COMP.CE.400

System Design

Final Report

Group 05

Trinh Gia Huy – H290290 – [giahuy.trinh@tuni.fi](mailto:giahuy.trinh@tuni.fi)

Phan Vu Thien Quang - - quang.phan@tuni.fi

# Overview of the System

Describe here what kind of system you implemented in the exercise project. What does it do and what are its main parts and features? Focus only on the implementation on the FPGA board, not the design process, like SystemC simulations. You may illustrate the description with an image, if you wish.

## Hardware

Describe here the hardware components of the system. Obviously this includes the FPGA board, its add-ons and the PC, but more importantly you should describe what resources (processors, FPGAs, memories, HW accelerators, buses etc. if any) are used on the FPGA board and how they communicate with each other.

## Software

What software modules are used by the system and what is the purpose of each one? This includes programs and processes that are running when the system is on. How are they mapped on the hardware resources and how do they communicate with each other?

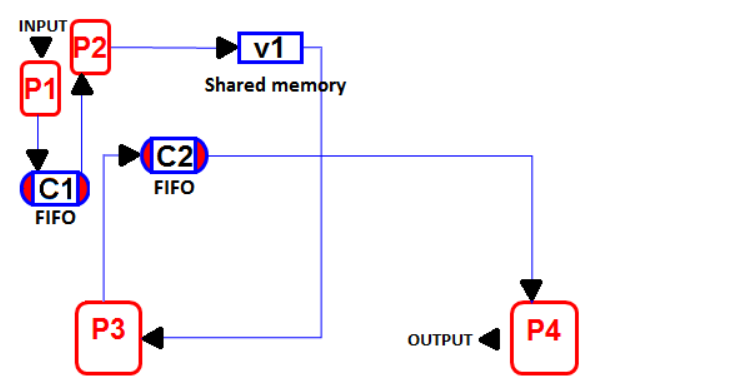
# Summary of the Exercises

Write here a short summary of what you did in each of the individual exercises. Emphasize the big picture: what was the purpose of the exercise, what tools you used and what you did with them, and what was the end result. A couple of paragraphs per exercise is sufficient. The purpose of this section is to ensure that you have a good general picture of the exercise project as a whole.

## Exercise 1

The main objective of exercise 1 is introducing us to new tools and basic knowledges of system design such as profiling Kvazaar performance with gprof profiling and encoding HEVC video. The tool, which is Kactus2, is used to generate Makefile for Kvazaar compiling, configure different Kvazaar system (i.e: PC-system, Veek-system, …) , integrate and reuse the Intellectual Properties(IP) in order to increase the hardware development productivity. At first, we would configure the library and prepare some video input for encode. Next step, we would compile and run the Kvazaar using pre-made script encode\_run.sh. The output encoding video result will be logged into external files for data parse. By analyzing these log files with different quantization value QP and running profiling build to create the gmon.out output file, we could make some conclusions on encoded frames per second(FPS), the effect of decreasing/increasing quantization value, and average PSNR in both profiling and non-profiling build run. In addition, the generated file from profile\_image.sh script would provide us with the illustrating profiling image which we could summarize the executing time used within as well as the time used for recursion of each functions in percentage.

## Exercise 2

The main purpose of this exercise is to get us familiar with SystemC modeling, simulating and visualizing using GTKWave. At the very beginning, we start checking the functionality of SystemC library by running the Hello World example. Next, we would have to run and examine the PC-application by dividing it into the inter-communication SystemC process and add timing to the model. To be precisely, we start verifying application’s functionality using pre-made synchronized testbench by check the difference between the input and output. Since the purpose of the app is to receive, permutates, encrypts the input with a key and undo all these processes, the output must be indifferent compared to its input. Compiling the system requires a make file from Kactus2 as well. Next, we would have to implement the decrypting processing (process 3 and 4) based on stub code for encrypting process (process 1 and 2) as image below.

**Figure 1**: The app process

The channel 1 and 2 will be modeled as FIFO to implement the communication between process 1-2 and process 3-4, while data polled for meta data can be accessed via shared memory v1. The GTKWave software simulate the application by generating waveform of signals. Next task is to optimize the overall application performance by interchanging the CPU between process 1 and 4 and change the bus interface of all 4 process from I2C to AMBA. The bonus task\*\*\*\*\*\*\*\*

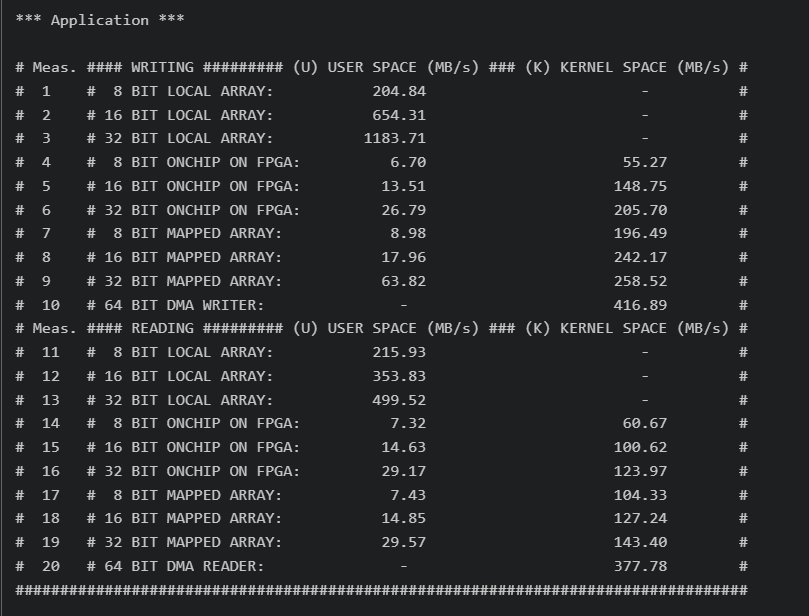
## Exercise 3

This exercise comprises 3 parts, which are implementing the inter-process communication of the previous application (CrypterSimApp) using TLM Socket, some system call (read/write) used as interface between the encoder and hardware accelerator and accelerating the system performance. The socket transfer between modules at transaction level modeling would be more complex compared to previous exercise which is FIFO. To be precise, in socket transfer, the process 1 would be master (initiator) while the process 2 acts as slave (target). Besides, the data which is passed in this payload must be C++ data type and memory reservation should be taken into account. The encoder would run on user space while accelerator calculate the prediction on FPGA. The function read() and write() will read and write to the correct chosen based address where transferred data are saved when implement b\_transport() function. To handle event, the read() function has to wait for interrupt request from accelerator when data is ready. The average PSNR result that yield from running the system using script with same previous QP value should be same with previous exercise. In addition, the bonus task require us to optimize the performance of Kvazaar intra prediction by dividing the function intra\_get\_angular() and its SystemC thread for parallelism run.

## Exercise 4

The main objective of this exercise is to introduce to interfacing with FPGA from Hardware Processor System (HPS) by controlling led with button and get familiar with Quartus Project and Qsys. Qsys is the bus design tool that integrated with Quartus Prime software which allow connection to Altera Avalon bus and provide bridges to HPS via AXI bus. At very beginning, we can add FPGA components into premade soc system design using Qsys tool. In more detail, in library we add 2 parallel input/output components (PIO) for led and button with their clock, reset and ports connected to HPS ports. The s1 port (acted as slave) will be connect to h2f\_lw\_axi\_master of HPS via light-weight AXI bus and base addresses can be set. Next, we would generate Qsys and compile the Quartus project into raw binary file (.rbf) which will be uploaded into board. On the other hand, we need to implement and test the blinking software on virtual machine and transfer to the board as well. The function read\_btn() from BlinkerApp.c will read the button’s input value and set\_led() function will write it into the led for displaying. Since physical memory cannot be directly access the software, it should be mapped to virtual memory with correct regions. In addition, we use Kactus2 to configure the ARM cortex, generate and compile Makefile for the system. Since BlinkerApp\_0 file will not be executed on VM due to lack of binary file, we would have to transfer and run the executing file on the board. The led should be displayed when we push either the button or both simultaneously.

## Exercise 5

The main target of this exercise is to examine into the memory system and mapping which includes the DMA and kernel space measurement using Signal Tap II tools. At first, we start generating the Makefile using Kactus2 software for the premade benchmarking software with configuration for arm cortex and compiler gcc tool chain. The provided image of FPGA (which is a raw binary file) and application should also be upload to the board as well. Running the executable on the board will print out the benchmarking result which comprise the different memories reading speed. However, some zero results need a new kernel driver made for the benchmarking hardware to register. The implemented kernel driver benchmark\_driver.ko can now be loaded and running the application would yield a complete benchmarking result as below.

**Figure 2**: Benchmarking result

From figure above, there are 2 Direct Memory Access (DMA) which implemented on FPGA and can read and write from/to the HPS memory (DDR) directly. On the contrast, we need to measure this DMA reading/writing speed as accurately as possible using SignalTap II. In detail, we have to measure the clock cycle it takes from address read/write start to be valid to the end. At this point, all these measurements could be marked down using Sticky Note Tool in Kactus2 and memory allocated for ARM as well. Additionally, the speculative processing power should be taken into account by comparing 16 bits multiplication (in MIPS) of CPU and FPGA. We need also to measure accelerated FPS using the relation of processing power to VM with assumption of 40% executing time on ARM CPU rather than FPGA. And following that, the pure software FPS can be measured using relation between ARM MIPS and VM MIPS only. Finally, run Kvazaar on board with both profiling and non-profiling build Makefile with video sequences transferred to the board already. This will encode 10 frames but with different average PSNR value comparing to previous exercise, which results from our performance concern only.

## Exercise 6

The objective of this exercise is setting in order to match the performance of Kvazaar on Terasic Cyclone V Veek board, exploring the challenges design Kvazaar space with SystemC simulation, and evaluating what it takes to encode a 25 fps full HD video. At first, we need to test and find the right value of delay\_c so that the simulated FPS of untimed model matches with the one on ARM (17000). Next we will run Kvazaar on ARM with timed HW-accelerator with declaring variables (hps\_to\_fpga\_ns\_per\_byte\_c, hps\_ddr\_to\_fpga\_ns\_per\_byte\_c,…) from previous exercise. In addition, we may add the calculation to function b\_transport in sc\_kvazaar\_ip\_sub.c source file. Since the Gprof will not help us to know the execution time (in percentage) different parts of encoder, it is a necessity to implement a simpler profiler in function sc\_kvazaar\_main with EXPLORATION\_SW/EXPLORATION\_HW defined in sc\_kvazaar.c file. Thus, you can verify its functionality by simulating Kvazaar on ARM and compared with the one added to exploration.h.\*\*\*Big simuation\*\*\*\*\*

## Exercise 7

The exercise comprises multiple tasks, which are creating the QSYS component with intra prediction HW accelerator and camera control. The HW accelerator driver will be used with Kvazaar and camera driver will be load with user application. At the beginning, we start creating and compiling the Kvazaar\_qsys component following the instruction using qsys tool in Quartus Prime software. When the the HW is ready, we can export the top-level design, generate Quartus project for synthesizing and HTML documentation for the design. When synthesizing the Quartus Project, we have to set our devices’s assignment as global using file cyclone\_v\_setting.qsf. Beside, the timing constraint file Kvazaar\_IP\_acc\_Camera.sdc have to be added. After all, we run Analysis & Synthesis to complete and verify its functionality using provided Tcl scripts. The resulted raw binary file (.rbf) can be loaded to the board for synthesizing. On the other hand, we generate the accelerated Makefile and compile the Kvazaar as previous. The camera driver (camera\_driver.ko) and kernel module (ip\_acc\_driver.ko) along with other files and scripts have to be loaded into the board as well. To test the HW, we load the kernel module and run script to encode the video while run the user space application with the camera driver loaded as well. The recorded result would be similarly as below.

## [Video encode 1] [Video Encode 2]

## Exercise 8

The final exercise asks us to stream the video from the board and profile the hardware accelerated encoder. At first, we start creating profiling build for accelerated Kvazaar and run it similarly to exercise 7, which also yields out the file gmon.out. In VM, we run encode script to create the profiling image. In next task, we start by creating a non-profiling build of accelerated hardware. The resulted Kvazaar executable file and the previous encoding video should be load to the board and preparing streaming using ffmpeg. Also, we start creating our own shell script for streaming and viewing stream on VM using ffplay command. Finally, while streaming, we could measure the FPS and latency of system performance based on different video size input and camera resolution setting.

# Suggestions and Feedback

Give us some suggestions on how we could improve the course and the exercise project. Even this section will affect the grade for this exercise so be thoughtful and constructive.

## What Did We Learn

What did you learn on the course as a whole and on the exercise project in particular? Did you learn everything you expected from the course? Were some topics missing?

## What was Easy/Hard

Which parts of the exercise project did you find the easiest? On the other hand, what was hard and time-consuming?

## What Needs Improvement

We think the guiding TA session should increase since 2 hours guidance is not really much to ask questions and cover almost groups that need help. If possible, employ more TA would be perfect.