HW/SW Co-Design
Main Task - Ray Tracing

Vienna University of Technology

October 30, 2016

1 Assignment

The goal of the main task is to implement a ray tracer to render a simple 3D scene (in almost realtime) and display the result on the display attached to the DE2-115 FPGA board. All arithmetic operations should be implemented with the Q16.16 fixed point format. You don't have to take care of overflows, the test data we provide won't produce values that are out of the range of this data type.

1.1 Template

You are provided with a working (fixed point) software implementation of the ray tracing algorithm on a minimal Nios 2 system. Figure 1 shows an overview of the Qsys system of the template.

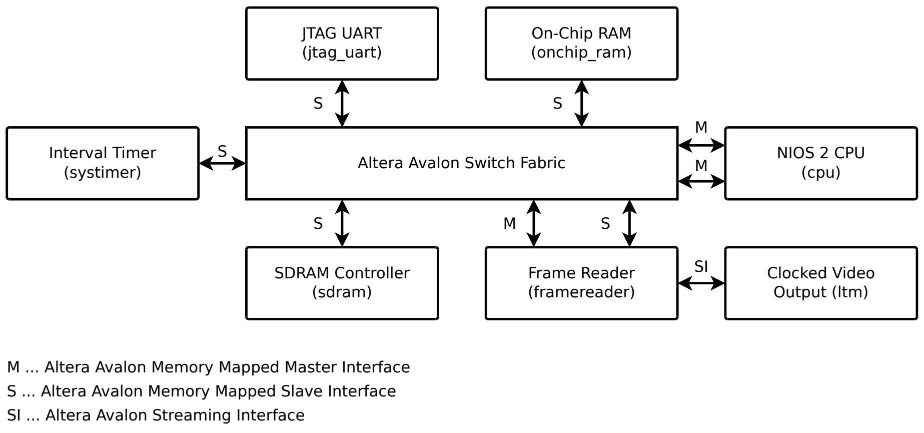
JTAG UART (jtag_uart)
On-Chip RAM (onchip_ram)
Interval Timer (systimer)
Altera Avalon Switch Fabric
NIOS 2 CPU (cpu)
SDRAM Controller (sdram)
Frame Reader (framereader)
Clocked Video Output (ltm)
S
M
SI
M ... Altera Avalon Memory Mapped Master Interface
S ... Altera Avalon Memory Mapped Slave Interface
SI ... Altera Avalon Streaming Interface

Figure 1: System Overview

- Nios 2 Processor
  The fast version of the Nios 2 processor equipped with a hardware multiplier and divider as well as data and instruction caches. You are free to change any settings of the processor, i.e. you can deactivate unneeded features to free up resources for other parts of your design.
- On Chip RAM
  150 KB of the FPGA's on-chip memory for program and data storage. Adjust the size of this memory to your needs.

- JTAG UART
  The UART interface to communicate with the `nios2-terminal`.

- Interval Timer
  This timer will be used the measure the frame rate of your solution (i.e. don't remove it from the design).

- SDRAM Controller
  The SDRAM Controller enables your system to interface the 128 MB SD RAM of the DE2-115 Board. The Controller is configured to run at 100 MHz. Note that if you change the operation frequency of your system, the parameters of this core also need to be adjusted. The SDRAM is used to hold the frame buffer(s).

- Frame Reader
  The frame reader core constantly reads the frame buffer from the SDRAM and writes the data to the Video Controller.

1.2 Software Interface

The software implementation is based on the ray tracer developed by Peter Shirley in [1]. The original source code from this book can be found on github [2]. From the Nios 2 software's point of view, the following functions are used to control the ray tracer. Currently these functions are implemented solely in software (i.e. no hardware acceleration). Your task is to provide your own implementations that interact with your hardware acceleration units. Other parts of the template software don't need to be changed. However, you are free to modify, extend or optimize them if it benefits your solution. Just keep the main loop and the ray tracer interface as they are.

- `rtInit`:

  ```
  void rtInit (uint8_t num_objects, sphere_t *spheres,
               uint16_t max_reflects, uint16_t num_samples);
  ```

  The function `rtInit` is called to initialize the ray tracer, load the scene and set the important render parameters `max_reflects` and `num_samples`. The scene is basically an array of spheres (`sphere_t`), where `num_objects` is at most 16. The parameter `max_reflects` states how may reflections of a ray the ray tracer should calculate. Your implementation should support values less than or equal 7. Finally, `num_samples` specifies the number of rays that should be cast for each pixel. The final pixel color is then determined by the average of the color values calculated for these rays. You should support all powers of two less than or equal 16. Note that the scene is completely static and won't change during calls to `rtRenderFrame` (only the camera may change).

- `rtSetCamera`:

  ```
  void rtSetCamera (vec3_t *lookfrom, vec3_t *lookat, fix16_t vfov);
  ```

  This function is used to adjust the camera's position and line of sight. It is called in between calls to `rtRenderFrame` to move the camera around the scene.

- `rtRenderFrame`:

```
void rtRenderFrame (void);
```

Finally, `rtRenderFrame` should render the 3D scene with the current camera settings into the current frame buffer.

2 Build Environment

The build environment is basically the same as for the gettoknow task. Note that we will ONLY use the makefile-based build process when we check your submission. This means that a simple make in the project directory must successfully run the build process and download the hardware and software to the board. So keep in mind that if you change the build environment or use the Eclipse IDE, this "feature" must still be functioning! If you are working on your own computer make sure that the build works on the TI lab computers! However, during the implementation process of your solution you are of free to use the graphical interface of Quartus/Qsys as well as Eclipse for software development (this can especially be useful if you what to use the debugger).

3 Submission and Grading

The goal is to render the test scene provided with the template as fast as possible. You may use all resources on the board and FPGA without restrictions.

The task is divided into two parts:

Midterm presentations In the first part we ask you to profile the provided code (e.g. gprof, gcov) to gain a good understanding of the computational effort required for each part of the calculation. Based on this, you should prepare architectural plans for your solution. You will present and we will discuss your profiling insights and solution concept in a lab appointment and additionally, you will make a slide show presentation to show them to the other groups.

As a group, make an appointment at hwsw@ecs.tuwien.ac.at for the presentation/discussion in the lab with all group members present. The deadline for the lab presentation/discussion is **12. 12. 2016**. Before coming to the appointment, upload your slides for the public presentation in MyTI. You do not need to show working HW/SW at this time.

The public (all groups) presentations will be on **13. 12. 2016**.

You can get 35 points for the lab and public presentations based on the quality and feasibility of your concept, your expertise during the discussion and the quality of your presentation.

Final submission By the end of the term, you will submit your solution and present it in a lab appointment (like the get-to-know task). Again, make an appointment each group member can attend at hwsw@ecs.tuwien.ac.at and upload your solution to MyTI 24 hours in advance.

The grading will be based on the achieved performance and the submitted solution quality. For the performance, your solution should render the scene provided with the template with the parameters `max_reflects`=5 and `num_samples`=1 and measure the rendering time. When your solution achieves x FPS, you will get

$$\max\big(0, 10(x-1)/3\big) - \max\big(0, 7(x-10)/3\big)$$

points. For code and optimization quality, you can get a maximum of 20 points.

References

[1] Peter Shirley. *Ray Tracing in One Weekend*. Amazon, 2016.

[2] Peter Shirley. Ray tracing in one weekend. `https://github.com/petershirley/raytracinginoneweekend`, 2016.