ComS / CprE 583 – Reconfigurable Computing

MP-3: Reconfigurable Coprocessing

Assigned: Friday of Week 7
Due: Friday of Week 10
Points: 100 + bonus for fastest image processing implementation

[Note from Joe: The goal of this Machine Problem is for you to work with your group to gain experience in three main areas:

1. *Accuracy and performance issues with floating-point and fixed-point representation.*
2. *Reconfigurable coprocessor design, and testing / integration with a real-world embedded system.*
3. *Translating software code for use in a hardware accelerator.*

Note that this is a three-week assignment, as the design challenge provided is more open-ended than in MP-1 and MP-2, and the LEON-based hardware/software infrastructure will take some time to become accustomed to.]

Reading: Chapter 23 provides a reasonable overview of fixed-point representation issues, although the automated analysis described there is unnecessary for the video processing system we are implementing. The first part of chapter 26 presents a high-level overview of hardware/software partitioning that is reasonably close to the type of optimization we are attempting here. Finally, pages 734-758 of the GRLIB IP core user's manual contains a succinct description of the LEON3 microprocessor that we are using.

1) Platform Overview. Figure 1 shows the high-level structure of the MP-3 hardware setup. For the first part, you will generate a hardware configuration (bitfile) for the FPGA using the existing VHDL source (the SPARC open source processor LEON), download that bitfile to the FPGA, and then test that the initial design works before making modifications in subsequent sections. This design is significantly more complex than MP-1 or MP-2, but we only have to edit a small part of it: `coproc.vhd` and `coproc_core.vhd`.

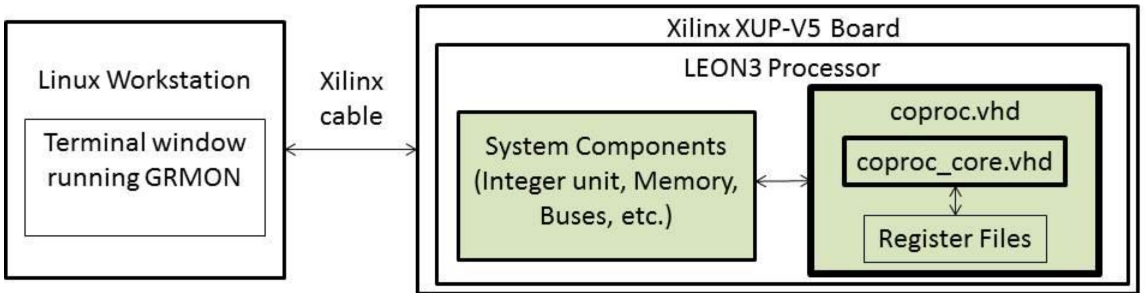


Figure 1: MP-3 Hardware Setup

2) Software Overview. After using NX client (or software of your choice) to connect to an appropriate Linux machine, download the `MP-3.zip` file from Blackboard into your ISU home directory. Don't forget to source the MP-3 specific `setup.sh` file each time you open a new terminal – for your convenience a copy of `setup.sh` is included in the `MP-3.zip` attachment.

The Xilinx ISE project is located in `MP-3/hw/designs/leon3-xilinx-ml509/leon3mp.xise`, and can be opened using the `ise` command:

```
ise leon3mp.xise &
```

Review [Jon12A] as well as MP-1 for instructions on how to connect to the FPGA and work remotely. MP-3 is structured in a different manner than MP-1 or MP-2. A clean build of the LEON codebase requires the following commands, and takes approximately 17 minutes to complete when the files are housed in the `/local` directory:

```
cd MP-3/hw/designs/leon3-xilinx-ml509
make clean – delete temporary files to ensure a clean build
make scripts – create compilation scripts
make ise – use Xilinx Synthesis Tools to generate the bitfile
```

After the build is complete, the `leon3mp.bit` file can be loaded onto the FPGA using impact. To interact with LEON, the interface application GRMON must be used. **Do not have impact and GRMON running at the same time. This causes the board to lock up and it must be disconnected.** To start GRMON, run the following commands:

```
cd MP-3/sw/board – this directory is most convenient for loading programs
grmon-eval -xilusb -u – connect to LEON using the Xilinx USB cable and send
                        output to the console
exit - exit GRMON; this must be done or the board will be locked for other users
```

If GRMON prints error messages saying JTAG input is invalid, reload the bitfile and try again. For some reason, this seems to happen often if you do not press "Enter" after running impact as a background process. To load and run programs, you can run the following commands:

```
load program.leon – load the compiled program.leon into program memory
run – run the program currently loaded into program memory
(ctrl+c) – interrupt the currently executing program
```

If a program triggers a hardware trap (e.g. tt=0x2b Data Store Error) or is interrupted, you can enter the command "`cont`" to continue execution. The program may need to be loaded again work properly and to restart the program counter.

When working remotely, the FPGA monitors can be viewed by accessing the following three web cameras using your browser:

- http://xilinxcam-1b.ece.iastate.edu to see the monitor for the xilinx-1 FPGA board
- http://xilinxcam-2b.ece.iastate.edu to see the monitor for the xilinx-2 FPGA board

- http://xilinxcam-3b.ece.iastate.edu to see the monitor for the xilinx-3 FPGA board

The username and password for each of the camera web interfaces is “xilinxuser/drjones”.

3) Hello World. Your first task is to ensure that the base design works. All the programs you will write go in `MP-3/sw/board`. Write a simple C program, `hello.c`, using printf to output “Hello world!”

LEON uses the SPARC ISA, so our applications must be cross-compiled. To compile your application, open the Makefile in `sw/board` and edit/add the bold parts of the following lines:

```
all: frameloop hello

hello: hello.o report_device.o
    $(CC) $(CCOPT) -o hello.leon hello.o report_device.o
```

Adding the target “hello” uses the GCC cross compiler for SPARC to generate a file that can be read by LEON: hello.leon. If GRMON is opened properly and everything is set up correctly, you should see the output of your program in the terminal when you load/run hello.leon.

4) Software Grayscale Conversion. Your next task is to modify the `frameloop.c` code to convert image pixels from color to grayscale. Browse through the main function and provide a description of what it does. Then browse through each of the provided mode functions and describe what they do. This step is important to understand how the code you write works on each image it processes.

To generate the image stream used for this MP, run the following commands:

```
cd MP-3/sw/host
make
source avi2raw.sh
```

This will take a video from MP-3/sw/data and turn it into a series of images that can be loaded onto LEON. Once they are generated, run the following commands **without interruption:**

```
cd MP-3/sw/board
grmon-eval -xilusb -u
batch ../data/loadvideo.bat
```

Loading images can take several minutes, but once they are loaded, they generally do not need to be loaded again unless that memory region has been overwritten (which may happen depending on your software code). As we are using DRAM for storage, the images can stay in memory even after a board power cycle, although this is not guaranteed.

By default, LEON does not configure its video output settings. In order for anything to show up on the monitor, you must run the following command in GRMON:

`i2c 1 dvi init_ml50x_dvi` – enable i2c driver 1 to use dvi output format

To verify the complete setup, compile the frameloop program, load it, and verify output goes to the monitor. If using the Coover 2041 lab you will need to set the second monitor to digital input (source 3).

Using the floating point grayscale conversion mode as an example, develop a new mode that uses Q2.14 fixed point to do the conversion. A good implementation will be much faster than the floating point version because LEON does not have a floating point unit, and must emulate these instructions in software. In your writeup, provide a run-time comparison between the floating-point and fixed-point modes.

5) Hardware Coprocessor – Grayscale Conversion. Browse through `coproc.vhd` and `coproc_core.vhd` in ISE and provide a description of how the controller sends data to the core. Also describe the basic structure of the core. This part is important to understand how the coprocessor receives input, when it should operate, and what it should do.

In order to simulate applications on LEON, they must be loaded into PROM. To do this, go to `MP-3/sw/board` and change the APP variable to the name of the program you want to simulate; in this case "frameloop". This will create and copy s-records so they can be loaded for simulation.

LEON cannot be simulated from ISE, but uses its makefile system to generate the necessary files. To simulate LEON, run the following commands:

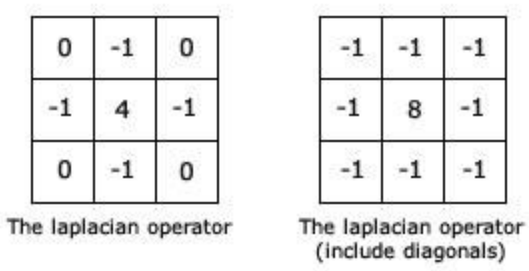
> `make prom` – make s-records for use in simulation (run from `MP-3/sw/board`)
> `cd MP-3/hw/designs/leon3-xilinx-ml509`
> `make vsim` – use ModelSim in batch mode to compile LEON for simulation
> `vsim testbench &` - use ModelSim to simulate

When frameloop is simulated, there are no images loaded into the memory of LEON, so no meaningful sample input is given. Instead, you must change the main loop in frameloop to generate sample data in the regions you want to test. Test the first 10-20 values of an image address at a time, or the simulation time becomes too long. You will also need to change the amount of loops your mode function performs to 10 or 20, or it will read invalid data and take an extremely long time to simulate.

Your objective is to design a hardware/software system that uses hardware acceleration (i.e. the coprocessor) to implement the fixed-point functionality. Create a hardware diagram for this component (either a dataflow or a simple FSM), and describe how your architecture processes pixels, and including how it integrates with the main LEON processor. In order to do so, you will need to use the coprocessor interface instructions: ldc, lddc, stc, stdc, and cpop1. Their use is documented in `MP-3/sw/board /coproc.h` and the SPARCv8 manual. Provide a brief explanation of the arguments to these functions and what they do.

6) Software Edge Detection. Your next task is to modify the `frameloop.c` code to detect edges on grayscale images. You will copy the grayscale image into a buffer and then operate on this buffer to preserve the original data.

Laplacian edge detectors are a simple and computationally fast way to get the edges out a grayscale image. The Laplacian operator basically takes the second order derivative between a pixel and its neighbors to determine if there is a large color difference (an edge). Figure 2 illustrates the kernel used to perform this operation. We will use the diagonal inclusive kernel.



0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

-1	-1	-1
-1	8	-1
-1	-1	-1

The laplacian operator (include diagonals)

Figure 2: Laplacian kernels

This kernel multiplies the center pixel by 8 and subtracts all of its neighbors. If the result is less than 0, there is no edge, and if the result is greater than numbers we can represent, it is definitely an edge.

Use the Laplacian kernel to modify the frameloop application to add a mode that performs edge detection on a grayscale image. You will need to use a grayscale mode function that stores to a temporary image, then operate on it with the frame buffer as the output. Try to avoid reading every pixel several times per row of the image. In your writeup, describe your software implementation and its performance on the LEON processor.

7) Hardware Coprocessor – Edge Detection. Your next task is to modify `coproc_core.vhd` and `frameloop.c` code again to create a coprocessor that performs the grayscale edge detection.

Edge detection is a bit more of a complicated operation than grayscale conversion; there is a lot more input data, but the bandwidth of input to the core is limited. You must decide how much data you will send in at a time to the execution unit, where it will come from, and how many operations your edge detection will use. Op-code space is not an issue, so you can use as many operations as you like, but Laplacian edge detection must be used to compute the value for every pixel.

To change how input is received by the execution unit, you will have to understand the cpop operations state machine in `coproc.vhd`. Extra setup and output processing code will need to be added to it, but it will probably be pretty similar to the existing code. You will also need to add logic for the coproc input control. After deciding on the data format, you can write the processing code in the core.

The software part of this task is a bit more complicated now that operations on pixels are not done in place. You must be aware of Endian storage issues, where data is located, and how it is stored and read from the coprocessor.

Design a hardware coprocessor to perform Laplacian edge detection, and integrate with the LEON coprocessor interface. Create a diagram for the main coprocessor core component, and in your writeup, defend your choice of input format and describe how the operations you added process pixel(s). Also describe how the software sends and receives data to/from the coprocessor. Provide an analysis of the performance of this coprocessor and the speedup provided over the software-only implementation. **Bonus** points will be provided for the fastest coprocessor implementation in the class.

Summary – What to Submit. Create a zip file containing any modified and newly-created .vhd and .c files. Attach this zip file with your writeup as a PDF file. Make sure the writeup covers all the highlighted sections in this document. In the Blackboard submission field, list each team member with a percentage of their overall effort on MP-3 (with percentages summing to 100%).

Summary – What to Demo. At least one group member must be available to demo your MP-3 applications running on the FPGA and to review aspects of your implementation. A full demo score requires correct results when running the MP on the FPGA board, but partial credit will be given for effort. Be prepared to briefly discuss your source code.

APPENDIX - Tips and Tricks for ISE/ModelSim

ModelSim:

1. If there is a particular window layout you like, you can arrange them how you want, select *Layout->Save Layout As*, and choose a name for it.

2. If there are signals you know you want to add every time, add them in simulation and while the simulation window is active, select *File->Save Format*, and choose a name. From now on, when you want to load that format, in the main ModelSim window, you can select *File->Load->Macro File* and choose the file.

3. To add variables to the wave view, choose the component and its process you are interested in. Select *View->Locals* to show the variables of that process, which can be added to the wave window just like signals.

4. ModelSim 10 supports fixed point radix. To show this notation, pick the signal in the waveform, right click it, and select *Radix->Fixed Point*.

5. You can make long signal names shorter by removing their parent information, i.e., you can select *Format->Toggle Leaf Names* to shorten them.

6. To add breakpoints in code you can right click on a wave and select *Object Declaration*, which will open the related file. Once the file is open, you can select a line that is numbered on the left in red as a breakpoint. To create it, simply click the white space to the right of the red line number, and the code will stop at this point in execution. You can left click on that breakpoint at any time to make it no longer stop at that point or right click and select *Remove Breakpoint* to delete it.

7. To add wave breakpoints you can right click on a wave and select *Add->New Breakpoint*. The execution will stop when that signal changes values.

8. You can search for a specific value in a waveform by selecting the signal of interest and then selecting from the top menu *Edit->Signal Search*.

ISE:

1. LEON has a manual compile order and does not show up as a hierarchical design for ease of viewing. You can view the files in alphabetical order using the *Files* tab of the Design Explorer.

2. You can comment out multiple lines at once by selecting them and pressing *Alt+C*. This can be undone with *Alt+Shift+C*.

3. You indent multiple lines at once by selecting them and pressing *Tab*. Lines can be un-indented with *Shift+Tab*.