive.google.com/folderview?id=1hrl2jdR2Vth-LHSbGfdPNsoaX_dAuQHE>

The most important files :

- **Makefile: the *make application* command allows compilation**
- **Linker = test.ld : Check that the start address is on 0x8000_0000**
- **application.c: File containing most of the useful functions. To be completed.**
- **date2020_config.h : File containing definitions of global variables**
- **syscall.c : Necessary file for the resolution of the interrupt part**
- **ethlib.c : Necessary file for the resolution of the interrupt part**
- **lowrisc_memory_map.h : Useful file for the interrupt part**
- **ff.c : File containing utility functions**
- **The entire folder cnn_riscv_mancini/src completely describes the cnn application. More precisely, the file top_cnn_mancini.c calls the functions described in the other files of the folder. To be completed.**

V. Parts of the Project and Tasks to be done

A. Reading images from the SD card

1) Reading a file from the SD card
2) Reading the header and checking the format if it is a PPM3
3) Browse through the rest of the files and extract pixel values
4) Storage of pixel values in a global array
5) Closing the file
6) Repetition for each image

Functions to be modified

- **read_pic**, which describes the reading of a single image.
- **main**, which allows repetitive playback for each image.

Inputs and outputs

- **input :** loaded image files

- **output :**
    - Array **tab_length** of all sizes of images loaded. Its size is the number of image to read (here 14)
    - Array **tab_width** of all loaded image widths. Its size is the number of image to read (here 14)
    - Array **tab_size** of all loaded image lengths. Its size is the number of image to read (here 14)
    - Array **global_tab** of pixel of all the images arranged one after the other. Its size is = number of image to be read (here 14) * output image size (here 480*640) * 3 (for the 3 types of RGB pixels)
      The size of global_tab is therefore 14*(640*480)*3

- **Notes:** the *read_pic* function itself only takes one image file as input, and only completes the parts of the arrays concerned by this image as output. In order to fill the global_tab array image by image, the **pixel** array (size = image size *3) is used as an intermediary.

Files to consult

- **application.c**
  - The *read_pic* function is described in the section "Reading the image".
  - The main function describes the loop through all images.

- **date2020_config.h** for the description of the global variables associated with the dimensions.

Ideas to ensure proper functioning

Check image loading with picocom.
Possibility to print part of the arrays to ensure that they are correctly filled in.

B. Image pre-processing

1) Global variables: arrays of images ...
2) Adjust the dimensions for the CNN
3) Adjust dimensions for VGA display@640x480
4) Normalisation for the CNN: see section 4.2.2 of the technical doc.

Functions to be modified

- Declarations of *main.c* and *date2020_config.h* header variables

- **my_resizing**, to create an array containing size-adjusted images for the CNN

- **convert_to_greyscale** to create a greyscale image array suitable for VGA display

- **normalizing**, allowing the normalization of an image. However, the *top_cnn_mancini* function as described in the supplied folder takes a standardised tensor as input. If we wish to keep this structure, we will rather develop a **normalizing_tensor** function.

- **img_to_tensor** allowing the conversion of an image into a tensor to allow the application of the normalizing_tensor function.

Inputs and outputs

For the *convert_to_greyscale* function :

- **input** :
    - Array **global_tab** of all the pixels of the RGB images arranged one after the other, produced by read_pic.

- **output** :
    - Array **TAB_GS**, 2-dimensional, containing the pixels of the greyscale images, arranged one after the other. Dimensions of the table: Number of images X Dimension of an image here is 14X(480*640)

For the *my_resizing* function:

As this function is called in a loop on the global_tab array, it only works on one image at a time. So we have :

- **input** :
  image **source_img** corresponding to an RGB image of the global_tab array, size 640*480
- **output** :
    - image **resized_img** in RGB, resized according to the CNN algorithm, size 24*24

The *normalizing* function, or *normalizing_tensor*, is called in the same loop as the my_resizing function, and therefore works on a single image. When developing a normalizing function, we would input the resized_img image obtained above, to output a normalized normalized_img image.

Here we prefer to use tensors, so we will use the *img_to tensor* function to convert the resized image into a tensor, on which we apply *normalizing_tensor* to obtain a resized tensor.

Therefore, for the *img_to_tensor* function:

- **input :** image **resized_img**, resized image produced by my_resising

- **output :** tensor **resized_tensor**

And for *normalizing_tensor* :

- **input :** tensor **resized_tensor**

- **output :** tensor **normalized_tensor** , normalized tensor that can be used as input to the CNN algorithm.

Files to consult

- **date2020_config.h** for the declaration of global variables

- **application.c**
  - The header for the declaration of arrays and global variables
  - "Reading images" part for the convert_to_greyscale function
  - “CNN" part for the functions my_resizing, img_to_tensor and normalizing_tensor

Ideas to ensure proper functioning

You can to compare arrays via prints during conversion

C. Image display ("bypass" filter)

1) Selecting the image to be displayed and the filter (bypass)
2) Selecting the label to be displayed (bypass)
3) On-screen display of the selected image

Functions to be modified

- **display** which changes the parameters of on_screen according to the selected filter.

- **on_screen** which selects the label to be displayed over the image, and displays it on the screen
- **main** for the infinite loop allowing continuous display, as well as detection of the selected image and filter.

Inputs and outputs

- **input :**

  ImageSel and current filter **filterSel** indicators (as well as **previous_imageSel** and **previous_filterSel** indicators to apply the change if requested)

- **output :** Displaying the image and a label on the VGA screen

- **Note :** The *display* function selects the image to be displayed in the **TAB_GS** array according to the input indicator, and calls the *on_screen* function according to the selected filter.

  (If the selected filter is the edge detector, then the image will come from the **TAB_GS_FILTERED** array, and if it is the CNN, then *display* will call the *perform_cnn* function to obtain the result to be displayed on the label).

  Then, for the *on screen* function:

  input : image to be displayed (640*480 size), the filter used (and the CNN result if necessary)

  output : Image display

Files to consult

- **application.c** for the algorithm to be implemented, in particular in the "Display" part.
  - The main loop is in the main function.
  - Current image and filters indicators are changed by the *external_interrupt* function.

Ideas to ensure proper functioning

The operation can be checked directly by displaying the results on a VGA screen.

D. Development of the Sobel filter

1) Browse through the array of resized images
2) Application on each of them of a convolution function with kernel and bias corresponding to an edge detector.
3) Storage of the pixel values of the filtered images in a global array
4) Displaying an image with a Sobel filter

Functions to be modified

- **convolution_filter**, for the convolution algorithm applying the Sobel filter to an image. The kernel and bias declaration is made above this function.

- **main** for the repetition of this task for each image.

- Functions described in the Display section for display.

Inputs and outputs

- **input** : **TAB_GS** array of size 14X(480*640) containing the pixels of the images resized and arranged one after the other.
- **output : TAB_GS_FILTERED** array of size 14X(480*640) containing the pixels of the filtered images, arranged one after the other.

- **Notes :**
    - The *convolution_filter* function itself takes only one image of TAB_GS and returns only one image of TAB_GS_FILTERED. It also takes as input the kernel and the biases described above the function.

    - Display is performed as described in the “Image Display” section described above. If the *display* function input filterSel corresponds to the EDGE_DETECTOR value, then the *on_screen* function will take the filtered image from TAB_GS_FILTERED for display, and use the label corresponding to EDGE_DETECTOR.

Files to consult

- **application.c** for the convolution function.
    - The kernel is described above it.
    - The main file describes the loop through the entire pixel array

Ideas to ensure proper functioning

Apply the function for an image.

E. Development of the convolutional neural network (CNN) CIFAR-10

1) Step 1: Adjusting the dimensions of an image
2) Step 2: Writing a ConvolutionRELU function
3) Step 3: Writing a Maxpool function
4) Step 4: Application of 3 consecutive floors of ConvolutionRELU and Maxpool to the image
5) Step 5: Writing a reshape function and applying the image to it
6) Step 6: Writing a Perceptron function and applying the image to it
7) Determining probabilities and class
8) Display of the class over the image on a VGA screen

Functions to be modified

- **perform_cnn**, which will take an image of the *global_tab* array (in RGB), will call the functions *my_resizing*, *img_to_tensor* and *normalizing_tensor* to adapt the image as described above, then call the function *top_cnn_mancini* which will apply all the steps of the CNN application.

- **display** which will call the perform_cnn function if it receives CNN as the value of the filter indicator, and will apply this function (and thus the full application of CNN) to the image selected by display.

- **on_screen** will display the CNN result as a label on top of the tested image.

- **top_cnn_mancini** which will perform the different functions of the CNN and return the calculated class. This function is described in the file *top_cnn_mancini.c* in the folder *bare_metal/cnn_riscv_mancini/src* and uses functions written in the other files of this folder. These different functions describing the different stages of the CNN are also to be completed.

Inputs and outputs

- **input :** a **global_tab** image (or more specifically the position in the image in the table), size 640*480

- **output :** the calculated class corresponding to the image

- **Notes :**
    - The *display* function calling the CNN-specific functions requests as input the image selection flag **imageSel** (set to *perform_cnn* as input) as well as the **filterSel** flag of the CNN filter.

    - The function *top_cnn_mancini* asks for **the table of coefficients and biases** in addition to the normalized tensor **normalized_tensor**, and returns the **probabilities** in addition to the **class**.

      The *on_screen* function will take the calculated class as input in addition to the image from the TAB_GS table, and will thus display in the label the class over this image.

Files to consult

- **application.c** for the cnn adaptation and call functions of the image processing functions.
    - The image display functions, calling the cnn, are also in this file, in the “Display” part.
    - The functions describing the cnn itself are in the **cnn_riscv_mancini** folder.

Ideas to ensure proper functioning

Display the cifar results for several images on the terminal and observe the percentage of success.

F. Interruptions via push buttons

1) Push buttons and their connections on the board
2) Find the IDs of the interrupt sources
3) Define the necessary PLIC addresses
4) Developing the interrupt handler
5) Initializing the registers
6) PLIC interrupt initialisation
7) Writing the interrupt algorithm using the push buttons :
    a. If the west button is pressed, decrement the global variable imageSel, or set it to its maximum value if it was previously at its minimum value.

b. If the button is pressed, increment the global variable imageSel, or set to its min value if it was previously at its max value.
c. If the south button is pressed, the global variable filterSel is decremented, or set to its max value if it was previously at its min value.
d. If the north button is pressed, the global variable filterSel is incremented, or set to its min value if it was previously at its max value.

8) Infinite loop in the main
    a. update at the end of each loop of the previous_imageSel and previous_filterSel variable values to current values
    b. comparison with the previous values of imageSel and filterSel at the beginning of each loop.
    c. If there is a difference, then the display on the screen is updated.

Functions to be modified

- **external_interrupt** which will describe the part of the algorithm concerning the updating of the global variables according to the interrupts.

- **init_csrs** for the initialization of registers

- **enable_plic_interrupts** for initializing PLIC interrupts

- **main** for the part of the algorithm describing the comparison of the current values of the global variables to their previous value, calling the **display** function in case of a difference.

- You will need to browse the *test.ld*, *syscall.c*, *ethlib.c* and *lowrisc_memory_map.h* files to complete this part, although no functions need to be modified.

Inputs and outputs

- **input :** press one of the four cardinal buttons on the card
- **output :** Updating the values of the global selection variables **imageSel** and **filterSel**.

- **Notes :**
    - The *external_interrupt* and *init_csrs* functions will initialize **registers** and **PLIC interrupts** respectively.

    - The update on the *imageSel* or *filterSel* values identified in the main function will allow the *display* function to be called with the values of the two global selection variables as arguments, as well as the variables describing their value in the previous state.

Files to consult

- **application.c** for all the functions to be completed, in the "Interruptions" section, as well as the main.

- Exploring the files **test.ld**, **syscall.c, ethlib.c** and **lowrisc_memory_map** is necessary for the realization of this part.

Ideas to ensure proper functioning

Associate interruptions with an easily identifiable reaction on the terminal.

G. Bonus

1) If there is time left, doing polling with the keyboard entries <up/down/east/west> ...
2) If there is still time, making a filter to generate fractals.
3) If there is still time, creating an algorithm for reading images in .jpg format.

- The code describing these ideas has not been implemented, so there is no code to guide you if you decide to deal with them.

- For polling, you can use the hello.c file, which contains a driver that allows the keyboard to be read in ps2 format.

- For the filter, you can write the algorithm for it and then add it as an option in the display functions.

- For reading the images, you can write an additional case based on the reading of the header.