Report Lab Assignment
Running x264 on Host Processor
Extracting and Accelerating Kernel on Co-processor

Imara Speek (150000)
Aimee Ferouge (4014030)

6 november 2013

Date Performed: October, 2013
Instructor: Ir. A. Brandon

Samenvatting

10 lines of Abstract text

1 Introduction

For the lab a computational intensive kernel has to be extracted from a x264 software application, free software library for encoding video stream into H.236/MPEG-4 AVC format. The application is executed on a FPGA running a MicroBlaze host processor. By running the particular kernel on ρ-VEX co-processor, the execution time can be decreased by making use of hardware acceleration.

1.1 Getting Used to the Environment

The first lab is meant to get used to the project environment. A few input files are given to show the compile and run commands of the x264 application, by inserting a .y4m stream and creating a short .mkv movie. By adding the gprof flag to the compile command, a list is created of all functions ordered by their share of the total execution time (in percentage).

1.2 Detecting the Computationally Most Intensive Kernel

Profiling the x264 execution for the .y4m files that are provided by the lab, the following ranking is obtained: Given the chart in 1, we decide to extract the x264_pixel_satd_8x4 kernel, since its share in execution time increases as the files become larger. This kernel evaluates the Sum of Transformed Differences between a 8x4 pixel block from the input stream and reference blocks using 4x4 transform (Hadamard???).

1.3 Executing an Application on the Development Board and ρ-VEX

When executing ./configure, a file is created for configuration of the application. However, this config.mak file is made for applications running on the guest (Ubuntu). In order to configure for MicroBlaze, the config.mak file has to altered. All references to m32 have to be removed and the –DWORDS_BIGENDIAN flag has to be added to the CFLAGS variable. This has to be done everytime when configurating the application for MicroBlaze.

Input fie	Execution time (in sec)	Share (in %)	Kernel name
eledream_32x18_1.y4m	0.02	100.00 0.00 0.00	x264_analyse_init_costs x264_free x264_cabac_encode_desicion_c
eledream_64x32_3.y4m	0.03	66.67 33.33 0.00	x264_analyse_init_costs x264_pixel_satd_4x4 x264_pixel_satd_8x4
eledream_640x320_8.y4m	1.61	14.29 11.80 4.97	x264_pixel_satd_8x4 x264_get_ref x264_pixel_satd_x4_16x16
eledream_640x320_32.y4m	8.54	20.61 13.23 4.57	get_ref x264_pixel_satd_8x4 x264_pixel_satd_x4_8x8
eledream_640x320_128.y4m	29.86	17.48 14.17 6.56	get_ref x264_pixel_satd_8x4 x264_pixel_satd_x4_16x16

Tabel 1: Chart with computationally most intensive kernels for each input stream.

After doing this, the application now can be 'made' for MicroBlaze by first moving to the Scratchbox 2 environment for MicroBlaze and then execute the make command. Fig. **??** shows a block diagram of both the platforms.
In order to run the application on the MicroBlaze host processor, the x264 executable must be compressed along with an input stream in a tar.gz file. This file can be put, via the FPGA host machine, on one of the three FPGAs using the scp, ftp and put command. For the programmer to connect to the development board, one must use the ssh and telnet command. See also fig 2.
For the first lab, the satd_8x4 kernel has to be:

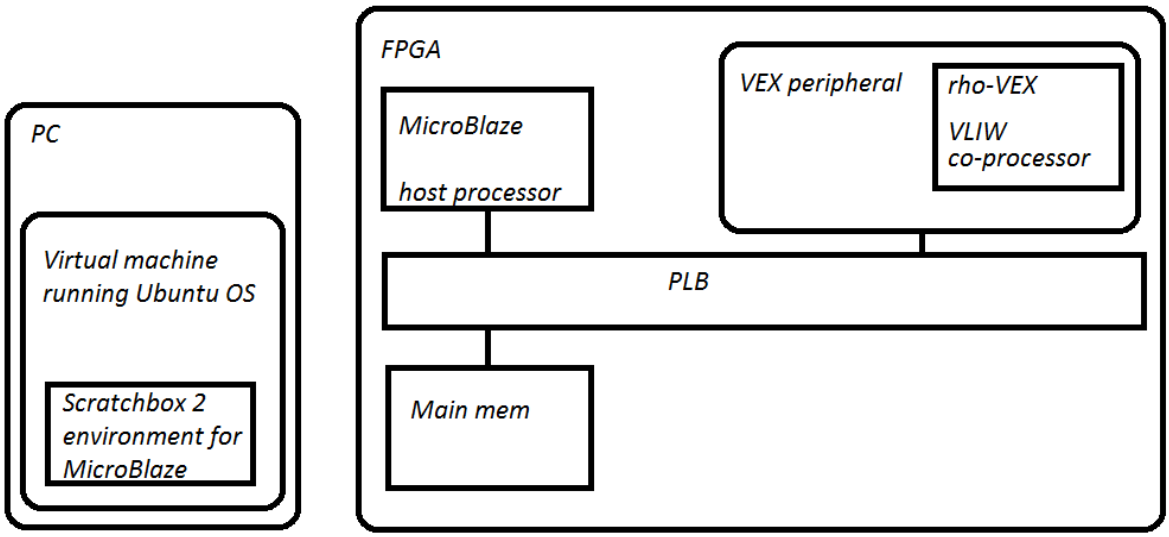
- extracted into a separate file
- supplemented to a proper .c file which can be compiled and run using the ρ-VEX
- included in the makefile that creates a bytecode and bytedata file of the kernel
- debugged

Looking back, the debugging part has been chasing us all the way through the lab. Not only did we suffer from bugs in our source code, the ρ-VEX itself did also have shortcomings that had to be circumvented by downloading several fixes, competing for FPGA availability and break downs of the entire host machine due to unsufficient capacity for the amount of students.

2 Implementation

Our extracted kernel can be found in **??**. In order to make the extracted kernel qualified for compilation and execution, a few things have to be altered. First, the new file has to be recognized by the makefile in the rovex-examples directory. By executing make byte-¡kernel¿ two files are created:

- bytecode, containing all the intstructions to be executed by the ρ-VEX
- bytedata, containing the pixels of the input stream

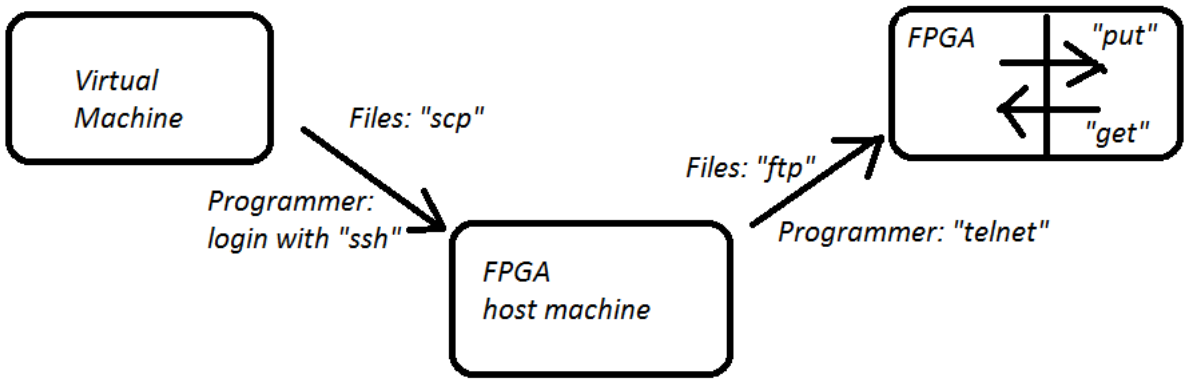


Figuur 1: Block diagram of the Virtual Machine and the ERA platform

Second, some type definitions have to be made. Originally, pixel_satd_8x4 resides in the pixel.c file of the x264 application. When extracting this kernel, all prior knowledge is lost and has to be defined again. Then, in order to make the kernel compile and run on the ρ-VEX, the development board has to be reset and started. Bytecode has to be written to the instruction memory (rvex-imemory) and bytedata to the data memory (rvex-dmemory). Finally, the calculated result should be returned to the host. Figure X shows the ρ-VEX memory layout for our kernel.

2.1 Adjusting the makefile

In order to make the makefile recognize the extracted kernel, the file can simply be added to the EXECUTABLES. Other files are already in the makefile (e.g. adpcm), but these can be removed since we will not need them for our application. An important issue is the difference between logical memory and physical memory. While the first address of a logical memory is obviously zero, the physical memory can have the first part of the register being occupied by the operating system creating a so-called stride. For the ρ-VEX, this stride appeared to be 120. Thus, when writing for example the result of the kernel to the logical address '0', it actually should be written to the physical address '120' of the data memory. This can be done using __DATA_START to indicate the start address.
Despite the fact that this is a rather simplistic operation, it took us some struggling to have this action confirmed as correct. For example, we were told to remove the AUTOINLINE flag which resulted in a lot of wrong hexdumps. Also, half way the lab a fix has been made to eliminate the issue of the stride. All references to __DATA_START had to be removed again, which felt like we had wasted lots of time. Unfortunately, we still had major problems getting the application to run correctly. As it turned out, the fix had only affected the 'home' directory. In order to avoid conflicting files, we had created a separate folder named 'lab2' from where we executed the application. Due to this, the fix did not reach our code and thus did not eliminate the stride issue.



Figuur 2: Ubuntu commands for putting files on and navigating to the FPGAs

2.2 Type Definition

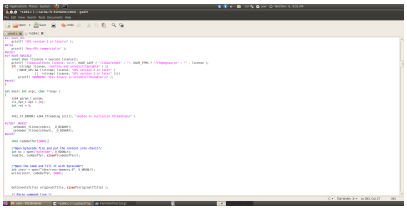
As stated earlier, an extracted kernel is unable to obtain information from previous code. Consequently, all parameters have to be pre-defined. Table 2 shows all required type definitions.

Parameter	**Type definition**	**Motivation**
intptr_t	unsigned int	This parameter represents the stride, which cannot be <0
pixel	unsigned char	Pixels are made out of bytes, which have 8 bits (like a char)
sum_t	short int	Same type definition as in the source code (16 bits)
sum2_t	long int	Same type definition as in the source code (32 bits)
BIT_DEPTH	defined as 8	Also in the source code, plus the kernel handles pixels (bytes)
BIT_PER_SUM	(8 * sizeof(sum_t))	Also predefined as in the source code, being 8 as well

Tabel 2: Type definitions in the extracted kernel file.

2.3 Communication between the MicroBlaze and the ρ-VEX

In order to delegate the satd_8x4 kernel from the MicroBlaze to the ρ-VEX, both the environments need to communicate with each other. When executing the x264 application, the MicroBlaze has to load the instructions of the extracted kernel into the instruction memory of the ρ-VEX and the data (for which the SATD has to be evaluated) into the data memory. This is when the x264 becomes involved. The pixel_satd_8x4 kernel, which is residing in the pixel.c file of the application, needs to be overwritten. Instead of calculating the SATD, it should send the instructions and input to the ρ-VEX. The commands **??** and **??** are therefore added to the pixel.c file of the x264 application.



(a) Bytecode

(b) Bytedata

Figuur 3: Commands for loading the bytecode and bytedata into the ρ-VEX